ESO-MIDAS

MIDAS Environment

Revision 1.3, June 1, 1995

Reference Number: MID–SPE–ESO–11000–0001/1.3

ESO-MIDAS: *MIDAS Environment*

Change Record:

Issue	Date	Sections affected	Remarks
1.0	1991-01-31	all	First official version
1.1	1992-11-01	all	
1.2	1994-05-01	all	Change of `long` to `int` Graphics Interfaces added Applicable from 94MAY release
1.3	1995-06-01	Chapter 6	Communication interfaces added

Contents

Preface

ESO-MIDAS [1]is the acronym for the **E**uropean **S**outhern **O**bservatory - **M**unich **I**mage **D**ata **A**nalysis **S**ystem which is developed and maintained by the European Southern Observatory. The official name, ESO-MIDAS, is a registered trademark. In this manual the name MIDAS is used as an abbreviation of ESO-MIDAS. The system is available for both VAX/VMS and UNIX systems. The design of MIDAS has been influenced by the original design of the STARLINK project in the UK (see reference [1]) as well as by the long experience gained from the IHAP system (Image Handling And Processing, see reference [2]), the first (HP based) comprehensive image processing system used at ESO for astronomical data reduction as well as for data acquisition at the telescopes.

The process of making MIDAS "portable" also benefitted from looking extensively at the other two portable astronomical image processing systems, the Astronomical Image Processing System (AIPS) and the Image Reduction and Analysis Facility (IRAF), which are used today.

MIDAS comprises two somewhat different sets of programs - the System Primitives (MIDAS programs developed by ESO software staff) to provide the basic functions of the system, and individual application programs (written by or in collaboration with astronomers according to the MIDAS standards defined below). In addition to the application programs MIDAS provides a powerful command language (the MIDAS Command Language, MCL). This command language makes it possible to create MIDAS procedures, series of MIDAS commands which look like programs. The MIDAS system from the user's point of view is described in the MIDAS User Guide which includes three volumes.

This document concentrates on the development of MIDAS application code either in FORTRAN or in C. Hence, if you are not interested in writing a FORTRAN or C program to run within the MIDAS environment, then you have time for a coffee: This document is not for you. However, if you are interested you will find in this document a description of the MIDAS environment with its various data structures and the subroutine/function calls in FORTRAN 77 or C to access these structures. It replaces and supersedes all previous versions of the Environment Document.

Chapter 1 provides a general introduction and description of the MIDAS system and its data structures. Chapter 2 gives an overview of the coding standards which should be

[1]Trade-mark of the European Southern Observatory

used when writing a MIDAS application program. Chapter 3 describes the "STandard" Interfaces to images, catalogues, keywords, descriptors, files, text, errors in a language independent way. The detailed descriptions of these interfaces for C and FORTRAN 77 are given in Appendix A. Chapter 4 describes the interfaces to the MIDAS Table File System; the detailed C and FORTRAN 77 interface description is given in Appendix B. Chapter 5 contains the graphics interfaces. Detailed descriptions of the routines, for C and FORTRAN, can be found in Appendix C. Chapter 6 describes the interfaces for connecting to a Midas server ("background Midas"); the detailed C interface description is given in Appendix D. Chapter 7 explains how to integrate and run your application in the MIDAS environment. Appendix E contains a brief description of the standard descriptors and other conventions used for images and tables while Appendix F contains a number of complete examples of MIDAS application programs using the interfaces described in this document.

If you are new to the MIDAS environment, you should read Chapter 1 and 2 completely, find out what subroutine calls are available in Chapters 3, 4 and 5, and then look carefully at the examples in Appendix F and the detailed interface descriptions in the Appendices A, B, C and D. Our intention was to write this document for you, and we hope you will find it useful.

If you are an experienced MIDAS programmer, then you probably have not bothered reading this far unless you happen to read this kind of stuff for pleasure (?). In any case, Chapters 3, 4 and 5 should serve as a reference on how to use the "STandard" Interfaces, the MIDAS Table File System Interfaces and the Graphics Interfaces. The appendices should provide the information about the exact calling sequences for either FORTRAN 77 or C.

Chapter 1

The MIDAS System

The MIDAS System consists of two major parts:

- The command language which enables the user to run programs and to communicate with the image displays and the interactive graphic devices. The image display can be a peripheral device of your mainframe (*e.g.* DeAnza, Ramtek, ...) or a window on the screen of your workstation. The graphic device may be a graphics terminal connected to your main computer (HP, Tektronix compatible, ...) or again a window if you work on a workstation;

- The program interface, *i.e.* a set of subroutines/functions described in this document, which is used by the application programs to interface to terminals, to Bulk Data Frames (images), to tables, to obtain information from the image display and graphics device as well as the command language.

The reader should distinguish between the MIDAS system described above and the MIDAS environment which is the programming "environment" in which application programs are written and which uses the interfaces described here.

MIDAS knows three basic abstract data structures:

- Keywords, are variables which can be used to pass information from one MIDAS program to the next or to store intermediate results;

- Descriptors, are variables which describe the structure of the tables and images. They can also store any other auxiliary information connected to the data like histograms, RA and DEC values *etc.*;

- Data Values.

Combining descriptors and data yield the composite data structures:

- Bulk Data Frames (usually called images), which are a set of data of physical significance. The set can have a certain dimensionality (in MIDAS up to three). The data can be stored in a wide range of formats on disk *e.g.* bytes, integers and reals. The default file extension of this data structure is `.bdf`;
- Tables, are structures for handling heterogeneous data sets which can be arranged in rows and columns. The default file extension is `.tbl`;
- Fit files, which contain the instructions for a fit process to be performed on a data structure. The default file extension is `.fit`.

These higher level structures can be organized and manipulated in a global manner by using catalogues containing four lists of data structures.

- Image Catalogues;
- Table Catalogues;
- Fit file Catalogues;
- ASCII file Catalogues.

In order to access data structures or to pass or retrieve information from the user the MIDAS Environment provides the ST and TB, for Fortran, and the SC and TC interfaces for C programs. The following data structures can be accessed using the ST and TB (SC and TC) interfaces:

- Keywords;
- Descriptors;
- Bulk Data Frames;
- Catalogues;
- Tables.

In addition, the interfaces take care of the communication between MIDAS and the application program.

A general description of these interfaces is given in Chapter 3 and 4. Appendix A and B provide the detailed interface descriptions.

Since the 94MAY release of Midas there is also a set of graphics interfaces, the PT (for Fortran) and PC (for C) interfaces included in the Midas environment.

A general description of these interfaces is given in Chapter 5, and Appendix C provides the detailed interface descriptions.

Communication interfaces which support a client/server scenario have been added with the 94NOV release. These XC interfaces are only implemented for C and only for Unix (no plans to port them to Fortran and/or VMS).
Please, note, that this is a very first version of the XC interfaces. Therefore, we reserve the right to modify the XC interfaces in the future!

A general description of these interfaces is given in Chapter 6 and Appendix D provides the detailed interface descriptions.

Chapter 2

Coding Standards for MIDAS

This chapter describes the minimum set of coding and documentation standards for MIDAS application software, and includes conventions for file names and directory structures. Code **must** fulfill this set of requirements in order to be included in MIDAS. Standards are identical for UNIX and VAX/VMS systems.

The general directory structure and conventions for file names are described in the first section. The minimum set of rules for MIDAS software is given in the next section. In the last section the requirements for general documentation are discussed.

2.1 General structure

The MIDAS system is built as a hierarchical directory tree containing installation procedures, source code and documentation. The top directory is named after the release month (*e.g.* `90NOV` for the November release of 1990) and is put directly under the MIDAS home directory. New releases can be added as additional branches under the MIDAS home directory. Both the MIDAS home directory and the top directory must be stored in environment variables[1]. `MIDASHOME` should contain the MIDAS home directory (*e.g.* `/home/ns2b/midas` for UNIX); `MIDVERS` should contain the name of the top directory of the release (*e.g.* `91NOV`).

In this chapter, the local path of a MIDAS release is abbreviated as **Mv** which is `$MIDASHOME/$MIDVERS` for UNIX or `MID_DISK:['MIDASHOME'.'MIDVERS']` for VAX/VMS where `MID_DISK` is the logical name to the disk. The UNIX style for the directory path of files is used in this chapter (*e.g.* /midas/94NOV/doc/env) rather than VAX/VMS style. In the following subsections the main directory structure and naming conventions for files are explained.

[1] In VMS, `MIDASHOME` and `MIDVERS` symbols could be inferred from your own `SETMIDAS` or `INMIDAS` definitions. In UNIX, the parameters of the command `inmidas` override the values of the environment variables `MIDASHOME` and `MIDVERS` variables.

Directory	Support	Usage
Mv/monit	A	MIDAS monitor
Mv/install	A	Installation utilities
Mv/incl	A	Header files for Core MIDAS
Mv/system	A	General system utilities
Mv/systab	A	System tables
Mv/doc	A	Documentation
Mv/libsrc	A	Source code for general MIDAS libraries
Mv/prim	A	Core MIDAS
Mv/test	A	Test and validation procedures
Mv/lib	A	Object libraries for general MIDAS
Mv/applic	B	General applications
Mv/stdred	C	Standard reduction packages
Mv/context	C	Definition of contexts
Mv/contrib	D	Contributed packages
Mv/gui	D	Graphical User Interfaces
Mv/util	E	General MIDAS related utilities

Table 2.1: Main directories in a MIDAS release version.

2.1.1 Release Directory Structure

The main directories in a MIDAS release are listed in Table 2.1 where also their support level and general content are given. The directories at the highest support level (*i.e.* 'A') are part of the MIDAS basic system or Core whereas general packages are placed at level 'B' in the subdirectory **applic**. The lower levels (*i.e.* C-E) provide additional applications and are maintained at an increasingly lower support level.

General applications are located in the two directories **stdred** and **contrib** which have the same structure described in Table 2.2. For each package, source code and documentation are placed in subdirectories named after the package, while executable files, MIDAS procedures, and object libraries are migrated to the appropriate directories during the installation.

Each of the package directories has the structure described in Table 2.3. The required subdirectories must be present and contain all information related to the package. An application package is allowed to use object libraries, procedures and include files at the level A-B only. If application programs within a package use header/include files or common subroutines, they should be placed in the appropriate directories. The source directories **src** and **libsrc** should contain makefiles which compile and link the code and place the resulting executables and libraries in the directories in the higher level directories **exec**

Directory	Usage	Remarks
exec	executable modules	filled during installation
lib	object libraries	filled during installation
proc	procedure files	links to context procedure files
help	help files	links to help files of the packages
template	template package	example of an application package
*package*1	complete code for "*package*1"	
...	code for other packages	
*package*n	complete code for "*package*n"	

Table 2.2: Subdirectories in the **stdred** and **contrib** directories.

Directory	Content	Remarks
help	Help-file for individual commands	Required
doc	User documentation in LaTeX	Required
proc	Procedure files for commands	Required
test	Test procedures and data	Required
src	Source code	Required
etc	Context file *etc.*	Optional
incl	Header file	Optional
libsrc	Source code for general package routines	Optional

Table 2.3: Subdirectories in a package directory.

and **lib**, respectively.

In general, a given directory may either contain subdirectories or files but not both. The only exceptions to this rule are the `AAREADME` and `DESCRIPTION` file which give a short description of the content of a directory and application packages. These files are also present in directories with only subdirectories.

2.1.2 File/extension names

File names in MIDAS can be any legal filename in their respective host system. They usually consist of a proper name and an extension. For easy migration between Unix and VMS systems we suggest to use mainly lowercase characters. Furthermore, to avoid possible traps in some Midas commands we suggest that filenames do not begin with a digit or special character.

The file type is indicated by an extension. The different extensions used are given in

Extension	Comment	Usage
.for	C	FORTRAN-77 source code with ESO extensions
.f	C	FORTRAN-77 source code
.inc	C	Include files for FORTRAN code
.c	/* */	source code for C programs
.h	/* */	Header files for C-code
.hlq	%	MIDAS help file for individual command
.hlc	%	MIDAS help summary for command
.prg	!	MIDAS procedure file
.ctx	!	Context files
.com	$!	VAX/VMS command files
.sh	#	UNIX Bourne shell scripts
.csh	#	UNIX C-shell scripts
.tex	%	Documentation in TEX/LATEX format indexfile!extensions
.a		UNIX object libraries
.olb		VAX/VMS object libraries
.exe		Executable files
.bdf		MIDAS Bulk Data Frames or Image files
.tbl		MIDAS table files
.fit		MIDAS fit files
.cat		MIDAS catalogue files
.mt		Data files in IHAP or FITS format

Table 2.4: General file-extension names used in MIDAS

Table 2.4 where also the comment character for the file type is indicated. The comment character must be placed in the first column of a line except for C-code file where the characters enclose the comment.

2.2 Code Standards

This section specifies the general coding standards which must be obeyed by application programs in MIDAS. These standards cannot ensure that application code is good nor correct. The individual programmer must take care that the programs are well structured and have a clear implementation.

2.2.1 General

Normal application programs written for MIDAS should be structured so that the main program handles parameter input/output and initiation of data while all computations and algorithms should be placed in subroutines. This gives the general rules for applications:

- Access to MIDAS keywords must be done in the main program only.
- All MIDAS files (*e.g.* frames, tables and catalogues) must be created, opened and/or closed in the main program only.
- Direct I/O is not allowed *e.g.* read/write on terminals. All access to devices must be done through the `ST/SC`-interfaces.
- No operating system related calls are allowed.
- No calls to operating system utilities are allowed.
- All access to MIDAS data structures must be done through the `ST/TB-` or `SC/TC`-interfaces.
- No calls to external libraries (*e.g.* `NAG`).

In general, algorithms should be separated from routines that access data. This makes it easier to re-use such algorithms in other applications.

2.2.2 Coding rules

Applications for MIDAS must be written in either the FORTRAN or C language. The specific rules for these languages are given in the following paragraphs

C code

Code written in the C language must conform to the ANSI definition of the *Programming Language C* as specified in ISO/IEC Std. 9899-1990[5]. In addition, the following rules must be followed:

- Structures and unions must be defined by `typedef` declarations.
- Structures must always be passed to or returned by functions using pointers.
- Include files may not contain executable code statements.
- Each module must contain a comment header block which defines name, purpose, author, version *etc.* (see Section 2.3.1).

- All modules using the interfaces defined in Appendix A must include the header file '`midas_def.h`' which contains standard definitions and proto-types.
- Use of proto-types is encouraged however they should be made conditional on the definition of the '`__STDC__`' variable so that code still can be complied with K&R compilers.

To make the code easier to understand, the usage of frequent comments, meaningful variable names and spaces between elements are recommended, whereas `goto` statements are strongly discouraged. The portability of the code must be verified by using the UNIX **lint** utility or a similar tool. Any potential problems reported by this utility must be corrected.

FORTRAN code

Programs written in FORTRAN must conform to the ANSI FORTRAN-77 specifications defined by ISO Std. 1539-1980(E) [6]. Five extensions may be used to improve readability and structure of the code. They can be removed by the MIDAS utility **esoext**. The five extensions are:

- **IMPLICIT NONE** statements must be specified in each file in order to ensure all variables are declared explicitly.
- **INCLUDE** statements may be used to include files with common definitions, etc.
- **!-comments** may be used to add comments at the end of a statement line. All characters including and after an !-mark on a line will be regarded as comments. The usage of such comments is strongly encourage because it significantly improves the documentation.
- **DO/ENDDO** statements may be used for convenience instead of the standard FORTRAN-77 DO/CONTINUE statements.
- **Long identifiers** with up to 16 character including underscores may be used. The first five and the last character (excluding underscores) must yield a unique identifier.

The usage of these extensions is not mandatory; however, the three first extensions are recommended in order to ensure that all variables are declared, definitions are common in all modules and that the code is fully documented with in-line comments.

The FORTRAN code must obey the following rules:

- All variables must be declared explicitly with type. This may be ensured by using the `IMPLICIT NONE` statement or using a compiler option which checks this.

- Indentation of at least two columns must be made for each level of `IF/ELSE/ENDIF` and `DO/END/CONTINUE` statements. For long `DO`-loops, additional comments at their start and end are strongly encouraged. Further, statement labels must clearly indicate the loop structure and hierarchy.
- Statement labels should be given in increasing order and reflect the logical structure of the program. The number range `80000-81000` should be avoided because it is used by the MIDAS preprocessor.
- When using `DO/CONTINUE` statements, it is recommended to terminate the statement label after the `DO` with a '`,`'(ref. ANSI FORTRAN-77) to avoid misinterpretation even through typing errors.
- Include files should not contain executable code statements.
- Each module must contain a comment header block which defines name, purpose, author, version *etc.* (see Section 2.3.1).
- Subroutines and functions may share the source file with a main program only if they are not used in any other module.
- Internal `READ` statements using free or list-directed format are not allowed (*e.g.* `READ(LINE,*)`). The general usage of internal `READ` is deprecated due to usage of explicit editing formats.

To make the code easier to understand, the usage of frequent comments, meaningful variable names and spaces between elements are recommended whereas `GOTO` statements are strongly discouraged.

A number of FORTRAN-77 statements and features have been defined obsolete by FORTRAN-90. Although they are still valid in FORTRAN-90, they may be recommended for deletion in new revisions. The usage of features on the FORTRAN-90 list of obsolescent features is deprecated for MIDAS application written in FORTRAN in order to ensure the longest possible life time of the code. The current list of obsolescent features in FORTRAN-90 is:

- arithmetic `IF`;
- branching to an `ENDIF` statement from outside its block;
- real and double precision `DO` variables and control expressions;
- shared `DO` termination, and `DO` termination on a statement other than on a `CONTINUE` and an `ENDDO` statement;
- `ASSIGN` and the assigned `GOTO`;
- alternate `RETURN`;
- `PAUSE` statement; and
- assigned `FORMAT` specifiers.

It should be noted that all 5 ESO extensions to FORTRAN-77 are included in FORTRAN-90.

Library	Content	Usage
Mv/lib/libmidas.a	Standard MIDAS object library	All
Mv/lib/libagl3.a	AGL routines	System, available
Mv/lib/libidi.a	IDI routines	System, available

Table 2.5: General MIDAS libraries.

MIDAS procedures

Each MIDAS command must have a procedure, written in the MIDAS Control Language (MCL), which handles the command line parameters and defines the keywords used by the actual application coded. The procedure will execute the basic modules (*i.e.* using the `RUN` command). The rules for MIDAS procedures are:

- Command line parameters must be defined by the `CROSSREF` command which must be the first command.
- Parameters must be defined by a `DEFINE/PARAMETER` command. These commands must follow immediately after the `CROSSREF` command.
- Local variables must be declared with the `DEFINE/LOCAL` command.
- Each module must contain a comment header block which defines name, purpose, author, version *etc.* (see Section 2.3.1).
- Indentation of at least two columns must be made for each level of `IF-THEN/ELSE/ENDIF` and `DO/ENDDO` statements.
- All MIDAS commands must be specified completely, *i.e.* 6 characters for the command verb and 4 characters for the qualifier.

2.2.3 Library references

The MIDAS system provides a number of library modules which may be used by application programs. They are grouped in object code libraries and are generated during the installation procedure. These libraries and their usage are given in Table 2.5. Normal applications should use the standard Midas library, e.g. `libmidas.a` for Unix, which includes all the interfaces described in this document. Although usage of the IDI- and AGL-libraries is possible, it is not recommended as these are very low level libraries. They do not ensure proper updates of MIDAS data structures which must be done explicitly. The OS-routines are also included in the standard library but are not finalized yet, i.e. not documented. Therefore, they may change in the future!

Useful routines may be found in the `Mv/libsrc/...` and `Mv/applic/.../libsrc` libraries. However, they are maintained at a lower level and may not be documented adequately.

2.2.4 Installation

The installation of an application package is done by running `make` in the subdirectories **libsrc**, **src**, **help**, **proc** and **etc**. In each of these directories a **makefile** must exist with the instructions (for VAX/VMS a **makefile.com** file is used). The format and typical content of these **makefile**'s can be found in the **contrib/template** directory.

2.2.5 Code Documentation

Proper documentation and comments in the application code itself are essential. As an absolute minimum of documentation in the code, each module must contain the elements below:

- Each code module or file must start with a header comment block that at least contains the required entries defined in Table 2.6.
- Any routine or function within a code module must contain a routine comment block immediately after the routine declaration statement. This block must include the required routine entries defined in Table 2.6.
- All external or global variables (*e.g.* `COMMON`'s) must be identified in the module comment block.
- All formal variables in routines or functions must be declared in separate lines with comments indicating their usages and if they are used for input, output or both.
- Each major section of the code must be preceded by a comment describing what the section does.

Usage of meaningful names of variables and functions is recommended to improve readability of the code. Although the exact amount of comments compared to the code will depend on the actual module, a general guideline is that approximately 30% of a module should be comments.

2.3 Documentation

It is very important that application programs not only execute correctly but that they are also well documented. The minimum set of documentation required includes: a) header and comments in all source code files, b) help-files for each MIDAS command, and c) a general user manual. The format of these three items is given in the subsections below.

2.3.1 Code headers

All printable files must start with a header comment block. This block must have a comment character in the first column (see Table 2.4) except for C-language source code

Label	Usage	Remarks
`+++++++++++`	Start of comment block	Required
`.COPYRIGHT`	Copyright statement	Required, Header
`.IDENTifier`	File name of module	Required, Header
`.MODULE`	Type of module	
`.AUTHOR`	Name of creator	Required, Header
`.KEYWORDs`	Subject keywords	Required
`.LANGUAGE`	Language of module	Required, Header
`.PURPOSE`	General purpose of module	Required
`.COMMENTs`	Comments	Optional
`.VERSION`	Version no., date, etc.	Required, Header
`.RETURNS`	Return values for functions	Required, Routine
`.ENVIRONment`	Special environment requirements	Required
`------------`	End of comment block	Required

Table 2.6: Section labels for documentation in text files.

where it must be enclosed by the comment characters. The header block must contain the required entries given in Table 2.6. The label must start just after the comment character (for C-code in the first column within a comment). The label may be truncated as indicated by lower case characters. If an entry exceeds one line it may be continued on the next. Such lines must either be indented or repeat the section label. In program source code additional comment blocks are required after each routine or function declaration (see Section 2.2.5).

The general layout of the section can be seen in the example for a User Manual given below. The `VERSION` section must have the format indicated, namely: version no., date, short description and initials of person making the change. New `VERSION` sections must be added each time a modification is made. Old `VERSION` sections must be maintained so that the full history of the module is present.

2.3.2 Help-file syntax

Each command in MIDAS must have its own help-file which can be used by the on-line help facility. The name of the file is constructed from the command name and qualifier. The name is in lower case and consists of the first six characters of the command name with the first four characters of the qualifier appended. If the command name is shorter than six characters, an underscore '_' is inserted to separate the two fields. The help files have the extension '.hlq'. They contain a number of sections with different help information. The format and an example of a help file for the command `READ/DESC` is given in the

Appendix G.

2.3.3 AAREADME and DESCRIPTION files

Each directory must have an `AAREADME-file` which contains a short description of the individual files. The `AAREADME-file` should normally have one line for each file. The line must start with the file name followed by a short description. If one line is not sufficient, more lines may be used. Additional lines must, however, be indented. A text file `DESCRIPTION` containing a short description of the package must be present in each application package directory under the **stdred** and **contrib** directories. This file should be in plain ASCII format and be at most 20 lines long.

2.3.4 User manuals

A complete user description of the package must exist in the `doc` subdirectory. The documentation must be written in the LaTeX language using the *report*-style for possible later inclusion in the MIDAS Users Guide. The general format is indicated below:

```
 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 %.COPYRIGHT (c)  1991  European Southern Observatory
 %.IDENT     applic.tex
 %.LANGUAGE  LaTeX
 %.AUTHOR    Your Name, Institute
 %.PURPOSE   Documentation for application package
 %.VERSION   1.0  1991-Jan-22 : Creation, YN
 %--------------------------------------------------------------------
 %\documentstyle[11pt,twoside]{report}     % remove first '%' for print
 %\begin{document}                         % remove first '%' for print

 \day=22  \month=1  \year=1991             % release date for chapter

% \setcounter{page}{1}
 \chapter{Application package}
 \label{ap:application}
 This chapter describes the application package ...

 \section{Introduction}
 \label{ap:introduction}
 The package provides a wide set of utilities for ...

 ...
 ...
```

```
\cleardoublepage
%\end{document}                                    % remove first '%' for print
```

This file can be printed with LaTeX by removing the comment character '%' from the documents lines as indicated above.

Chapter 3

Standard Interfaces for Application Programs

This chapter describes the "Standard Interfaces" to the basic MIDAS data structures MIDAS data structures which are needed in an application program in order to execute within the MIDAS environment.

The routines consist of a name of six characters (all capital letters), where the first two characters are either `ST` for FORTRAN 77 or `SC` for the C language. The next character indicates the data structure the function it accesses or the subsytem it operates on. Currently these are:

- `S` for connecting to the MIDAS system (environment);
- `E` for operating on the error subsystem in MIDAS;
- `K` for accessing the keywords;
- `T` for displaying text;
- `F` for accessing bulk data frames;
- `D` for accessing the descriptors;
- `C` for manipulating catalogues;
- `I` for accessing data frames and their standard descriptors;

The final three characters describe the functionality of the routine.

Below and in the same order we give a general description and explanation of the various interfaces. Refer to Appendix A for the detailed calling sequence of each routine mentioned below. Appendix A contains two parts: first, all C routines, grouped by functions, are listed and then follow all FORTRAN routines.

3.1 Symbolic names

Whenever parameters serve as flags, *i.e.* can only take a limited number of values, symbolic names have been defined (e.g. F_IO_MODE) in order to hide the explicit value from the application. These definitions are stored in a central Midas directory (logical name `MID_INCLUDE`).

To obtain these definitions use: #include <midas_def.h> for C-routines.
In FORTRAN code use: INCLUDE MID_INCLUDE:ST_DEF.INC
and INCLUDE MID_INCLUDE:ST_DAT.INC to get the necessary translations. The `INCLUDE` statement is not in standard FORTRAN 77 but it will be processed via the ESO utility **esoext**, cf. chapter 7.

3.2 Access to the MIDAS Environment

3.2.1 Initialize and connect to the environment

Setting up the MIDAS environment for an application program is done via a call to `STSPRO` (in FORTRAN 77), or `SCSPRO` (in C). This *Prologue* routine performs the following functions:

- initialize all internal tables needed by the environment;
- connect to the keyword area;
- open the MIDAS logfile;
- enable automatic abort on errors.

`STSPRO` should be called once in the main routine before any other references to the Midas environment are made.

3.2.2 Disconnect from the environment

A call to `STSEPI` (`SCSEPI`) will disconnect the calling program from the environment, i.e.:

- update keywords in the global keyword area
- release all data frames and tables
- update any used catalog
- close the MIDAS logfile

Furthermore, `STSEPI` will stop your application, so this should be the last call in the main routine of your MIDAS application.

3.3 Error Processing

3.3.1 Control exception and error handling

As a default any application is aborted if an error occurs in any of the `ST` or `TB` interfaces. Thus, the application program is free from numerous statements like:

```
IF (ISTAT.NE.IOK) THEN ...
```

which so often clutter the code of an application.

However, there are occasions where the application needs to examine any possible error and decide on the further flow of control. This is achieved via a call to `STECNT` (`SCECNT`). This error control routine lets you decide about continuing after an error, logging in the MIDAS logfile, and displaying of an error message.

3.3.2 Signal an error from a subroutine

If you want to pass an error (non-MIDAS) from one of your subroutines to the calling module use `STESIG` (`SCESIG`) . It will also ensure that this error is recorded in the logfile.

3.3.3 Display error message and abort

Usually, your application is embedded in a MIDAS procedure (see Chapter 7). Besides your application, there may be more commands or applications in that procedure as well. In most cases it is no use continuing the procedure if any of its embedded applications have failed. In order to abort your application as well as all the procedures which are involved in the execution of this application, use the interface `STETER` (`SCETER`). This interface will display an error message and return to the highest level, i.e. the MIDAS user prompt.

Note

All following interface routines return an integer value as status. Values less than zero indicate a warning message, greater than zero an error and equal to zero is a successful return.

3.4 Keywords

All the parameters a program needs to fulfill its purpose, other than the raw (e.g. image, table) data and their descriptors, are called keywords in MIDAS. The keywords are implemented as a global data structure in the environment and thus provide communication among different (main) programs. Keywords consist of a name, a type and a number (max. 32767) of associated data values of the given type. The keyword name is a character string of maximum 8 alpha-numeric characters (case insensitive) starting with a letter. The keyword type is either 'Integer', 'Real', 'Double Precision', or 'Character', for `INTEGER, REAL, DOUBLE PRECISION, CHARACTER` (in FORTRAN 77) or `int, float, double, char` (in C). They fall into 4 distinctive groups:

a) **MIDAS reserved keywords.** These keywords are used by MIDAS system programs and are generally of no concern to the application programmer. Great care should be taken when modifying these keywords!

b) **Fixed keywords.** These keywords are fixed in their meaning, size and data type in order to obtain some standardization (otherwise one might end up having many different keywords describing the same parameter). Application programs may read and write these keywords.

c) **User-defined keywords.** These keywords are used by application programs to define and handle parameters not covered by the fixed keywords. Application programs may define, delete, read and write user-defined keywords as needed for their purposes.

d) **Local keywords.** These keywords are used like local variables in MIDAS procedures. Local keywords are only known within the context of the particular procedure where they are defined (and optionally in the procedures called from it). In case of doubly defined names, the local keyword takes precedence. Application programs cannot tell a local from a global keyword.

3.4.1 Read value(s) from keyword area

A keyword's value is obtained by calling `STKRDC` (`SCKRDC`) for character keywords, `STKRDD` (`SCKRDD`) for double precision keywords, `STKRDI` (`SCKRDI`) for integer keywords, and `STKRDR` (`SCKRDR`) for real keywords. These routines will return data from a given keyword which may have been written previously by the MIDAS Monitor or any other application.

If you want to prompt and get the keyword value(s) via the terminal keyboard, use the routines `STKPRC` (`SCKPRC`), `STKPRD` (`SCKPRD`), `STKPRI` (`SCKPRI`), and `STKPRR` (`SCKPRR`) for the appropriate keyword type. These routines will display a prompt string on the terminal, wait for user input and update the contents of that keyword. If the user just hits RETURN, the current contents of the keyword (together with a warning, no error) are returned to the application. However, interactive Keyword input via `STSPRx` should only be used in applications that are used in an interactive manner or for debugging purposes.

Note

The interfaces dealing with character keywords have an additional parameter specifying the number of bytes (elements) per character array element. For the other keyword types this is a fixed number:
Four bytes per element are used with integer and real keywords, and eight bytes with double precision keywords.

3.4.2 Write value(s) to keyword area

To modify the contents of a keyword, call STKWRC (SCKWRC) for character keywords, STKWRD (SCKWRD) for double precision keywords, STKWRI (SCKWRI) for integer keywords, and STKWRR (SCKWRR) for real keywords.

If the keyword does not exist, these routines return an error. No interface exists to create a new keyword; that has to be done at the command language level.

Note

Again, the interface for character keywords has an additional parameter as discussed above.

3.4.3 Get information about keywords

Two routines may be used to get information about keywords of which you only know the name or its position (index) in the keyword data base. These are STKFND (SCKFND) if you know the keyword name and STKINF (SCKINF) if you use the position of the keyword.

3.4.4 Creating keywords

In order to ensure the integrity of the internal MIDAS keyword data base, there are no interfaces to create a new keyword. Instead, you have to use the relevant MIDAS commands (e.g., WRITE/KEY or COPY/KEY - see Chapter 3 Vol.A of the MIDAS User Guide).

Note

Many keyword interface routines use a UNIT and a NULL parameter. These parameters are currently not used, but just serve as placeholders, so that you will not have to change your code once we implement that functionality in the MIDAS environment.

3.5 Text

Application programs do not interact with the user via direct FORTRAN (or C) I/O statements. Input is obtained via keywords and for output only the ST interfaces should be used in order to permit consistent logging of all MIDAS I/O.

3.5.1 Display text on terminal

For displaying messages and results use STTPUT (SCTPUT). This routine will write a line of text to the user's interactive terminal, as well as to the logfile if logging is enabled (see STECNT). The output is written as a character string followed by a carriage return and line feed. Numeric values must be converted by use of the 'internal file' mechanism of

FORTRAN-77 or some other mechanism. Below is an example for using the FORTRAN-77 'internal file' mechanism:

```
      INTEGER STATUS, IVAL
      CHARACTER*80 OUTPUT
      ...
      WRITE (OUTPUT,100) IVAL
100   FORMAT('The value is', I6)
      CALL STTPUT(OUTPUT, STATUS)
      ...
```

3.5.2 Display text with special characters on terminal

If your text has special requirements like backspaces, you should use `STTDIS` (`SCTDIS`) instead of `STTPUT`. This text will not be recorded in the MIDAS logfile.

Note

The text buffer in the interfaces above is not limited to 80 characters. In fact, the routines split the character buffer into chunks of the size of your MIDAS terminal or window.

3.6 Bulk Data Frames

In MIDAS any array of data is called a *Bulk Data Frame* (BDF) or image. These images may be accessed in two different ways: via *mapping* or via *direct disk I/O*.

Mapping: Working on a computer with a virtual memory system makes it possible to access data entirely within a program array instead of reading it in a portion at a time from a file. This is done efficiently in the VAX/VMS environment by mapping files directly into the program's virtual memory (the file is not physically read from disk, only some pointers are changed). The maximum size of a mapped `BDF` depends (in the VAX) on the system parameter `VIRTUALPGCNT`.

On UNIX systems the mapping mechanism is generally not available and is emulated in the relevant ST interface and not as efficient as in VMS. The maximum size for mapping depends upon the maximum size of virtual memory which can be allocated.

The advantage of mapping is that the application programmer need not care how to get his/her data from a disk file to memory and vice versa - the relevant interface routines will do that task. All the image data may be viewed as a large matrix inside the program space.

Disk I/O: Getting data in and out via direct disk I/O is fast and efficient in both the VMS and the UNIX environment. However, the application program itself has to take care of the managing of the data I/O. This may be quite heavy and even more complex than the actual algorithm itself ...

Therefore, we recommend for the "normal" application programmers to use the mapping interfaces to get their data in and out. Then you only have to debug your algorithm and not the data I/O management as well. Furthermore, if you are not satisfied with the performance of your application, you can still move to direct disk I/O (but your algorithm may have to change, too).

3.6.1 Create a Bulk Data Frame on disk

To create an image frame use `STFCRE` (`SCFCRE`). This routine will create the file on disk and store the relevant information in an internal MIDAS file control table. A 'file id' (an integer number) is returned which is used in all following references to that frame.

The data values (pixels) of a Bulk Data Frame are stored by default as floating point (real) numbers using 32 bits per pixel. However, your application may use bytes (8 bits), unsigned short integers (16 bits), integers (32 bits) or double precision (64 bits) per pixel as well.

Bulk Data Frames and their associated Descriptors are stored together as a normal (unformatted) file on disk. We refer to this file as a Bulk Data Frame as well. The internal format of this file is not described in this document, because one should only access these files via the Standard Interfaces to ensure consistency and backward compatibility of different versions.

3.6.2 Open a Bulk Data Frame

Opening an existing image frame is done via a call to `STFOPN` (`SCFOPN`). This routine will open the file on disk and store the relevant information in the internal MIDAS file table. Again a 'file id' is returned for later usage by other interfaces.
If an application opens a frame with a different data type than the one with which it was created, the data are converted automatically to the format specified in the File Open routine. When writing the data back to the frame it is converted again to the original data format. Except a few specialised tasks, all MIDAS core applications open image frames with the real data format.

Access Mode

By default, all frames are opened in MIDAS with *write* access. If this fails, the frame is opened with *read* access. Subsequent modifications of a file with *read* access only will fail, however. Therefore, you define via the file protection how a Bulk Data Frame may be accessed inside MIDAS, i.e. via `SET PROTECTION` (VMS) or `chmod` (Unix).

3.6.3 Close a Bulk Data Frame

Use `STFCLO` (`SCFCLO`) to close a frame on disk. Also the entry in the internal MIDAS file table is cleared and any active catalogue is updated. The *Epilogue* routine `STSEPI` closes all opened frames (and tables) automatically.

If that frame was mapped `STFCLO` also releases the virtual memory that was allocated for the data of that file.

3.6.4 Delete a Bulk Data Frame

To delete a Data Frame on disk in a portable (i.e for VMS and UNIX) way use `STFDEL` (`SCFDEL`). If the file is still open it will be closed first in the way described above for `STFCLO`.

3.6.5 Rename a Bulk Data Frame

To rename a Data Frame on disk in a portable way use `STFRNM` (`SCFRNM`). This interface will not work if the file is still opened inside MIDAS.

3.6.6 Get information about a Bulk Data Frame

To obtain some internal information about a frame (e.g. the data format of the pixels) use `STFINF` (`SCFINF`).

3.6.7 Map a Bulk Data Frame for reading and/or writing

If you want to access your image data via mapping use `STFMAP` (`SCFMAP`). This routine 'maps' an existing Bulk Data Frame (or parts of it) into virtual memory for reading and/or writing. The technique used to map data returns a pointer which is definitely not very FORTRAN like. Look at the examples in Appendix F to see how this problem is solved for FORTRAN programs (for C there is no problem anyway).

If the file has been opened using a different data format than the original format, data are converted automatically.

3.6.8 Unmap a Bulk Data frame

Use `STFUNM` (`SCFUNM`) to unmap data which have been mapped before by a call to `STFMAP`. No further reference can be made to the data of this frame within the program. The Descriptors of the Bulk Data Frame are still accessible via the 'file id'.

If necessary, the data are converted back to the original format automatically.

3.6.9 Disk I/O for reading and/or writing data

If you want to read/write image data from/to disk use the `STFGET` (`SCFGET`) and `STFPUT` (`SCFPUT`) interfaces.

Like with the mapping/unmapping interfaces the data are converted, if necessary.

Note

You should not mix the two modes of accessing image data. That means, do not e.g. use STFMAP (SCFMAP) and STFGET (SCFGET) calls together for a frame.

3.7 Descriptors

The descriptive information associated with Bulk Data Frames (this is equivalent to the header of an IHAP file) is stored in a data structure called *Descriptors* in MIDAS. The concept is similar to the FITS header data (see reference [3]).

Like keywords, descriptors consist of a name (max. 15 characters, case insensitive) and associated data values of any length (max. 32767) and of type 'Integer', 'Real', 'Character', or 'Double Precision'. The descriptor name may contain alpha-numeric characters and underscore. Descriptors are not global parameters, but are connected to a specific data frame. A minimum set of descriptors must exist for every data frame. These 'Standard Descriptors' are explained in Appendix E for Bulk Data Frames and Tables.

Note

All descriptor handling interfaces use the 'file id' that is returned by STFCRE or STFOPN as well as the file identifier (called 'table id' there) returned by TBTINI and TBTOPN.

3.7.1 Read descriptor value(s)

Obtaining values from a descriptor of a frame is done via a call to `STDRDC` (`SCDRDC`) for character descriptors, `STDRDD` (`SCDRDD`) for double precision descriptors, `STDRDI` (`SCDRDI`) for integer descriptors and `STDRDR` (`SCDRDR`) for real descriptors.

Note

The interfaces dealing with character descriptors have an additional parameter specifying the number of bytes (elements) per character array element. For the other descriptor types this is a fixed number:
Four bytes per element are used with integer and real descriptors and eight bytes with double precision descriptors.

3.7.2 Write descriptor value(s)

Modifying the contents of an existing descriptor or creating a new descriptor and its associated data is achieved via `STDWRC` (`SCDWRC`) for character descriptors, `STDWRD` (`SCDWRD`) for double precision descriptors, `STDWRI` (`SCDWRI`) for integer descriptors and `STDWRR` (`SCDWRR`) for real descriptors. Contrary to keywords, which have a fixed size, the descriptor data area may be extended at any time.

Note

Again the interface for character descriptors has an additional parameter as discussed above.

3.7.3 Delete a descriptor

`STDDEL` (`SCDDEL`) deletes a descriptor of a frame.

3.7.4 Get descriptor information

Two routines can be used to get information about each descriptor that is present in a data frame. If you use `STDINF` (`SCDINF`), information about the descriptor at a given relative position within the frame's descriptor list is returned. If you know the name of a descriptor, you'll find out more about it via `STDFND` (`SCDFND`).

3.7.5 Copy a descriptor

`STDCOP` (`SCDCOP`) copies some or all descriptors of a frame to another frame.

Note

Many descriptor interface routines use a UNIT and a NULL parameter. These parameters are currently not used, but just serve as placeholders, so that you will not have to change your code once we implement that functionality in the MIDAS environment.

3.8 Catalogues

MIDAS catalogues are best described as a list/collection of any of the data structures *images, tables* or *fitfiles*. If a catalogue is enabled (cf. Chapter 3 of the MIDAS Users Guide, Volume A) new entries of the relevant data structures are automatically added. Entries in a catalogue may then be referenced also via that entry number instead of the physical file name.

The MIDAS command language provides the functionality to exercise any single application or complete MIDAS procedures on all entries in a catalogue instead of only an individual frame. Therefore, it is usually not necessary to handle catalogues in a user application. Only in specific cases, e.g. where the input to your algorithm is a whole list of images, are the catalogue interfaces needed (an example of this is the program to read/write FITS tapes).

3.8.1 Create a catalogue

To create a catalogue for a given type of MIDAS data structures use `STCCRE` (`SCCCRE`). Catalogues are implemented as simple ASCII files to make it easy to access them outside the specific MIDAS commands.

3.8.2 Add/Subtract an entry to/from a catalogue

Adding/subtracting entries to/from a catalogue is done via `STCADD` (`SCCADD`) and `STCSUB` (`SCCSUB`). These routines are only necessary if the catalogue is not the currently enabled catalogue for a given data structure.

3.8.3 Get entries from a catalogue

You can obtain the next sequential entry in a catalogue by calling `STCGET` (`SCCGET`).

3.8.4 List the contents of a catalogue

To list the entries of a catalogue use `STCLIS` (`SCCLIS`). Catalogues are implemented as simple ASCII files to make it easy to access them also outside the specific MIDAS commands.

3.8.5 Find an entry in a catalogue

To find a given entry in a catalogue use `STCFND` (`SCCFND`).

3.8.6 Get information about a catalogue

Obtaining the size and type of a catalogue use `STCSHO` (`SCCSHO`).

3.9 Images

Besides tables, BDF files (images) are the most important MIDAS structure. Therefore, we included 'high-level' interfaces, using the interfaces described earlier, to handle images, i.e., the data and the connected descriptive information, in a very simple and compact way. These routines use the 'mapping' mechanism to access the data, which may not be the most efficient way in UNIX systems (see the discussion above). However, the following interfaces provide an easy and painless way to manage MIDAS image files in an application. In fact, many algorithms could be integrated into MIDAS by just using the 'image' interfaces and `STSPRO` and `STSEPI`.

3.9.1 Read image frame and standard descriptors

To read the standard descriptors and get all image data into memory use `STIGET` (`SCIGET`).

3.9.2 Create image frame and standard descriptors

To create an image frame with its standard descriptors use `STIPUT` (`SCIPUT`). This routine also allocates the memory space for the image data but does not initialize it.

Note

Because the STIGET (SCIGET) and STIPUT (SCIPUT) routines use the mapping technique for accessing the images, you should not mix them with STFGET (SCFGET) or STFPUT (SCFPUT) calls for a frame.

Chapter 4

Table Interfaces

MIDAS has a very extensively developed table data structure system that is useful for handling data derived from various applications and can be used as a general analysis tool. This chapter describes the subroutine calls which interface to tables and which an application program needs to read from or to write to a MIDAS table.

The names of these routines consist of six letters (capital letters), where the first two characters are either `TB` for FORTRAN 77 or `TC` for the C language. The next character indicates a table object as indicated below, and the final three letters describe the action of the routine.

The table objects, defining the third letter are:

- `A` for operations on a table array;
- `T` for opening and closing a table file;
- `C` for operations on a table column;
- `E` for operations on a table element;
- `R` for operations on a table row;
- `I` to get general table information;
- `F` for the column format;
- `L` for the column label;
- `U` for the column unit;
- `S` for the column selections;
- `K` for reference column.

Standard operations on Tables are:

- initialization of the table data, either by creating a new table or by opening an existing table;
- element and row access;
- sorting and searching;
- editing of the table descriptor;
- handling of selection column.

The following sections give a general description of the different Table interface routines. The detailed calling sequences are included in appendix A. A program example showing how to use the routines described in this section is given in Appendix A.2 of this document.

Information about using Tables from the MIDAS command language is included in the "MIDAS User's Guide" [4] Chapter 3.

4.1 Table File Access

In order to create a new table four steps have to be carried out:

1. The table has to be initialized (*i.e.* allocated) as a disk file;
2. Each table column has to be defined by specifying its type, format, unit and associated label;
3. The data can then be written into the corresponding columns and rows;
4. The table has to be closed.

As for the creation of a new table, access to an existing table also involves four steps:

1. The table has to be opened;
2. The table column numbers corresponding to the given labels have to be found;
3. The data can be read or written;
4. Finally, the table has to be closed.

4.1.1 Create a new table

A table file can be created on disk using the routine `TBTINI` (`TCTINI`). This routine defines the size of the file as well as the physical organisation of the table. The size of the file is defined by the number of rows and the number of words per row. Tables are physically stored on disk in two formats: by records, corresponding to the natural way of storing the rows sequentially, and transposed, where all the values of a single variable – a column – are stored together. The routine returns the file identifier.

4.1.2 Open an existing table

Opening an existing table file is done with the routine `TBTOPN` (`TCTOPN`). The routine returns the file identifier for later use by other routines.

4.2 Table Columns

4.2.1 Create one or more columns

The creation of new columns in a table is done with the routine `TBCINI` (`TCCINI`). Column type, format, label, units of the associated information as well as the output format are defined at this moment. The routine returns the column index to be used in later references to the column. If the required space for the new table exceeds the allocated column space, the table is structured on disk with some extra space for the new column(s).

4.2.2 Column labels and numbers

To find the correspondence between the column numbers and their labels the routine `TBLSER` (`TCLSER`) has to be used.

4.2.3 Delete a column

Columns can be deleted from a table with the routine `TBCDEL` (`TCCDEL`).

4.3 Table Elements, Arrays and Rows

The data are read or written into the corresponding columns and rows either element by element, or by complete rows.

4.3.1 Read and write a Table Element

Access to table elements is done via the column index and row number. Elements in a table can be integer, float, double precision and character string data types, as defined when the corresponding column is created. Implicit conversion between the internal data type and the argument type of the input/output routine is done automatically. Elements are accessed sequentially. The search routines `TBESRx` (`TCESRx`) used in connection with sequential access allow direct access by value. Access to subfields for elements of CHARACTER type is not supported. Automatic conversion is performed if the type of argument in the routine does not correspond to the column type.

To read an element from the table, use the routine `TBERDx` (`TCERDx`), where `x` defines the type of argument as character string `C`, single precision floating point number `R`, double precision `D`, or integer `I`.

To write an element into the table, use the routine `TBEWRx` (`TCEWRx`), where `x` defines the type of argument as before.

An element can be deleted, i.e., replaced by an undefined value, with the routine `TBEDEL` (`TCEDEL`).

4.3.2 Read and write Table Array Elements

Access to array elements is done by column, row index and length. The length is taken as the maximum possible when this argument is zero. element arrays in the table can be integer (I*1, i*2 or I*4), float, double precision or character string data types, as defined during the creation of the columns. Implicit conversion of the element value is done automatically if the input/output is done with a routine not corresponding to the data type. To read an extract of an array element from the table use the routine `TCARDx` where `x` defines the type of the argument as character string `C`, single precision floating point number `R`, double precision `D`, or integer `I`. To write an extract of an array element into the table use the routine `TCAWRx` where `x` defines the type of the argument as before. A array element can be deleted, i.e., replaced by null values with the routine TCADEL.

4.3.3 Read and write a Table Row

For a complete row the routine `TBRRDx` (`TCRRDx`) is available. Here again, the x defines the argument format. Of course, one can also overwrite existing data values in the table or extend the table with new columns. To do this, the routine in the previous subsection can be used. Access to table rows is done via a set of column indices and the row number. Implicit conversion between the internal data types and the argument type of the input/output routine is done automatically.

To read a row from the table, use the routine `TBRRDx` (`TCRRDx`), where `x` defines the type of argument as character string `C`, single precision floating point number `R`, double precision `D`, or integer `I`.

To write a row into the table, use the routine `TBRWRx` (`TCRWRx`), where `x` defines the type of argument as before.

A row can be deleted, i.e., all the elements in that row replaced by an undefined value, with the routine `TBRDEL` (`TCRDEL`).

4.4 Table Descriptors

Tables contain a set of auxiliary information defining the data stored in the table. This information defines the general characteristics of the table, like physical size of the table, and also column related parameters like units, external format and column label. The auxiliary information is stored as MIDAS descriptors in the table file.

General information about the physical size of the table as well as the current number of columns and rows is obtained through the routine `TBIGET` (`TCIGET`).

Columns can be accessed by labels, which can be regarded as variable names at the command level. Labels are character strings (up to 16 characters) defined by the user when the column is created with `TBCINI` (`TCCINI`). Valid characters are letters, digits, and the underscore symbol (_). It is possible to read the label of a column using the routine `TBLGET` (`TCLGET`) and to update the column label with the routine `TBLPUT` (`TCLPUT`). Correspondence between label and column index is found with the routine `TBLSER` (`TCLSER`).

Each column in a table has an associated unit defined when the column is initialized. This unit can be read using the routine `TBUGET` (`TCUGET`) and can be updated with the routine `TBUPUT` (`TCUPUT`). The external format, used to display the information in a column, is controlled by the column format. In the current version, this format is defined in FORTRAN 77. The routine `TBFGET` (`TCFGET`) is used to read the format while the routine `TBFPUT` (`TCFPUT`) will update it.

4.5 Select a Table Subset

A query to a table in MIDAS, using the command `SELECT/TABLE`, will generate a subset of the original table. In the current version, the selected subset is defined by a flag associated with each row in the table that is handled in a consistent way by all the system commands.

In order to maintain this consistency, it is recommended that new application programs accessing the information in a given row, will read this flag, using the routine `TBSGET` (`TCSGET`), to test if the corresponding row belongs to the current – active – subtable.

The flag corresponding to a given row is updated with the routine `TBSPUT` (`TCSPUT`).

To select all rows in the table use the routine `TBSINI` (`TCSINI`).

4.6 Sorting and Searching

A table can be sorted according to the ascending/descending values in a given column with the routine `TBCSRT`.

Searching for an element in a given column is done with the routine where `x` defines the type of the argument. `TBESRx` (`TCESRx`).

4.7 Close a Table

The routine `TBTCLO` (`TCTCLO`) is used to close a table file.

Chapter 5

Graphics Interfaces

5.1 Introduction

The MIDAS graphics package is designed to enable data representation in a reasonable flexible way. It can be used for graphical display of most of the MIDAS internal data structures: bulk data frames, table files, keywords, and image descriptors. The software consists of four layers:

- main programs and procedures, executed by the MIDAS graphics commands;
- high level MIDAS Graphics Interfaces;
- low level MIDAS Graphics Interfaces;
- low level Astronet Graphics Library (AGL).

This chapter is focused on the MIDAS Graphics Interfaces, a well defined set of high level graphics routines. These routines should be used by MIDAS programs for graphical applications. The name of the routines consist of six characters (all capital letters), from which the first two determine the language. These letters can be either `PC` (for C) or `PT` (for Fortran 77). The following four characters indicate the functionality of the routine.

Currently, the standard Graphics Interfaces cover the following functions:

- set up of graphics characteristics;
- open a device, set the clip values, etc.;
- plot the coordinate box;
- plot data;
- plot text;
- get the graphic cursor;
- close the graphics device and plot files.

This set of interfaces is intended to cover the basic needs for doing software development for graphical data representation in MIDAS. The lower level AGL library can be used to get more flexibility. However, when using this library, information about the graphics (e.g. graphics setup, coordinate box, etc.) is not stored in the MIDAS keywords structure. Hence, mixing calls to the MIDAS Graphics Interfaces and calls to the AGL library should be avoided.

In the following sections a general description of the Graphics Interfaces is presented. The detailed calling sequences are included in Appendix C. Two program examples, one in C and one in Fortran, showing how to use the Graphics Interfaces are included in Appendix F.

Information about using the graphics facilities in MIDAS can be found in the ESO-MIDAS Users Guide, Volume 1, Chapter 6.

5.2 Set up the graphics characteristics: graphics keywords

The graphics characteristics is controlled by a number of MIDAS graphics specific keywords. The contents of these keywords and hence the graphics setup can be read and written with a specific set of interface routines:

- `PCKRDC` and `PCKWRC` (`PTKRDC` and `PTKWRC`) for character graphics keywords;
- `PCKRDI` and `PCKWRI` (`PTKRDI` and `PTKWRI`) for integer graphics keywords;
- `PCKRDR` and `PCKWRR` (`PTKRDR` and `PTKWRR`) for real graphics keywords.

The MIDAS graphics keyword structure can be accessed in two ways. First, you can control the graphics setup in the same way as if you are using the command `SET/GRAPH`. In that case you are dealing with the keywords `PLCSTAT`, `PLISTAT` and `PLRSTAT`. However, changing these keywords within our application program means that, like with `SET/GRAPH`, you are changing the user setup of the graphics system. To avoid this problem the setting of the coordinate box, scales, and offsets can be done via the keyword `PLRGRAP`. This keyword can be manipulated without changing the user setup. `PLRGRAP` also contains information about which part of a MIDAS image has been plotted. In case you plot image data using the high level graphics interfaces you should set these values correctly, in order to ensure correct output from the high level cursor routine.

The keyword `PLCMETA` contains the name of active plotfile. The keyword `PLCDATA` contains information about the data plotted. Although the content is not vital for the performance of the graphics package itself, programmers are advised to fill these keywords properly.

The values in the graphics keywords are not accessed by number but, similar to the `SET/GRAPH` command, by name. Below, you will find an overview of all relevant graphics keywords in combination with the names of the various entries, and their default values.

The routine `PCKDEF` (`PTKDEF`) can be used to reset the keywords: `PLCSTAT`, `PLISTAT` and `PLRSTAT` to their default values.

function: PCKRDI (PTKRDI) and PCKWRI (PTKWRI)				
keyword	parameter	length	default	description
PLISTAT	PMODE	1	2	pmode mode (ranging from 0 to 2)
	FONT	1	0	font used (ranging from 0 to 6)
	LTYPE	1	1	line type (ranging from 0 to 6)
	STYPE	1	5	symbol type (ranging from 0 to 21)
	LWIDT	1	1	line width (ranging from 1 to 4)
	TWIDT	1	1	text width (ranging from 1 to 4)
	COLOUR	1	1	colour (ranging from 0 to 8)
	BCOLOUR	1	0	background colour (0 to 8)
function: PCKRDR (PTKRDR) and PCKWRR (PTKWRR)				
keyword	parameter	length	default	description
PLRSTAT	XAXIS	4	0,0,0,0	x axis frame limits and ticks
	YAXIS	4	0,0,0,0	y axis frame limits and ticks
	ZAXIS	4	0,0,0,0	z axis frame limits and ticks
	XSCALE	1	0.0	scale set by user in x
	YSCALE	1	0.0	scale set by user in y
	ZSCALE	1	0.0	scale set by user in z
	XOFFSET	1	-999.0	offset in mm set by user in x
	YOFFSET	1	-999.0	offset in mm set by user in y
	SSIZE	1	1.0	symbol size multiplier
	TSIZE	1	1.0	text size multiplier
function: PCKRDC (PTKRDC) and PCKWRC (PTKWRC)				
keyword	parameter	length	default	description
PLCSTAT	BINMODE	4	OFF	bin mode: ON or OFF
	COLMODE	4	SUB	colour mode: Sub or Xor
	CLEARGRA	4	ON	erase for plot commands or over plot
	DEBUG	4	OFF	debug mode: ON or OFF
	FRAME	4	AUTO	AUTO scaling or SQUAre frame
	XFORMAT	4	AUTO	format for values along the x axis or NONE
	YFORMAT	4	AUTO	format for values along the y axis or NONE
	ZFORMAT	4	AUTO	format for values along the z axis or NONE
PLCMETA	PLNAME	80	unknown	name of plot file
PLCDATA	DNAME	60	unknown	name of data stucture
	DTYPE	20	unknown	type of data

function: PCKRDR (PTKRDR) and PCKWRR (PTKWRR)				
keyword	parameter	length	default	description
PLRGRAP	XWNDL	4	0,0,0,0	x axis frame limits and ticks
	YWNDL	4	0,0,0,0	y axis frame limits and ticks
	ZWNDL	4	0,0,0,0	z axis frame limits and ticks
	SCALE	3	0,0,0	scale parameters in x, y, z
	PIXEL	4	0,0,0,0	pixel area of BDF file plotted
	OFFSET	2	-999,-999	offset parameters in x and y

5.3 Open a device

`PCOPEN` (`PTOPEN`)reads the MIDAS graphics keywords and initialises the device using this keyword setup. The device limits are set according to the currently assigned device (with the command `ASSIGN/GRAPHIC`). The keyword setup determines:

- plot boundaries limits,
- plot scales,
- line width,
- line type,
- symbol dimension,
- symbol type,
- text dimension,
- foreground colour,
- background colour.

If the access mode is "new plot" `PCOPEN` (`PTOPEN`) opens a new plot file or overwrites an existing one with the same name. If the access mode is "over plot" `PCOPEN` (`PTOPEN`) appends to an existing plot file.

5.4 Plot a coordinate box

A coordinated box (axes) can be drawn with the routine `PCAXES` (`PTAXES`) or `PCFRAM` (`PTFRAM`). `PCFRAM` (`PTFRAM`) uses the device setting done by `PCOPEN` (`PTOPEN`) and you will have to give the values for the big tick marks and the small tick marks (no defaults!). `PCFRAM` (`PTFRAM`) is used in all MIDAS application programs. Hence, the axes drawn by `PCFRAM` (`PTFRAM`) in your application program will look identical to the axes drawn by using the MIDAS command `PLOT/AXES`.

`PCAXES` (`PTAXES`) uses also the device setting done by `PCOPEN` (`PTOPEN`), but it adjusts the scaling so that the quotes, labels and title will fit into the device viewport. The big tick marks and the small tick marks are calculated and this routine offers a lot of nice options.

Note that `PCAXES` (`PTAXES`) has to be called before plotting any data or text, while `PCFRAM` (`PTFRAM`) may be called before or after plotting the data.

After having the coordinate box drawn by one of the axes routines the graphics system is set to the so-called `USER` mode, the user (world) coordinate system. All subsequent plotting activities are assumed to take place in this coordinate system and hence all coordinates should be in user coordinates. To switch to the `NORM`alized (clip) coordinate system (that runs from 0 to 1 in both the x- and y-direction of the graphics device) one has to use the specific AGL routine `AGSSET`.

5.5 Plot data

Data points can be presented either by symbols or by connecting lines with the routine `PCDATA` (`PTDATA`). The coordinates of the data ponits are expected to refer to the current graphic mode, *i.e.* `USER` (world) coordinates or `NORM`alized (clip) coordinates. `PCDATA` (`PTDATA`) connects the data points with lines if the line type is not equal to zero, else it will present the data points with symbols. `PCDATA` (`PTDATA`) produces a simple staircase if the bin mode flag is set.

Data can also be represented by a histogram using `PCHIST` (`PTHIST`). It offers some sophisticated options so you can draw a histogram according to your taste.

5.6 Plot text

It is often very useful to draw additional information in a plot, for example time of creation, type of data, source of data, etc. This information can be retrieved from MIDAS keywords and drawn in the current viewport using `PCTEXT` (`PTTEXT`).

5.7 Get the graphics cursor

`PCGCUR` (`PTGCUR`) gets a graphics cursor and returns coordinates referring to the current active coordinate system. The pixel area of the BDF file and coordinate frame have to be set before calling `PCGCUR` (`PTGCUR`), see section 5.2. If the requested position is outside the clipping area a warning message will be given. Note, the key code returned by the function is device dependent.

5.8 Close the viewport

`PCCLOS` (`PTCLOS`) closes all open meta files and AGL operations.

Chapter 6

Communication Interfaces

Besides the usual interactive usage of MIDAS it should also be possible to employ the services and utilities provided by MIDAS from other processes. These processes (the clients) reside either on the same Unix system as the process executing the MIDAS session (the server) or on a different system in a network. Therefore, a set of "Communication Interfaces" has been implemented which is described in this chapter.

The routines consist again of a name of six characters (all capital letters), where the first two characters are `XC`. The next character indicates the functionality which is addressed by the routine. Currently these are:

- `X` for establishing, controlling and closing the connection to a MIDAS server;
- `C` for sending commands to a MIDAS server;
- `K` for accessing the keywords of a MIDAS server;

The final three characters describe the functionality of the routine.

Note

These interfaces are only implemented in the C language and for the Unix environment. It is, also, the very first implementation of these interfaces. Therefore, we reserve the right to modify the calling sequence of them in the future, if necessary.

Below and in the same order we give a general description and explanation of the various interfaces. Refer to Appendix D for the detailed calling sequence of each routine mentioned below.

The functionality supported by these interfaces is provided to the interactive user of MIDAS via the commands `CONNECT/BACKGROUND`, `DISCONNECT/BACKGROUND`. The help text of these commands gives more detailed information about the communication among different MIDAS sessions as well as pointers to related commands.

6.1 Connecting to a MIDAS server

There are two ways of connecting a MIDAS client (which may be a MIDAS session or a separate process) to a MIDAS server (which we also call *background MIDAS*) already running:

- via simple ASCII files
- using Unix sockets

Using ASCII files limits the connections between client and server to the same host or different hosts sharing disks. On the other hand we can bridge different host systems, like e.g. Unix and Basic based systems. Furthermore, the *background* MIDAS can be used interactively as well.

Connections based on sockets can be established on the same host or over a network but are limited to Unix systems supporting sockets. The *background* MIDAS session has to be set in a special background mode (via `SET/BACKGROUND sockets`) where it's waiting on input via sockets and thus not available for interactive usage anymore.

The routine `XCZOPN` will establish a connection to a MIDAS server based on ASCII files, whereas `XCXOPN` will use sockets as the transport mechanism for the connection with the server. `XCZOPN` or `XCXOPN` must be called before any of the other communication interfaces can be used, and assume that a background MIDAS has been started already. Furthermore, for `XCXOPN` the command `SET/BACKGROUND sockets` must have been executed in the *background* Midas session before.

The interface `XCXSTA` checks the status of the *background* MIDAS and returns a *busy* or *idle* state of the MIDAS server.

With `XCXCLO` the connection to a MIDAS server is closed from the client side.

If a background MIDAS session is not running yet, one can start a MIDAS session, set it into background mode and connect to that MIDAS session via the `XCXINI` and `XCZINI` interfaces. Within `XCXINI` the routine `XCXOPN` is used and `XCZOPN` with `XCZINI`. Currently, it is only possible to start a background MIDAS on the same host.

In case of a socket based connection we can get the file descriptor of that socket via `XCXGFD`.

With the interface `XCXSTP` the connection is closed and also the connected MIDAS session is terminated.

6.2 Executing Commands

Application programs can send complete MIDAS commands to a *background* MIDAS via the `XCCSND` interface. The application may or may not wait for the execution of these commands. In case the command was sent without wait to the MIDAS server, the `XCCWAI` interface can be used to synchronize the application by explicitely waiting for the termination of the last command sent.

6.3 Accessing Keywords

A client can read and write the keywords of the *background* MIDAS via the `XCKRDx` and `XCKWRx` interfaces. They provide the same functionality as the corresponding `SCKRDx` and `SCKWRx` interfaces described in chapter 3.

6.4 Building a MIDAS Client Application

Since your application does not have to be integrated in the MIDAS environment no specific requirements for compiling your program exist. That also means, that you can skip chapter 7. You only need to include the MIDAS Communication library `libxcon.a` when linking your application. Like all other MIDAS libraries `libxcon.a` is stored in `$MIDASHOME/$MIDVERS/lib`.
Thus, compiling and linking a MIDAS client module would be:

```
% cc -c myclient.c
% cc myclient.o -L$MIDASHOME/$MIDVERS/lib -lxcon -o myclient.exe
```

An example of an interactive MIDAS client program can be found in Appendix F.

6.5 On-line MIDAS

The communication interfaces are a major part of the requirements of the ESO VLT project for `On-line MIDAS`, the version of MIDAS to be used at the VLT. The additional requirements, like. e.g. parallel displays or search paths for MIDAS procedures and data files, are described in Volume A, Chapter 3 of the MIDAS Users Guide.

Chapter 7

Integrating a MIDAS Application

7.1 Introduction

As described in chapter 2, MIDAS application programs must fulfill a minimum set of coding and documentation requirements. In addition to these standards, we have a number of rules and recommendations concerning the usage of keywords, descriptors, among others.

In this chapter you can find which rules should be kept in mind in order to run your application successfully. In Section 7.2 we give some general hints concerning the use of keywords and descriptors. In Section 7.3 and Section 7.4 the usage of table interfaces and of error routines is briefly discussed. Section 7.5 we describe the general MIDAS libraries. Finally, in Section 7.6 a number of examples for the integration of Fortran and C code are given.

7.2 General Hints

7.2.1 Include files and the common block `VMR`

Your MIDAS application code will certainly contain some calls to MIDAS interfaces to set up the MIDAS environment and possibly to get access to MIDAS data structures. In order to use these interfaces (ST and TB for FORTRAN; SC and TC for C) successfully, the program should contain two include files which declare the system global variables and assign values to these variables. The declaration and assignments for FORTRAN programs [1]are in the files:

`MID_DISK:['MIDASHOME'.'MIDVERS'.INCL]ST_DEF.INC` and `ST_DAT.INC` (in VMS)
or
`$MIDASHOME/$MIDVERS/incl/st_def.inc` and `st_dat.inc` (in Unix)

In your FORTRAN application you always refer to these files in uppercase (even in Unix). The ESO preprocessor will take care of that; see the example code in Appendix F. Also, if you write your application code in FORTRAN 77, you must include the very important common block `VMR`. This common block acts as an emulation of the pointer mechanism, which is well known in C. Using the array `MADRID`, the addresses of the input and output data arrays are passed to subsequent routines. Refer to Appendix F to see what the code looks like.

For C programs you have to include the file
`MID_DISK:['MIDASHOME'.'MIDVERS'.INCL]midas_def.h` (in VMS)
or
`$MIDASHOME/$MIDVERS/incl/midas_def.h` (in Unix)

7.2.2 Fixed Keywords

The keyword area is divided into so-called reserved (only used by the MIDAS system; you may read them but do not update them!), fixed and user-defined keywords. The reserved as well as the fixed keywords are stored with their initial default values in the ASCII file `MID_DISK:['MIDASHOME'.'MIDVERS'.MONIT]SYSKEYS.DAT` for VMS[2] and `$MIDASHOME/$MIDVERS/monit/syskeys.dat` for UNIX. When starting the interactive monitor via `INMIDAS` in VMS (or `inmidas` in UNIX) this file is read in and the keywords initialized accordingly. Look at this initial keyfile and see if a keyword which fits your purposes already exists. For example, if you need a character keyword to hold the name of your output frame, use the fixed key `OUT_A` instead of creating a new one, since `OUT_A` just serves that purpose. Thus you save space in the keyword data base and standardize your program. Another example are the keywords `INPUTI, INPUTR, INPUTC` and `INPUTD`, which you should use whenever possible to hold the input values for your program.

For all the internal calculations you need inside your procedure, you should always use local keywords (see Chapter 3 of the MIDAS Users Guide for details).

7.2.3 Keywords OUTPUTI, OUTPUTR, OUTPUTC, OUTPUTD

Any output values that are calculated/displayed in your program and not stored in a keyword or descriptor should be written into the appropriate keyword `OUTPUTx` before terminating to give any other program the chance to pick these values up again. Remember that the keywords form a common data base for all programs in the MIDAS environment.

[1] In VMS, the logical name `MID_DISK` refers to the disk where the MIDAS system is stored and MIDASHOME to the main directory. In UNIX, the variable MIDASHOME refers to the root directory of MIDAS. The symbol MIDVERS holds the MIDAS version, e.g. 91NOV in VMS and UNIX.

[2] In VMS, there is no distinction between file names in upper or lower case. In UNIX there is a difference! Therefore, the convention in MIDAS is to always use `lower case` for filenames (which you may type in as upper case in VMS).

7.2.4 Descriptor HISTORY

The descriptor HISTORY is a standard descriptor even though it is not in the calling sequence of STIPUT or STIGET. If you work on an input frame and produce a new output frame always update this descriptor. E.g.:

```
CALL STIGET(INFRAME, ..., IMFIN, ISTAT)
CALL STIPUT(OUTFRAME, ..., IMFOUT, ISTAT)
....  compute ...
CALL DSCUPT(IMFIN, IMFOUT, MYHIST, ISTAT)
```

where MYHIST is a character expression of 80 characters; e.g. filled via `MYHIST(1:) = 'subframe of'//INFRAME//' '` if you just extract a subframe from the input frame. The routine DSCUPT is also included in the MIDAS library (see below).

7.2.5 Descriptor LHCUTS

Another important descriptor is the real descriptor LHCUTS (6 values). It contains the user specified low + high cuts (which are e.g. used in a LOAD/IMAGE command) in element 1 + 2 and the physical minimum and maximum of the data frame in element 3 + 4 (the elements 5 + 6 are used in some special applications only). When you enter your data frames into the system via INTAPE, the descriptor values LHCUTS(3,4) are set up correctly. When you write your own conversion program make sure to fill LHCUTS correctly. All MIDAS core commands operating on frame data update LHCUTS(3,4) accordingly. The MIDAS commands STATISTICS/IMAGE or FIND/MINMAX will also store the right values into descriptor LHCUTS.

7.3 Table Rules

The following rules apply to programs using tables:

- It is necessary to initialize the table files via the routine TBTINI (TCTINI) or read the table using TBERDx or TBRRDx (TCERDx, TCRRDx) before accessing the tables.
- Up to 16 tables can currently be active in an application program.
- The maximum size of tables in the current version is 256 columns. There is no upper limit in the number of rows except for the total size of the table being limited to about 25 Mbytes, equivalent to 6 millions real*4 elements.

7.4 Error Handling

Each MIDAS interface routine calls an error routine before returning to the calling program. If the status is not zero, this routine will display the appropriate warning or error message on the user's terminal.

Whether errors lead to an abort or are handled by the program itself, is controlled via calls to `STECNT`.

There are several calls to the operating system via the OS interfaces within the MIDAS subroutines. Any error status encountered will force the execution of a system service exception handler, resulting in the display of various information on the user's terminal and abort of the application program. If the cause of this error is not obvious the error should be reported to the Image Processing Group at ESO, e.g. via the MIDAS command `REPORT/PROBLEM`) or, preferably, via the Feedback mechanism of the Motif based Graphical User Interface of the MIDAS Help (started via `CREATE/GUI help` in MIDAS).

7.5 MIDAS Libraries

The MIDAS system contains a number of library modules, each of them containing object code for a particular part of the system.

The most important one is the library that contains the modules needed for setting up the MIDAS Environment for your application. This library is:
`MID_DISK:['MIDASHOME'.'MIDVERS'.LIB]LIBMIDAS.OLB` (VMS),
or
`$MIDASHOME/$MIDVERS/lib/libmidas.a` (UNIX).
This library also contains the standard and table interfaces, as well as the FORTRAN to C routines.

For VMS `MID_DISK:['MIDASHOME'.'MIDVERS'.LIB]LIBGEN.OLB` contains a library of useful subroutines for MIDAS programs (`$MIDASHOME/$MIDVERS/lib/libgen.a`for UNIX). These subroutines are written specifically for the MIDAS environment, i.e. they use MIDAS interface routines themselves and assume correct data will be found in descriptors like `LHCUTS`. They perform a variety of functions such as pixel conversion, minmax calculation, subframe extraction, etc. In addition, the library `LIBDISP.OLB` (`libdisp.a`) contains a number of useful high level display routines.

You may use any of the routines in these libraries but beware, we do `NOT` guarantee that these routines remain unchanged from one MIDAS release to the next! That is also the reason why these routines are not documented, e.g. their calling sequence has to be retrieved from the header info.

It is planned to build up a library of *stable* high level routines which will then be documented in a similar way as the 'standard interfaces'.

7.6 Integrating a MIDAS Application

In this section we will describe how to integrate a user written application into the MIDAS environment. MIDAS application code can be written in Fortran or in C. For simplicity, we assume that the application does not interact with an image or graphics display; however, it may contain subroutine calls to the standard interfaces (ST interfaces for Fortran code, SC interfaces for C) or table interfaces (TB for Fortran, TC for C).

In order to create the executable module, the application code, which may consist of one or more modules has to be compiled to object module(s). Hereafter, these object module(s) must be linked with the MIDAS interface library which contains the standard interfaces (ST and SC), the table interfaces (TB and TC), "FORTRAN to C" interfaces and the OS interfaces. In addition to the MIDAS interface library some other libraries may be needed, possibility including a non-MIDAS library. The MIDAS interface library as well as some other MIDAS libraries all reside in the 'MIDAS library' subdirectory (see Section 7.5.

The OS interfaces provide the layer between MIDAS applications and the operating system. They are used internally in the standard and table interfaces and should not be used in the normal application programs.

The MIDAS primitives, i.e. the basic building blocks of the system written by ESO software staff, are stored in `MID_DISK:['MIDASHOME'.'MIDVERS'.PRIM.'utility'.SRC]` for VMS and `$MIDASHOME/$MIDVERS/prim/'utility'/src` for UNIX, with 'utility' either `GENERAL, DISPLAY, PLOT` or `TABLE` for VMS (`general, display, plot` or `table` for Unix). Use these programs as a standard and don't hesitate to copy them. Make sure you change the name if you copy and modify any of these modules!!!!

In the next two subsections you will find examples which show how to compile and link MIDAS application programs. The first example concerns a Fortran program, the second a C program. For both applications the compile and link sequence will be given for VMS and UNIX.

7.6.1 Fortran example

The first example is the application program `SMOOTH`. This program is written in FORTRAN 77 (with the ESO defined extensions); the complete code may be found in Appendix D.1.1. We assume that the program code is stored in the file `smooth.for`.

The compilation and linking for VMS is then:

```
$ FORTRAN SMOOTH
$ LINK SMOOTH,MID_DISK:['MIDASHOME'.'MIDVERS'.LIB]LIBMIDAS/L
```

For UNIX we first have to apply the ESO precompiler to produce a file `smooth.f` which can then be used for compilation and linking as follows:

```
% $MIDASHOME/$MIDVERS/system/exec/esoext.exe \
          -I$MIDASHOME/$MIDVERS/incl -f smooth.for
% f77 -c smooth.f
% f77 smooth.o -L$MIDASHOME/$MIDVERS/lib -lmidas -o smooth.exe
```

Note

Remember, that the precompiler expects the FORTRAN source code in upper case, in order to work correctly...

7.6.2 C example

In this example the application program `copymi` is compiled and linked. The program is written in C. We assume that the code is stored in the file `copymi.c`. The code can also be found in Appendix D.1.1

The compilation and linking for VMS is then:

```
$ CC /INCLUDE:([‘MIDASHOME’.‘MIDVERS’.INCL]) COPYMI
$ LINK COPYMI,MID_DISK:[‘MIDASHOME’.‘MIDVERS’.LIB]LIBMIDAS/L
```

For UNIX the sequence is similar and goes as follows:

```
% cc -I$MIDASHOME/$MIDVERS/incl -c copymi.c
% cc copymi.o -L$MIDASHOME/$MIDVERS/lib -lmidas -o copymi.exe
```

7.6.3 Running a MIDAS application

There are two ways to execute the program. The first one is simply to enter the MIDAS command:

```
Midas nnn > RUN smooth
```

The other (preferable) possibility is to edit a small MIDAS procedure (say `myproc.prg`) which prepares your input, e.g. setting default values, and actually executing your program inside MIDAS as above via the `RUN` command. This could be, say the last line of that procedure (See the MIDAS Users Guide, Chapter 3, Volume A for an in-depth discussion of MIDAS procedures). The program is executed when you activate the procedure, i.e. by entering the command:

```
Midas nnn > @@ myproc
```

Note

If you are executing in the VMS environment be careful not to mix the MIDAS `RUN` command with the DCL `RUN` command. E.g.

```
Midas nnn > $RUN smooth
```

Would also start the execution ot the module smooth.exe but no keywords would be set correctly.

7.6.4 Debugging a MIDAS application

Since your MIDAS application is written in a standard programming language, you can also use the standard debugger on your host system to debug that application. First, you compile and link your program with the debugger. E.g. for a Fortran program in Unix:

```
% $MIDASHOME/$MIDVERS/system/exec/esoext.exe \
          -I$MIDASHOME/$MIDVERS/incl -f smooth.for
% f77 -g -c smooth.f
% f77 smooth.o -L$MIDASHOME/$MIDVERS/lib -lmidas -o smooth.exe
```

Then you enter the MIDAS command `DEBUG/MODULE` and thereafter all modules executed from MIDAS are started with the host debugger enabled so you can inspect the source code, set breakpoints, etc. With the DEBUG/MODULE command you also turn the debug mode in MIDAS off, check the help for details.

Since there may be different debuggers available (on Unix systems, usually) the *debugger of choice* is saved in the keyword `MID$SYS` - this is done during the Midas installation. If you want to use a different debugger you have to modify that keyword appropriately.

7.7 Integrating a MIDAS Context

If you want to add a whole set of related applications, an application package, you have to add it as a new MIDAS context.

First, you have to add a subdirectory with all your code, helpfiles, etc. to the **contrib** subdirectory of MIDAS as described in chapter 2.

Furthermore, you need a context file which is a MIDAS procedure where you could create own command definitions, write out some information about the package and (most importantly) indicate to the help facility where the help files of that package are found.

As an example we use the current context file of the MIDAS context `geotest`:

```
!
! ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
! context file  geotest.ctx
! K. Banse ESO - IPG 901212
!
! ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
!
! indicate origin of this context for the MIDAS help facility
!HELP = contrib
!
WRITE/OUT Setting context GEOTEST.
WRITE/OUT
```

```
!
CREATE/COMM CREATE/RAMP @c geotest,ramp
CREATE/COMM CREATE/WAVE @c geotest,wave
CREATE/COMM CREATE/SPC1 @c geotest,spec1
CREATE/COMM CREATE/SPC2 @c geotest,spec2
CREATE/COMM CREATE/SPC3 @c geotest,spec3
CREATE/COMM CREATE/ART_IMAGE @c chaos
!
WRITE/OUT "The following commands are added:"
WRITE/OUT
WRITE/OUT CREATE/RAMP, CREATE/WAVE
WRITE/OUT CREATE/SPC1, CREATE/SPC2, CREATE/SPC3
WRITE/OUT CREATE/ART_IMAGE
WRITE/OUT
WRITE/OUT "Use HELP for a detailed explanation of these commands"
WRITE/OUT -
"Refer also to the extensive documentation in the MIDAS Users Guide, volume B"
WRITE/OUT
```

To make the context available to other MIDAS users at your site you have to store this context file in the directory `MID_DISK:['MIDASHOME'.'MIDVERS'.CONTEXT]` (in VMS) or `$MIDASHOME/$MIDVERS/context (in Unix)`.

The last step is to integrate the help files for that package: For each command/qualifier combination there should be a `.hlq` file and for each command a `.hlc` file written according to the standards described in chapter 2. These help files have to be stored (e.g. for Unix) in the directory `$MIDASHOME/$MIDVERS/contrib/package_name/help`.

In order to also add the command names to the main help display once this context has been enabled you have to add one more ASCII file, named `alll.hlz`, containing the names of all the new commands to this directory.

Look into the help directory of e.g. the context `geotest` for examples of the files mentioned above.

Remember that before a user can use the commands of a context, that context has to be enabled, which is done via the MIDAS command: `SET/CONTEXT package_name`. More details about contexts can be found in chapter 3 of the MIDAS User's Guide, Volume A.

Bibliography

[1] Starlink User Note 4.1 (1980). *An Interim Environment for the Development of Starlink Application Software.* Rutherford and Appleton Laboratories.

[2] Middleburg, F. (1980). *IHAP User's Manual.* ESO - Garching.

[3] Wells, D.C. and Greisen, E.W. (1979). *FITS: A Flexible Image Transport System.* Kitt Peak National Observatory.

[4] Image Processing Group, ESO Operating Manual No. 1 (1993) *Midas User's Guide.* ESO - Garching.

[5] ISO/IEC JTC1, ISO/IEC 9899-1990 *Programming Languages-C*, Geneva.

[6] ISO 1539-1980(E) *Programming Language Fortran*

Appendix A

Standard Interfaces

Module *Version [3.10] (17-Jan-1994)* **Module SCC**

Author: K. Banse, ESO - Garching *Language:* C

Classification: MIDAS catalogues

Environment:
- VMS and UNIX
- #include <midas_def.h>

Comments:

This module contains the routines to interact with the catalogs of MIDAS. The functions provided are:

- Create catalogs (SCCCRE)
- Add and remove entries in catalogs (SCCADD, SCCSUB)
- Find entries and get sequential entries in a catalog (SCCFND, SCCGET)
- Display contents of catalogs (SCCLIS, SCCSHO)

Main arguments used by the routines are:

catfile name of catalog

Function *SCCCRE*

SCCCRE(catfile,type,flag)

char *catfile	*IN: catalog file (terminated with '\0')*
int type	*IN: type of files to use, same as in SCFCRE*
int flag	*IN: = 1, if file 'dirfile.dat' exists*

Purpose: Create a catalog.

Returns: return status (0 = o.k.)

Algorithm: If flag = 1, read each filename from ASCII file 'dirfile.dat', create an entry for it in the catalog, read relevant descriptors of each file and store in catalog file, else create an empty catalog.

Function *SCCADD*

SCCADD(catfile,name,ident)

char *catfile	*IN: catalog file*
char *name	*IN: name of frame to be added*
char *ident	*IN: IDENT info of frame to be added*

Purpose: Add catalog entry.

Returns: return status (0 = o.k.)

Algorithm: Add an entry with given name and identification field to the catalog.

Function *SCCSUB*

SCCSUB(catfile,name)

char *catfile	*IN: catalog file*
char *name	*IN: name of frame to be removed*

Purpose: Remove entry with given name from catalog.

Returns: return status (0 = o.k.)

Algorithm: Search for given entry and change first 2 chars to '//', which makes this entry unused.

Function *SCCGET*

SCCGET(catfile,flag,name,ident,no)

char *catfile	*IN: catalog file*
int flag	*IN: = 0 for name only, = 1 for name and identifier*
char *name	*OUT: name of frame in next entry of catalog*
char *ident	*OUT: identifier of that frame*
int *no	*IN/OUT: current/next entry no. as input/output*

Purpose: Get next sequential entry from a catalog

Returns: return status (0 = o.k.)

Algorithm: Return name and identifier of next entry in catalog file;
return a blank in name[0], if at EOF.
Parameter 'no' has to be set to 0 in the first call of the routine!

Function *SCCLIS*

SCCLIS(catfile,intval)

char *catfile	*IN: catalog file*
int intval[2]	*IN: listing interval [low,hi]*

Purpose: List contents of catalog file.

Returns: return status (0 = o.k.)

Algorithm: Read and display catalog, entry by entry.
The listing interval specifies entry no.s of the catalog.

Function *SCCFND*

SCCFND(catfile,frmno,frame)

char *catfile	*IN: catalog file*
int frmno	*IN: entry no. in catalog*
char *frame	*OUT: corresponding frame name (terminated by \0)*

Purpose: Find corresponding frame for given entry no.

Returns: return status (0 = o.k.)

Algorithm: Search for given entry no. and return connected frame name.

Function *SCCSHO*

SCCSHO(catfile,noent,last)

char ∗catfile	*IN: catalog file*
int ∗noent	*OUT: no. of entries*
int ∗last	*OUT: last entry no.*

Purpose: Show size and type of a catalog

Returns: return status (0 = o.k.)

Algorithm: Read and count active entries of catalog.

Module	*Version [3.10] (17-Jan-1994)*	**Module SCD**

Author: K. Banse, ESO - Garching *Language:* C

Classification: standard interfaces, descriptors

Environment:
- VMS and UNIX
- #include <midas_def.h>

Comments:

This module contains the routines to interact with the descriptors of MIDAS frames.
The functions provided are:

- Find descriptors and get information about them (SCDFND, SCDINF)
- Copy and delete descriptors (SCDCOP, SCDDEL)
- Read contents of descriptors (SCDRDC, SCDRDD, SCDRDI, SCDRDR)
- Write contents of descriptors (SCDWRC, SCDWRD, SCDWRI, SCDWRR)
- Read contents of char. descriptors in a simplified way (SCDGETC)

Main arguments used by the routines are:

imno file id of related MIDAS frame

descr name of descriptor, max. 15 chars. long and must begin with an alphabetic char.

felem first element to be accessed, the very first element is indicated by 1 (not by 0 as in C-arrays) for compatibility with the relevant FORTRAN interfaces.

maxvals maximum number of elements required.

actvals actual number of elements accessed, value returned by the routine.

unit unit of descriptor, currently not implemented - just used as a placeholder for now.

null no. of null values in descriptor, currently not implemented - just used as a placeholder for now.

Function *SCDCOP*

SCDCOP(from,to,mask,dsc)

int from	*IN: file id of source frame*
int to	*IN: file id of destination frame*
int mask	*IN: copy_mask*
char *dsc	*IN: name of descr. to copy (if mask = 4)*

Purpose: Copy descriptors from one frame to another.

Returns: return status (0 = o.k.)

Algorithm: Depending on 'mask' the relevant descriptors are read from the source frame and written to the destination frame.

Remarks: Mask for copying:
1 - copy all descriptors,
2 - copy only standard descriptors (cf. Appendix E),
3 - copy all but standard descriptors,
4 - copy single descriptor specified by par. 'dsc',
5 - copy all but extended list of standard descriptors

Function *SCDDEL*

SCDDEL(imno,descr)

int imno	*IN: file id of data frame*
char *descr	*IN: descriptor name (max. 15 chars.)*

Purpose: Delete a descriptor.

Returns: return status (0 = o.k.)

Algorithm: The descriptor directory is scanned for the specified name and if found, that name is marked as deleted.

Function *SCDFND*

SCDFND(imno,descr,type,noelem,bytelem)

int imno	*IN: file id of data frame*
char *descr	*IN: descriptor name (max. 15 chars.)*
char *type	*OUT: type of descr - I, R, C, D or ' ' (a blank)*
int *noelem	*OUT: no. of elements*
int *bytelem	*OUT: no. of bytes per element*

Purpose: Provide info about descriptor 'descr'.

Returns: return status (0 = o.k.)

Algorithm: Search descriptor directory, if 'descr' is not found, *type is set to a blank.

Function *SCDINF*

SCDINF(imno,npos,fno,buf,lbuf,numbuf)

int imno	*IN: file id of data frame*
int npos	*IN: position of descriptor, beginning with 1*
int fno	*IN: specify desired info*
char *buf	*OUT: buffer for character descriptor info*
int lbuf	*IN: max. length of buffer above*
int *numbuf	*OUT: return buffer for numerical data*

Purpose: Provide info about descriptor at position 'npos'.

Returns: return status (0 = o.k.)

Algorithm: Go through the descriptor directory and provide the desired info.
For 'fno' = 1 the name of the descr. is stored into 'buf';

for 'fno' = 2 the type of the descr. is stored as a char. string in 'buf';
for 'fno' = 3 the no. of elements of the descriptor is stored in 'numbuf'.

Function *SCDGETC*

SCDGETC(imno,descr,felem,maxvals,actvals,values)

int imno	*IN: file id of data frame*
char *descr	*IN: descriptor name (max. 15 chars.)*
int felem	*IN: position of 1st element to be accessed*
int maxvals	*IN: max. no. of characters to be returned*
int *actvals	*OUT: actual no. of characters returned*
char *values	*OUT: array for descriptor data*

Purpose: Get data from character descriptor and terminate with '\0' (no trailing blanks).

Returns: return status (0 = o.k.)

Algorithm: Use SCDRDC.

Function *SCDRDC*

SCDRDC(imno,descr,noelm,felem,maxvals,actvals,values,unit,null)

int imno	*IN: file id of data frame*
char *descr	*IN: descriptor name (max. 15 chars.)*
int noelm	*IN: no. of bytes per data value*
int felem	*IN: position of 1st element to be accessed*
int maxvals	*IN: max. no. of values to be returned*
int *actvals	*OUT: actual no. of values returned*
char *values	*OUT: array for descriptor data*
int *unit	*OUT: address of unit pointer*
int *null	*OUT: no. of null values in data*

Purpose: Read the contents of a character descriptor.

Returns: return status (0 = o.k.)

Algorithm: 'actvals' will be set to the actual no. of values returned.

Remarks: 'noelm' larger than 1 enables you to use character arrays, where each data value is in effect a char. string of 'noelm' bytes.

Function *SCDRDD*

SCDRDD(imno,descr,felem,maxvals,actvals,values,unit,null)

int imno	*IN: file id of data frame*
char ∗descr	*IN: descriptor name (max. 15 chars.)*
int felem	*IN: position of 1st element to be accessed*
int maxvals	*IN: max. no. of values to be returned*
int ∗actvals	*OUT: actual no. of values returned*
double ∗values	*OUT: array for descriptor data*
int ∗unit	*OUT: address of unit pointer*
int ∗null	*OUT: no. of null values in data*

Purpose: Read the contents of a double precision descriptor.

Returns: return status (0 = o.k.)

Algorithm: 'actvals' will be set to the actual no. of values returned.

Function *SCDRDH*

SCDRDH(imno,descr,felem,maxvals,actvals,values,totvals)

int imno	*IN: file id of data frame*
char ∗descr	*IN: descriptor name (max. 15 chars.)*
int felem	*IN: position of 1st char. in help text to be accessed*
int maxvals	*IN: max. no. of chars. to be returned*
int ∗actvals	*OUT: actual no. of chars. returned*
char ∗values	*OUT: buffer for help text*
int ∗totvals	*OUT: total size of help text*

Purpose: Read the help text of a descriptor.

Returns: return status (0 = o.k.)

Algorithm: 'actvals' will be set to the actual no. of characters of the help text returned in 'values'. If no help text exists for the descriptor, 'totvals' is set to -1.

Function *SCDRDI*

SCDRDI(imno,descr,felem,maxvals,actvals,values,unit,null)

int imno	*IN: file id of data frame*
char ∗descr	*IN: descriptor name (max. 15 chars.)*
int felem	*IN: position of 1st element to be accessed*
int maxvals	*IN: max. no. of values to be returned*
int ∗actvals	*OUT: actual no. of values returned*
int ∗values	*OUT: array for descriptor data*
int ∗unit	*OUT: address of unit pointer*
int ∗null	*OUT: no. of null values in data*

Purpose: Read the contents of an integer descriptor.

Returns: return status (0 = o.k.)

Algorithm: 'actvals' will be set to the actual no. of values returned.

Function *SCDRDR*

SCDRDR(imno,descr,felem,maxvals,actvals,values,unit,null)

int imno	*IN: file id of data frame*
char *descr	*IN: descriptor name (max. 15 chars.)*
int felem	*IN: position of 1st element to be accessed*
int maxvals	*IN: max. no. of values to be returned*
int *actvals	*OUT: actual no. of values returned*
float *values	*OUT: array for descriptor data*
int *unit	*OUT: address of unit pointer*
int *null	*OUT: no. of null values in data*

Purpose: Read the contents of a real descriptor.

Returns: return status (0 = o.k.)

Algorithm: 'actvals' will be set to the actual no. of values returned.

Function *SCDWRC*

SCDWRC(imno,descr,noelm,values,felem,nval,unit)

int imno	*IN: file id of data frame*
char *descr	*IN: descriptor name (max. 15 char.)*
int noelm	*IN: no. of bytes per data value*
char *values	*IN: buffer with descriptor data*
int felem	*IN: position of 1st descriptor value to be accessed*
int nval	*IN: no. of data values to write*
int *unit	*IN: unit-pointer*

Purpose: Write a descriptor and its data.

Returns: return status (0 = o.k.)

Algorithm: A new descriptor is added to the descriptor area or data of an existing descriptor is modified and extended, if necessary.

Remarks: If 'felem' = -1, descriptor values are appended to existing ones.
'noelm' larger than 1 enables you to use character arrays, where each data value is in effect a char. string of 'noelm' bytes.

Function *SCDWRD*

SCDWRD(imno,descr,values,felem,nval,unit)

int imno	*IN: file id of data frame*
char *descr	*IN: descriptor name (max. 15 char.)*
double *values	*IN: buffer with descriptor data*
int felem	*IN: position of 1st descriptor value to be accessed*
int nval	*IN: no. of data values to write*
int *unit	*IN: unit-pointer*

Purpose: Write a descriptor and its data.

Returns: return status (0 = o.k.)

Algorithm: A new descriptor is added to the descriptor area or data of an existing descriptor is modified and extended, if necessary.

Remarks: If 'felem' = -1, descriptor values are appended to existing ones.

Function *SCDWRH*

SCDWRH(imno,descr,values,felem,nval)

int imno	*IN: file id of data frame*
char *descr	*IN: descriptor name (max. 15 char.)*
char *values	*IN: buffer with help text*
int felem	*IN: position of 1st char. in help text to be accessed*
int nval	*IN: no. of characters to write*

Purpose: Write the help text of a descriptor.

Returns: return status (0 = o.k.)

Algorithm: A help text is added for the given descriptor or the help text of a descriptor is modified and extended, if necessary.

Remarks: If 'felem' = -1, the buffer 'values' is appended to the help text.

Function *SCDWRI*

SCDWRI(imno,descr,values,felem,nval,unit)

int imno	*IN: file id of data frame*
char *descr	*IN: descriptor name (max. 15 char.)*
int *values	*IN: buffer with descriptor data*
int felem	*IN: position of 1st descriptor value to be accessed*
int nval	*IN: no. of data values to write*
int *unit	*IN: unit-pointer*

Purpose: Write a descriptor and its data.

Returns: return status (0 = o.k.)

Algorithm: A new descriptor is added to the descriptor area or data of an existing descriptor is modified and extended, if necessary.

Remarks: If 'felem' = -1, descriptor values are appended to existing ones.

Function *SCDWRR*

SCDWRR(imno,descr,values,felem,nval,unit)

int imno	*IN: file id of data frame*
char ∗descr	*IN: descriptor name (max. 15 char.)*
float ∗values	*IN: buffer with descriptor data*
int felem	*IN: position of 1st descriptor value to be accessed*
int nval	*IN: no. of data values to write*
int ∗unit	*IN: unit-pointer*

Purpose: Write a descriptor and its data.

Returns: return status (0 = o.k.)

Algorithm: A new descriptor is added to the descriptor area or data of an existing descriptor is modified and extended, if necessary.

Remarks: If 'felem' = -1, descriptor values are appended to existing ones.

Module *Version [3.10] (17-Jan-1994)* **Module SCE**

Author: K. Banse, ESO - Garching *Language:* C

Classification: standard interfaces, errors

Environment:
- VMS and UNIX
- #include <midas_def.h>

Comments:

This module contains the routines to handle errors in MIDAS.
The functions provided are:

- Setup program behaviour in case of errors/warnings (SCECNT)
- Terminate program in case of errors and signal errors (SCETER, SCESIG)

Function *SCECNT*

SCECNT(action,cont,log,disp)

char *action	*IN: GET or PUT*
int *cont	*IN: pointer to continuation flag*
int *log	*IN: pointer to log flag*
int *disp	*IN: pointer to display flag*

Purpose: Modify flow of control after an error/warning occurred

Returns: status always = 0

Algorithm: straight forward

Remarks: Continue_flag
= -1 (stop on warnings and errors),
= 0 (stop on errors only),
= 1 (continue always).
Log_flag
= 0 (log nothing),
= 1 (log only errors),
= 2 (log warnings and errors).
Display_flag
= 0 (display nothing on the terminal),
= 1 (display errors),
= 2 (display warnings and errors).
In Midas the default is 0, 2 and 1 for the Continue, Log and Display_flag.

Function *SCETER*

SCETER(errno,text)

int errno	*IN: error no.*
char *text	*IN: message to display*

Purpose: Abort program and also the MIDAS procedure it was embedded in

Returns: nothing

Algorithm: display text, put error no. 'errno' into key PROGSTAT and stop execution

---

Function *SCESIG*

SCESIG(sys,text,stat)

char ∗sys	*IN: system from which error emanated*
char ∗text	*IN: message to display*
int stat	*IN: error no.*

Purpose: Signal an error to the calling routine

Returns: nothing

Algorithm: straight forward

Module	*Version [3.10] (17-Jan-1994)*	**Module SCF**

Author: K. Banse, ESO - Garching *Language:* C

Classification: standard interfaces, bulk data files

Environment:
- VMS and UNIX
- #include <midas_def.h>

Comments:

This module contains the routines to manipulate MIDAS data frames.
The functions provided are:

- Create, open and close data files (SCFCRE, SCFOPN, SCFCLO)
- Get info about data files (SCFINF)
- Rename and delete data files (SCFDEL, SCFRNM)
- Input and output of frame data (SCFMAP, SCFUNM, SCFGET, SCFPUT)

Main arguments used by the routines are:

name name of MIDAS data frame

imno file id of related MIDAS frame

dattype format how the data is stored on disk

filtype type of MIDAS data frame

iomode access mode of the frame

felem first element to be accessed, the very first element is indicated by 1 (not by 0 as in C-arrays) for compatibility with the relevant FORTRAN interfaces.

Function *SCFOPN*

SCFOPN(name,dattype,newopn,filtype,imno)

char *name	*IN: name of data frame*
int dattype	*IN: data types as defined below*
int newopn	*IN: new_open_flag: = 0 open normally, = 1 open same file again*
int filtype	*IN: filetype as defined below*
int *imno	*OUT: file id of the frame*

Purpose: Open an existing frame and return its internal file no.

Returns: return status (0 = o.k.)

Algorithm: Use low level interfaces to do the job

Remarks: Supported data types are:
D_R4_FORMAT, D_R8_FORMAT, D_I4_FORMAT, D_I2_FORMAT,
D_UI2_FORMAT, D_I1_FORMAT and D_OLD_FORMAT.
Filetypes are:
F_IMA_TYPE, F_TBL_TYPE, F_FIT_TYPE and F_OLD_TYPE.

Function *SCFINF*

SCFINF(name,fno,ibuf)

char *name	*IN: name of data frame*
int fno	*IN: flag for requested info*
int *ibuf	*OUT: buffer with requested info*

Purpose: open the frame (except for 'fno'=0) and return the file info in buffer of up to 5 integers

Returns: return status (0 = o.k., else the required info cannot be obtained)

Algorithm: Use low level interfaces to do the job

Remarks: 'fno' = 0 for getting the file id (imno) of 'name', i.e. check if frame 'name' has been opened already
'fno' = 1 for getting version_no., file_type, short int, int and floating format
'fno' = 2 for getting no_bytes_per_pixel, format, pixels_per_block, startblock and lastblock
'fno' = 3 for getting file_type, format, file_protect, file_compress
'fno' = 4 for getting no_bytes_per_pixel, format, file_type
'fno' = 99 to find out if file 'name' exists,
no buffer is returned, instead the status is set to 0 (o.k. = file exists) or not

---

Function *SCFCLO*

SCFCLO(imno)

int imno	*IN: file id of data frame*

Purpose: Close a frame.

Returns: return status (0 = o.k.)

Algorithm: Add frame entry to image catalog (if enabled) and close file on disk.

Function *SCFCRE*

SCFCRE(name,dattype,iomode,filtype,size,imno)

char *name	*IN: name of frame*
int dattype	*IN: data type, e.g. D_R4_FORMAT, ...*
int iomode	*IN: opening mode, e.g. F_O_MODE, ...*
int filtype	*IN: filetype no., e.g. F_TBL_TYPE, ...*
int size	*IN: size of frame (= no. of data values in file)*
int *imno	*OUT: file id*

Purpose: Create a frame with data of given type and return its internal file no.

Returns: return status (0 = o.k.)

Algorithm: Fill all internal data structures and create file on disk

Remarks: Opening modes are: F_O_MODE or F_X_MODE,
see SCFOPN for details of different data and file types.

Function *SCFDEL*

SCFDEL(name)

char ∗name	*IN: name of frame*

Purpose: Delete a frame

Returns: return status (0 = o.k.)

Algorithm: Use low level interfaces to do the job

Function *SCFMAP*

SCFMAP(imno,iomode,felem,size,actsize,pntr)

int imno	*IN: file id of data frame*
int iomode	*IN: opening mode, e.g. F_O_MODE, ...*
int felem	*IN: 1st pixel to be accessed in data space*
int size	*IN: number of data values (pixels) to be mapped*
int ∗actsize	*OUT: actual no. of pixels mapped*
char ∗∗pntr	*OUT: pointer to data in memory*

Purpose: Map a new data frame into memory for writing/reading
or just get virtual memory (iomode = F_X_MODE)

Returns: return status (0 = o.k.)

Algorithm: return pointer to first element which is required

Remarks: Modes are: F_I_MODE (read), F_IO_MODE (read_write),
F_O_MODE (write) and F_X_MODE (allocate virtual memory)

Function *SCFUNM*

SCFUNM(imno)

int imno	*IN: file id of data frame*

Purpose: Release mapped memory of a data frame

Returns: return status (0 = o.k.)

Algorithm: Use low level functions

Function *SCFGET*

SCFGET(imno,felem,size,actsize,bufadr)

int imno	*IN: file id of data frame*
int felem	*IN: 1st pixel to be accessed in data space*
int size	*IN: number of data values (pixels) to be read*
int ∗actsize	*OUT: actual no. of pixels read*
char ∗bufadr	*OUT: data buffer*

Purpose: Read data from disk frame into memory

Returns: return status (0 = o.k.)

Algorithm: synchronous disk I/O, data are converted on the fly if necessary

Remarks: SCFGET cannot be used for frames created with F_X_MODE.

Function *SCFPUT*

SCFPUT(imno,felem,size,bufadr)

int imno	*IN: file id of data frame*
int felem	*IN: 1st pixel to be accessed in data space*
int size	*IN: number of data values (pixels) to be written*
char *bufadr	*IN: data buffer*

Purpose: write data from memory into disk frame

Returns: return status (0 = o.k.)

Algorithm: synchronous disk I/O, data are converted on the fly if necessary

Remarks: SCFPUT cannot be used for frames created with F_X_MODE.

Function *SCFRNM*

SCFRNM(oldname,newname)

char *oldname	*IN: old name*
char *newname	*OUT: new name*

Purpose: rename a disk file

Returns: return status (0 = o.k.)

Algorithm: use low level interfaces

Remarks: Complete file names, i.e. including file type (extension), have to be used.

Module	*Version [3.10] (17-Jan-1994)*	**Module SCI**

Author: K. Banse, ESO - Garching *Language:* C

Classification: standard interfaces, data frames, descriptors

Environment:
- VMS and UNIX
- #include <midas_def.h>

Comments:

This module contains the (high level) routines to handle MIDAS images.
The functions provided are:

- Get image data and descriptors from disk (SCIGET)
- Create image frame and fill its standard descriptors on disk (SCIPUT). This routine does not fill in any data values!

Main arguments used by the routines are:

name name of MIDAS data frame

imno file id of MIDAS frame

dattype format how the data is stored on disk

filtype type of MIDAS data frame

iomode access mode of the frame

pntr pointer to mapped data

Function *SCIGET*

SCIGET(name,dattype,iomode,filtype,maxdim,naxis,npix,start,step,ident,cunit,pntr,imno)

char *name	*IN: data frame name*
int dattype	*IN: data type*
int iomode	*IN: opening mode of frame, F_I_MODE or F_IO_MODE*
int filtype	*IN: filetype no.*
int maxdim	*IN: maximum number of dimensions*
int *naxis	*OUT: no of dimensions of the image*
int *npix	*OUT: size of each dimension*
double *start	*OUT: start coordinates: start is an array of naxis elements*
double *step	*OUT: stepsizes : step is an array of naxis elements*
char *ident	*OUT: ascii identifier of image*
char *cunit	*OUT: unit of each axis (16 char per unit)*
char **pntr	*OUT: pointer to mapped data*
int *imno	*OUT: file id of data frame*

Purpose: Open image frame.
Map all data for reading or updating and read the standard descriptors.

Returns: return status (0 = o.k.)

Algorithm: Use SCDRDx and SCFMAP to map and to read the standard descriptors.

Remarks: For a detailed description of the parameters , see text for SCF... and SCD... routines

---

Function *SCIPUT*

SCIPUT(name,dattype,iomode,filtype,naxis,npix,start,step,ident,cunit,pntr,imno)

char *name	*IN: data frame name*
int dattype	*IN: data type*
int iomode	*IN: opening mode of frame, F_O_MODE or F_X_MODE*
int filtype	*IN: file type no.*
int naxis	*IN: no of dimensions of the image*
int *npix	*IN: size of each dimension*
double *start	*IN: start coordinates: start is an array of naxis elements*
double *step	*IN: stepsizes : step is an array of naxis elements*
char *ident	*IN: ascii identifier of image*
char *cunit	*IN: unit of each axis (16 char per unit)*
char **pntr	*OUT: pointer to mapped data*
int *imno	*OUT: file id of data frame*

Purpose: Create and map complete bulk data frame and write the standard descriptors

Returns: return status (0 = o.k.)

Algorithm: call SCFCRE, SCFMAP and SCDWRx routines

Remarks: This call does not write any data values into the new frame, it just allocates (maps) the space for it!
For a detailed description of the parameters , see text for SCF... and SCD... routines

---

Module *Version [3.10] (17-Jan-1994)* **Module SCK**

Author: K. Banse, ESO - Garching *Language:* C

Classification: standard interfaces, keyword data base

Environment:
- VMS and UNIX
- #include <midas_def.h>

Comments:

This module contains the routines to interact with the keywords of MIDAS. The functions provided are:

- Find keywords and get information about them (SCKFND, SCKINF)
- Prompt for keywords (SCKPRC, SCKPRD, SCKPRI, SCKPRR)
- Read contents of keywords (SCKRDC, SCKRDD, SCKRDI, SCKRDR)
- Write contents of keywords (SCKWRC, SCKWRD, SCKWRI, SCKWRR)
- Read contents of char. keyword in a simplified way (SCKGETC)

Main arguments used by the routines are:

key name of keyword, max. 8 chars. long and must begin with an alphabetic char.

felem first element to be accessed, the very first element is indicated by 1 (not by 0 as in C-arrays) for compatibility with the relevant FORTRAN interfaces.

maxvals maximum number of elements required.

actvals actual number of elements accessed, value returned by the routine.

unit unit of keyword, currently not implemented - just used as a placeholder for now.

null no. of null values in keyword, currently not implemented - just used as a placeholder for now.

Function *SCKFND*

SCKFND(key,type,noelem,bytelem)

char *key	*IN : keyword name (max. 8 chars.)*
char *type	*OUT: type of key*
int *noelem	*OUT: no. of elements*
int *bytelem	*OUT: no. of bytes per element*

Purpose: Find keyword named ‘key’ and provide info about it.

Returns: return status (0 = ok)

Algorithm: Search keyword data base to find key + related info.

Remarks: type of keyword
= I for integer, = R for real, = D for double, = C for character keywords;
if key not found, type = ’ ’ (a blank) is returned
bytelem = fixed (4,4,8) for integer, real, double keywords
for character keywords, bytelem >= 1

Function *SCKINF*

SCKINF(npos,fno,buf,lbuf,numbuf)

int npos	*IN: position of keyword*
int fno	*IN: specifies desired info, 1 = NAME, 2 = TYPE, 3 = SIZE*
char *buf	*OUT: buffer for NAME + TYPE info*
int lbuf	*IN: max. length of buffer above*
int *numbuf	*OUT: buffer for SIZE info*

Purpose: Provide info about keyword at position 'npos'.

Returns: return status (0 = ok)

Algorithm: Go through the keywords data base and provide the desired info.

Remarks: for fno = 1,2 the character buffer 'buf' is filled,
for fno = 3 the numerical output buffer 'numbuf' is filled.

Function *SCKGETC*

SCKGETC(key,felem,maxvals,actvals,values)

char *key	*IN: keyword name (max. 8 chars.)*
int felem	*IN: position of 1st element to be accessed*
int maxvals	*IN: max. no. of characters to be returned*
int *actvals	*OUT: actual no. of characters returned*
char *values	*OUT: buffer for data values*

Purpose: Get data from character keyword and terminate with '\0' (no trailing blanks).

Returns: return status (0 = o.k.)

Algorithm: Use SCKRDC.

Algorithm: This interface only works for char. keywords with 1 byte (char.) per element.

Function *SCKPRC*

SCKPRC(prompt,key,noelm,felem,maxvals,actvals,values,unit,null)

char *prompt	*IN: prompt string (null terminated)*
char *key	*IN: keyword name (max. 8 chars.)*
int noelm	*IN: no. of bytes per data value*
int felem	*IN: 1st data item to be read*
int maxvals	*IN: no. of elements to get*
int *actvals	*OUT: actual no. of elements returned*
char *values	*OUT: buffer for data values*
int *unit	*OUT: address of unit-pointer*
int *null	*OUT: no. of null values in keyword*

Purpose: Display prompt string and read character keyword data from terminal

Returns: return status (0 = ok)

Algorithm: 'actvals' will be set to the actual no. of values returned.

Remarks: 'noelm' > 1 enables you to use character arrays, where each data value is in effect a char. string of 'noelm' bytes.

Function *SCKPRD*

SCKPRD(prompt,key,felem,maxvals,actvals,values,unit,null)

char *prompt	*IN: prompt string (null terminated)*
char *key	*IN: keyword name (max. 8 chars.)*
int felem	*IN: 1st data item to be read*
int maxvals	*IN: no. of elements to get*
int *actvals	*OUT: actual no. of elements returned*
double *values	*OUT: buffer for data values*
int *unit	*OUT: address of unit-pointer*
int *null	*OUT: no. of null values in keyword*

Purpose: Display prompt string and read double precision keyword data from terminal.

Returns: return status (0 = ok)

Algorithm: 'actvals' will be set to the actual no. of values returned.

Function *SCKPRI*

SCKPRI(prompt,key,felem,maxvals,actvals,values,unit,null)

char *prompt	*IN: prompt string (null terminated)*
char *key	*IN: keyword name (max. 8 chars.)*
int felem	*IN: 1st data item to be read*
int maxvals	*IN: no. of elements to get*
int *actvals	*OUT: actual no. of elements returned*
int *values	*OUT: buffer for data values*
int *unit	*OUT: address of unit-pointer*
int *null	*OUT: no. of null values in keyword*

Purpose: Display prompt string and read integer keyword data from terminal.

Returns: return status (0 = ok)

Algorithm: 'actvals' will be set to the actual no. of values returned.

Function *SCKPRR*

SCKPRR(prompt,key,felem,maxvals,actvals,values,unit,null)

char ∗prompt	*IN: prompt string (null terminated)*
char ∗key	*IN: keyword name (max. 8 chars.)*
int felem	*IN: 1st data item to be read*
int maxvals	*IN: no. of elements to get*
int ∗actvals	*OUT: actual no. of elements returned*
float ∗values	*OUT: buffer for data values*
int ∗unit	*OUT: address of unit-pointer*
int ∗null	*OUT: no. of null values in keyword*

Purpose: Display prompt string and read real keyword data from terminal.

Returns: return status (0 = ok)

Algorithm: 'actvals' will be set to the actual no. of values returned.

Function *SCKRDC*

SCKRDC(key,noelm,felem,maxvals,actvals,values,unit,null)

char ∗key	*IN: keyword name (max. 8 chars.)*
int noelm	*IN: no. of bytes per data value*
int felem	*IN: first data item to be read*
int maxvals	*IN: no. of elements to get*
int ∗actvals	*OUT: actual no. of elements returned*
char ∗values	*OUT: buffer for data values*
int ∗unit	*OUT: address of unit-pointer*
int ∗null	*OUT: no. of null values in keyword*

Purpose: Read data from character keyword.

Returns: return status (0 = ok)

Algorithm: 'actvals' will be set to the actual no. of values returned.

Remarks: 'noelm' > 1 enables you to use character arrays, where each data value is in effect a char. string of 'noelm' bytes

---

Function *SCKRDD*

SCKRDD(key,felem,maxvals,actvals,values,unit,null)

char ∗key	*IN: keyword name (max. 8 chars.)*
int felem	*IN: first data item to be read*
int maxvals	*IN: no. of elements to get*
int ∗actvals	*OUT: actual no. of elements returned*
double ∗values	*OUT: buffer for data values*
int ∗unit	*OUT: address of unit-pointer*
int ∗null	*OUT: no. of null values in keyword*

Purpose: Read data from double precision keyword.

Returns: return status (0 = ok)

Algorithm: 'actvals' will be set to the actual no. of values returned.

Function *SCKRDI*

SCKRDI(key,felem,maxvals,actvals,values,unit,null)

char *key	*IN: keyword name (max. 8 chars.)*
int felem	*IN: first data item to be read*
int maxvals	*IN: no. of elements to get*
int *actvals	*OUT: actual no. of elements returned*
int *values	*OUT: buffer for data values*
int *unit	*OUT: address of unit-pointer*
int *null	*OUT: no. of null values in keyword*

Purpose: Read data from integer keyword.

Returns: return status (0 = ok)

Algorithm: 'actvals' will be set to the actual no. of values returned.

Function *SCKRDR*

SCKRDR(key,felem,maxvals,actvals,values,unit,null)

char *key	*IN: keyword name (max. 8 chars.)*
int felem	*IN: first data item to be read*
int maxvals	*IN: no. of elements to get*
int *actvals	*OUT: actual no. of elements returned*
float *values	*OUT: buffer for data values*
int *unit	*OUT: address of unit-pointer*
int *null	*OUT: no. of null values in keyword*

Purpose: Read data from real keyword.

Returns: return status (0 = ok)

Algorithm: 'actvals' will be set to the actual no. of values returned.

Function *SCKWRC*

SCKWRC(key,noelm,values,felem,maxvals,unit)

char *key	*IN: keyword name (max. 8 chars.)*
int noelm	*IN: no. of bytes per data value*
char *values	*IN: buffer with data values*
int felem	*IN: first data item to be written*
int maxvals	*IN: no. of elements to write*
int *unit	*IN: unit-pointer*

Purpose: Write data into character keyword

Returns: return status (0 = ok)

Remarks: ‘noelm’ > 1 enables you to use character arrays, where each data value is in effect a char. string of ‘noelm’ bytes

---

Function *SCKWRD*

SCKWRD(key,values,felem,maxvals,unit)

char *key	*IN: keyword name (max. 8 chars.)*
double *values	*IN: buffer with data values*
int felem	*IN: first data item to be written*
int maxvals	*IN: no. of elements to write*
int *unit	*IN: unit-pointer*

Purpose: Write data into double precision keyword.

Returns: return status (0 = ok)

Function *SCKWRI*

SCKWRI(key,values,felem,maxvals,unit)

char *key	*IN: keyword name (max. 8 chars.)*
int *values	*IN: buffer with data values*
int felem	*IN: first data item to be written*
int maxvals	*IN: no. of elements to write*
int *unit	*IN: unit-pointer*

Purpose: Write data into integer keyword.

Returns: return status (0 = ok)

Function *SCKWRR*

SCKWRR(key,values,felem,maxvals,unit)

char *key	*IN: name of keyword*
float *values	*IN: buffer with data values*
int felem	*IN: first data item to be written*
int maxvals	*IN: no. of elements to write*
int *unit	*IN: unit-pointer*

Purpose: Write data into real keyword.

Returns: return status (0 = ok)

Module *Version [3.10] (17-Jan-1994)* **Module SCST**

Author: K. Banse, ESO - Garching *Language:* C

Classification: standard interfaces.

Environment:
- VMS and UNIX
- #include <midas_def.h>

Comments:

This module contains the routines to link into the MIDAS environment and to display text.
The functions provided are:

- Link to and unlink from the MIDAS environment (SCSPRO, SCSEPI)
- Display (and log) text (SCTPUT, SCTDIS)

Function *SCSPRO*

SCSPRO(prog)

char *prog *IN: name of calling program*

Purpose: Get into the MIDAS environment.

Returns: return status (0 = o.k)

Algorithm: Read keywords data base, initialize File Control Table and open logfile

Remarks: Set the program name to the string "-1" to run the program outside a MIDAS session (this implies that there will be no keywords available...)

Function *SCSEPI*

SCSEPI()

Purpose: Write keywords back and clean up everything

Returns: return status (0 = o.k)

Function *SCTPUT*

SCTPUT(text)

char *text *IN: text string to be displayed*

Purpose: Display character string on the terminal and store it also in the Midas logfile.

Returns: return status (0 = o.k)

Function *SCTDIS*

SCTDIS(text,bc)

char *text *IN: text string to be displayed*
int bc *IN: backspace count*

Purpose: Display character string on the terminal; do not (!) store it in the Midas logfile.

Returns: return status (0 = o.k)

Algorithm: backspace count = 0, write line normally (with carriage-return and new-line)
= 1, ..., 80, do that many backspaces, so the next message may overwrite

Module *Version [3.00] (15-Jan-1992)* **Module STC**

Author: K. Banse ESO - Garching *Language:* F77

Classification: MIDAS catalogues

Environment:
- VMS and UNIX
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DEF.INC
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DAT.INC

Comments:

This module contains the routines to interact with the catalogs of MIDAS. The functions provided are:

- Create catalogs (STCCRE)
- Add and remove entries in catalogs (STCADD, STCSUB)
- Find entries and get sequential entries in a catalog (STCFND, STCGET)
- Display contents of catalogs (STCLIS, STCSHO)

Main arguments used by the routines are:

CATFIL name of catalog

Subroutine *STCCRE*

STCCRE(CATFIL,TYPE,FLAG,STAT)

CHARACTER*(*) CATFIL	*IN: name of catalog file*
INTEGER TYPE	*IN: type of files to use*
INTEGER FLAG	*IN: = 1, if 'dirfile.dat' exists*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Create a catalog

Algorithm: If 'FLAG' = 1, read each filename from directory output file 'dirfile.dat' and read relevant descriptors from each file and store in catalog file, else create an empty catalog.

Subroutine *STCADD*

STCADD(CATFIL,NAME,IDENT,STAT)

CHARACTER*(*) CATFIL	*IN: name of catalog file*
CHARACTER*(*) NAME	*IN: name of frame to be added*
CHARACTER*(*) IDENT	*IN: IDENT info of frame to be added*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Add entry in a catalog

Algorithm: Add an entry with given name and identification field to the catalog.

Subroutine *STCSUB*

STCSUB(CATFIL,NAME,STAT)

CHARACTER*(*) CATFIL	*IN: name of catalog file*
CHARACTER*(*) NAME	*IN: name of frame to be removed*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Subtract entry in a catalog

Algorithm: Search for given entry and change first 2 chars to '//', which makes this entry unused.

Subroutine *STCGET*

STCGET(CATFIL,FLAG,NAME,IDENT,NO,STAT)

CHARACTER*(*) CATFIL	*IN: name of catalog file*
INTEGER FLAG	*IN: 0 for name only, = 1 for name and identifier*
CHARACTER*(*) NAME	*OUT: name of frame*
CHARACTER*(*) IDENT	*OUT: identifier of that frame*
INTEGER NO	*IN/OUT: current/next entry no. as input/output*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Get next sequential entry from a catalog

Algorithm: Return name and identifier of next entry in catalog file;
return a blank in NAME(1:1), if at EOF.
Parameter 'NO' has to be set to 0 in the first call of the routine.

Subroutine *STCLIS*

STCLIS(CATFIL,INTVAL,STAT)

CHARACTER*(*) CATFIL	*IN: name of catalog file*
INTEGER INTVAL(2)	*IN: listing interval [low,hi]*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: List entries of a catalog

Algorithm: Read and display catalog, entry by entry.

Subroutine *STCFND*

STCFND(CATFIL,FRMNO,FRAME,STAT)

CHARACTER*(*) CATFIL	*IN: name of catalog file*
INTEGER FRMNO	*IN: entry no. in catalog*
CHARACTER*(*) FRAME	*OUT: corresponding frame name*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Find entry in a catalog and return corresponding frame name

Algorithm: Search for given entry no. and return connected frame name.

Subroutine *STCSHO*

STCSHO(CATFIL,NOENT,LAST,STAT)

CHARACTER*(*) CATFIL	*IN: name of catalog file*
INTEGER NOENT	*OUT: no. of entries*
INTEGER LAST	*OUT: last entry no.*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Show size and type of a catalog

Algorithm: Read and count active entries of catalog.

Module *Version [3.00] (15-Jan-1992)* **Module STD**

Author: K. Banse ESO - Garching *Language:* F77

Classification: standard interfaces, descriptors

Environment:
- VMS and UNIX
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DEF.INC
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DAT.INC

Comments:

This module contains the routines to interact with the descriptors of MIDAS frames.
The functions provided are:

- Find descriptors and get information about them (STDFND, STDINF)
- Copy and delete descriptors (STDCOP, STDDEL)
- Read contents of descriptors (STDRDC, STDRDD, STDRDI, STDRDR)
- Write contents of descriptors (STDWRC, STDWRD, STDWRI, STDWRR)

Main arguments used by the routines are:

NO file id of related MIDAS frame

DSC name of descriptor, max. 15 chars. long and must begin with an alphabetic char.

FELM first element to be accessed, the very first element is indicated by 1 FORTRAN interfaces.

MAXS maximum number of elements required.

ACTS actual number of elements accessed, value returned by the routine.

DUN unit of descriptor, currently not implemented - just used as a placeholder for now.

DNUL no. of null values in descriptor, currently not implemented - just used as a placeholder for now.

STAT return status.

Subroutine *STDCOP*

STDCOP(FROM,TO,MASK,DSC,STAT)

INTEGER FROM	*IN: file id of source frame*
INTEGER TO	*IN: file id of destination frame*
INTEGER MASK	*IN: copy_mask*
CHARACTER*(*) DSC	*IN: descriptor name (max 15 chars.)*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: Copy descriptors from one file to another one.

Algorithm: Depending on 'MASK' the relevant descriptors are read from the source frame and written to the destination frame.

Remarks: Mask for copying:
1 - copy all descriptors,
2 - copy only standard descriptors (cf. Appendix E,
3 - copy all but standard descriptors,
4 - copy single descriptor specified by par. 'DSC',
5 - copy all but extended list of standard descriptors

Subroutine *STDDEL*

STDDEL(NO,DSC,STAT)

INTEGER NO	*IN: file id of frame*
CHARACTER*(*) DSC	*IN: descriptor name (max 15 chars.)*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: Delete descriptors

Algorithm: The descriptor directory is scanned for the specified name and if found, that name is marked as deleted.

Subroutine *STDFND*

STDFND(NO,DSC,TYPE,NOELM,BYTELM,STAT)

INTEGER NO	*IN: file id of frame*
CHARACTER*(*) DSC	*IN: descriptor name (max 15 chars.)*
CHARACTER*(*) TYPE	*OUT: descriptor type - I, R, C, D or ' ' (a blank)*
INTEGER NOELM	*OUT: no. of elements*
INTEGER BYTELM	*OUT: no. of bytes per element*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: Get descriptor information

Algorithm: Search descriptor directory, if descr. 'DSC' is not found, TYPE(1:1) is set to a blank.

Subroutine *STDINF*

STDINF(NO,NPOS,FNO,BUF,NUMBUF,STAT)

INTEGER NO	*IN: file id of frame*
INTEGER NPOS	*IN: position of descr.*
INTEGER FNO	*IN: specify desired info*
CHARACTER*(*) BUF	*OUT: return buffer (FNO=1,2)*
INTEGER NUMBUF	*OUT: return value (FNO=3)*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: Find descriptor at position #n and return info.

Algorithm: Go through the descriptor directory and provide the desired info.
For 'FNO' = 1 the name of the descr. is stored into 'BUF';
for 'FNO' = 2 the type of the descr. is stored as a char. string in 'BUF';
for 'FNO' = 3 the no. of elements of the descriptor is stored in 'NUMBUF'.

Subroutine *STDRDC*

STDRDC (NO,DSC,NOELM,FELM,MAXS,ACTS,VALS,DUN,DNUL,STAT)

INTEGER NO	*IN: file id of frame*
CHARACTER*(*) DSC	*IN: descriptor name (max 15 chars.)*
INTEGER NOELM	*IN: no. of bytes per data value*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: max. no. of values to read*
INTEGER ACTS	*OUT: actual no. of values read*
CHARACTER*(*) VALS	*OUT: buffer for data*
INTEGER DUN	*OUT: units*
INTEGER DNUL	*OUT: no. of null values*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: Read character descriptor

Algorithm: 'ACTS' will be set to the actual no. of values returned.

Remarks: 'NOELM' larger than 1 enables you to use character arrays, where each data value is in effect a char. string of 'NOELM' bytes.

Subroutine *STDRDD*

STDRDD(NO,DSC,FELM,MAXS,ACTS,VALS,DUN,DNUL,STAT)

INTEGER NO	*IN: file id of frame*
CHARACTER*(*) DSC	*IN: descriptor name (max 15 chars.)*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: max. no. of values to read*
INTEGER ACTS	*OUT: actual no. of values read*
DOUBLE PRECISION VALS(1)	*OUT: buffer for data*
INTEGER DUN	*OUT: units*
INTEGER DNUL	*OUT: no. of null values*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: Read double precision descriptor

Algorithm: 'ACTS' will be set to the actual no. of values returned.

Subroutine *STDRDH*

STDRDH(NO,DSC,FELM,MAXS,ACTS,VALS,TOTVLS,STAT)

INTEGER NO	*IN: file id of frame*
CHARACTER*(*) DSC	*IN: descriptor name (max 15 chars.)*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: max. no. of values to read*
INTEGER ACTS	*OUT: actual no. of values read*
CHARACTER*(*) VALS	*OUT: buffer for help text*
INTEGER TOTVLS	*OUT: total size of help text*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: Read the help text of a descriptor

Algorithm: 'ACTS' will be set to the actual no. of characters of the help text returned in 'VALS'. If no help text exists for the descriptor, 'TOTVLS' is set to -1.

Subroutine *STDRDI*

STDRDI(NO,DSC,FELM,MAXS,ACTS,VALS,DUN,DNUL,STAT)

INTEGER NO	*IN: file id of frame*
CHARACTER*(*) DSC	*IN: descriptor name (max 15 chars.)*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: max. no. of values to read*
INTEGER ACTS	*OUT: actual no. of values read*
INTEGER VALS(1)	*OUT: buffer for data*
INTEGER DUN	*OUT: units*
INTEGER DNUL	*OUT: no. of null values*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: Read integer descriptor

Algorithm: 'ACTS' will be set to the actual no. of values returned.

Subroutine *STDRDR*

STDRDR(NO,DSC,FELM,MAXS,ACTS,VALS,DUN,DNUL,STAT)

INTEGER NO	*IN: file id of frame*
CHARACTER*(*) DSC	*IN: descriptor name (max 15 chars.)*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: max. no. of values to read*
INTEGER ACTS	*OUT: actual no. of values read*
REAL VALS(1)	*OUT: buffer for data*
INTEGER DUN	*OUT: units*
INTEGER DNUL	*OUT: no. of null values*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: Read real descriptor

Algorithm: 'ACTS' will be set to the actual no. of values returned.

Subroutine *STDWRC*

STDWRC(NO,DSC,NOELM,VALS,FELM,MAXS,DUN,STAT)

INTEGER NO	*IN: file id of frame*
CHARACTER*(*) DSC	*IN: descriptor name (max 15 chars.)*
INTEGER NOELM	*IN: no. of bytes per data value*
CHARACTER*(*) VALS	*IN: data buffer*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of values to write*
INTEGER DUN	*IN: units*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: Write character descriptor

Algorithm: A new descriptor is added to the descriptor area or data of an existing descriptor is modified and extended, if necessary.

Remarks: If 'FELM' = -1, descriptor values are appended to existing ones.
'NOELM' larger than 1 enables you to use character arrays, where each data value is in effect a char. string of 'NOELM' bytes.

Subroutine *STDWRD*

STDWRD(NO,DSC,VALS,FELM,MAXS,DUN,STAT)

INTEGER NO	*IN: file id of frame*
CHARACTER*(*) DSC	*IN: descriptor name (max 15 chars.)*
DOUBLE PRECISION VALS(1)	*IN: data buffer*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of values to write*
INTEGER DUN	*IN: units*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: Write double precision descriptor

Algorithm: A new descriptor is added to the descriptor area or data of an existing descriptor is modified and extended, if necessary.

Remarks: If 'FELM' = -1, descriptor values are appended to existing ones.

Subroutine *STDWRH*

STDWRH(NO,DSC,VALS,FELM,MAXS,STAT)

INTEGER NO	*IN: file id of frame*
CHARACTER*(*) DSC	*IN: descriptor name (max 15 chars.)*
CHARACTER*(*) VALS	*IN: buffer with help text*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of characters to write*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: Write help text of a descriptor

Algorithm: A help text is added for the descriptor area or the help text of an existing descriptor is modified and extended, if necessary.

Remarks: If 'FELM' = -1, the buffer values are appended to the help text.

Subroutine *STDWRI*

STDWRI(NO,DSC,VALS,FELM,MAXS,DUN,STAT)

INTEGER NO	*IN: file id of frame*
CHARACTER*(*) DSC	*IN: descriptor name (max 15 chars.)*
INTEGER VALS(1)	*IN: data buffer*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of values to write*
INTEGER DUN	*IN: units*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: Write integer descriptor

Algorithm: A new descriptor is added to the descriptor area or data of an existing descriptor is modified and extended, if necessary.

Remarks: If 'FELM' = -1, descriptor values are appended to existing ones.

Subroutine *STDWRR*

STDWRR(NO,DSC,VALS,FELM,MAXS,DUN,STAT)

INTEGER NO	*IN: file id of frame*
CHARACTER*(*) DSC	*IN: descriptor name (max 15 chars.)*
REAL VALS(1)	*IN: data buffer*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of values to write*
INTEGER DUN	*IN: units*
INTEGER STAT	*OUT: status (=0, o.k.)*

Purpose: write real descriptor

Algorithm: A new descriptor is added to the descriptor area or data of an existing descriptor is modified and extended, if necessary.

Remarks: If 'FELM' = -1, descriptor values are appended to existing ones.

Module	*Version [3.00] (15-Jan-1992)*	**Module STE**

Author: K. Banse ESO - Garching *Language:* F77

Classification: standard interfaces, errors

Environment:
- VMS and UNIX
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DEF.INC
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DAT.INC

Comments:

This module contains the routines to handle errors in MIDAS.
The functions provided are:

- Setup program behaviour in case of errors/warnings (STECNT)
- Terminate program in case of errors and signal errors (STETER, STESIG)

Subroutine *STECNT*

STECNT(ACTION,CONT,LOG,DISP)

CHARACTER*(*) ACTION	*IN: action = GET or PUT*
INTEGER CONT	*IN/OUT: continue flag*
INTEGER LOG	*IN/OUT: log flag*
INTEGER DISP	*IN/OUT: display flag*

Purpose: Set/clear error control flags.

Algorithm: Straight forward, but note that the parameters cannot be constants!

Remarks: Continue_flag
= -1 (stop on warnings and errors),
= 0 (stop on errors only),
= 1 (continue always).
Log_flag
= 0 (log nothing),
= 1 (log only errors),
= 2 (log warnings and errors).
Display_flag
= 0 (display nothing on the terminal),
= 1 (display errors),
= 2 (display warnings and errors).
In Midas the default is 0, 2 and 1 for the Continue, Log and Display_flag.

Subroutine *STETER*

STETER(ERRNO,TEXT)

INTEGER ERRNO	*IN: error number*
CHARACTER*(*) TEXT	*IN: error message*

Purpose: Display error message and abort.

Algorithm: display text, put error no. 'errno' into key PROGSTAT and stop execution

---

Subroutine *STESIG*

STESIG(SYS,TEXT,ERRNO)

CHARACTER*(*) SYS	*IN: subsystem*
CHARACTER*(*) TEXT	*IN: error message*
INTEGER ERRNO	*IN: error number*

Purpose: Signal an error to the calling routine.

Module	*Version [3.00] (15-Jan-1992)*	**Module STF**

Author: K. Banse ESO - Garching *Language:* F77

Classification: standard interfaces, bulk data frames

Environment:
- VMS and UNIX
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DEF.INC
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DAT.INC

Comments:

This module contains the routines to manipulate MIDAS data frames.
The functions provided are:

- Create, open and close data files (STFCRE, STFOPN, STFCLO)
- Get info about data files (STFINF)
- Rename and delete data files (STFDEL, STFRNM)
- Input and output of frame data (STFMAP, STFUNM, STFGET, STFPUT)

Main arguments used by the routines are:

NAME name of MIDAS data frame

NO file id of related MIDAS frame

DATTYP format how the data is stored on disk

FILTYP type of MIDAS data frame

IOMODE access mode of the frame

FELM first element to be accessed, the very first element is indicated by 1

STAT return status.

Subroutine *STFOPN*

STFOPN(NAME,DATTYP,NEWOPN,FILTYP,NO,STAT)

CHARACTER*(*) NAME	*IN: name of frame*
INTEGER DATTYP	*IN: data type*
INTEGER NEWOPN	*IN: new_open_flag: = 0 open normally, = 1 open same file again*
INTEGER FILTYP	*IN: file type*
INTEGER NO	*OUT: file id of frame*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Open a frame and return the internal file no.

Remarks: Supported data types are:
D_R4_FORMAT, D_R8_FORMAT, D_I4_FORMAT, D_I2_FORMAT,
D_UI2_FORMAT, D_I1_FORMAT and D_OLD_FORMAT.
Filetypes are:
F_IMA_TYPE, F_TBL_TYPE, F_FIT_TYPE and F_OLD_TYPE.

Subroutine *STFINF*

STFINF(NAME,FNO,IBUF,STAT)

CHARACTER*(*) NAME	*IN: name of frame*
INTEGER FNO	*IN: info_request flag*
INTEGER IBUF(5)	*OUT: returned info*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Open the frame (except for 'FNO'=0) and return the file info in buffer of up to 5 integers

Returns: return status (0 = o.k., else the required info cannot be obtained)

Remarks: 'FNO' = 0 for getting the file id ('NO') of 'NAME', i.e. check if frame 'NAME' has been opened already
'FNO' = 1 for getting version_no., file_type, short int, int and floating format
'FNO' = 2 for getting no_bytes_per_pixel, format,pixels_per_block, startblock and lastblock
'FNO' = 3 for getting file_type, format, file_protect, file_compress
'FNO' = 4 for getting no_bytes_per_pixel, format, file_type
'FNO' = 99 to find out if file 'NAME' exists,
no buffer is returned, instead the status is set to 0 (o.k. = file exists) or not

---

Subroutine *STFCLO*

STFCLO(NO,STAT)

INTEGER NO	*IN: file id of frame*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Close a frame

Algorithm: add frame to catalog (if enabled) and close file on disk

---

Subroutine *STFCRE*

STFCRE(NAME,DATTYP,IOMODE,FILTYP,SIZE,NO,STAT)

CHARACTER*(*) NAME	*IN: name of frame*
INTEGER DATTYP	*IN: data type, e.g. D_R4_FORMAT, ...*
INTEGER IOMODE	*IN: opening mode, e.g. F_O_MODE, ...*
INTEGER FILTYP	*IN: file type, e.g. F_TBL_TYPE, ...*
INTEGER SIZE	*IN: size of frame (= no. of data values in file)*
INTEGER NO	*OUT: file id of frame*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Create a frame with data of given type and return its internal file no.

Algorithm: fill all internal data structures and create file on disk

Remarks: Opening modes are: F_O_MODE, F_X_MODE,
see STFOPN for details of different data and file types

Subroutine *STFDEL*

STFDEL(NAME,STAT)

CHARACTER*(*) NAME	*IN: name of frame*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Delete a frame

Subroutine *STFMAP*

STFMAP(NO,IOMODE,FELM,SIZE,ACTSZ,PNTR,STAT)

INTEGER NO	*IN: file id of frame*
INTEGER IOMODE	*IN: mode*
INTEGER FELM	*IN: 1st pixel to map*
INTEGER SIZE	*IN: no. of pixels to map*
INTEGER ACTSZ	*OUT: actual no. mapped*
INTEGER PNTR	*OUT: pointer to mapped area*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Map a new data frame into memory for writing/reading
or just get virtual memory (iomode = F_X_MODE)

Algorithm: return pointer to first element which is required

Remarks: Modes are: F_O_MODE, F_IO_MODE, F_O_MODE, F_X_MODE

Subroutine *STFUNM*

STFUNM(NO,STAT)

INTEGER NO	*IN: file id of frame*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Unmap a mapped frame

Subroutine *STFGET*

STFGET(NO,FELM,SIZE,ACTSZ,BUFADR,STAT)

INTEGER NO	*IN: file id of frame*
INTEGER FELM	*IN: 1st pixel to be accessed*
INTEGER SIZE	*IN: no. of pixels to be read*
INTEGER ACTSZ	*OUT: actual no. read*
INTEGER BUFADR	*OUT: data buffer*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Read data from disk frame into memory

Algorithm: synchronous disk I/O, data are converted on the fly if necessary

Remarks: STFGET cannot be used for frames created with F_X_MODE.

Subroutine *STFPUT*

STFPUT(NO,FELM,SIZE,BUFADR,STAT)

INTEGER NO	*IN: file id of frame*
INTEGER FELM	*IN: 1st pixel to be accessed*
INTEGER SIZE	*IN: no. of pixels to be written*
INTEGER BUFADR	*IN: data buffer*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Write data from memory into disk frame

Algorithm: synchronous disk I/O, data are converted on the fly if necessary

Remarks: STFPUT cannot be used for frames created with F_X_MODE.

Subroutine *STFRNM*

STFRNM(OLDNAME,NEWNAME,STAT)

CHARACTER*(*) OLDNAME	*IN: old name of frame*
CHARACTER*(*) NEWNAME	*IN: new name of frame*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Rename a frame on disk

Remarks: Complete filename, i.e. including file type, have to be given.

Module	*Version [3.00] (15-Jan-1992)*	**Module STI**

Author: K. Banse ESO - Garching *Language:* F77

Classification: standard interfaces, data frames, descriptors

Environment:
- VMS and UNIX
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DEF.INC
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DAT.INC

Comments:

This module contains the (high level) routines to handle MIDAS images.
The functions provided are:

- Get image data and descriptors from disk (STIGET)
- Create image frame and fill its standard descriptors on disk (STIPUT). This routine does not fill in any data values!

Main arguments used by the routines are:

NAME name of MIDAS data frame.

NO file id of MIDAS frame.

DATTYP format how the data is stored on disk.

FILTYP type of MIDAS data frame.

IOMODE access mode of the frame.

PNTR pointer to mapped data.

STAT return status.

Subroutine *STIGET*

STIGET(NAME,DATTYP,IOMODE,FILTYP,MAXDIM, NAXIS,NPIX,START,STEP, IDENT,CUNIT,PNTR,NO,STAT)

CHARACTER*(*) NAME	*IN: name of frame*
INTEGER DATTYP	*IN: data type*
INTEGER IOMODE	*IN: opening mode, F_I_MODE or F_IO_MODE*
INTEGER FILTYP	*IN: file type*
INTEGER MAXDIM	*IN: max. no. of dimensions*
INTEGER NAXIS	*OUT: no. of dimensions*
INTEGER NPIX(1)	*OUT: no. of pixels per axis*
DOUBLE PRECISION START(1)	*OUT: start values*
DOUBLE PRECISION STEP(1)	*OUT: stepsizes*
CHARACTER*(*) IDENT	*OUT: ASCII identification for frame*
CHARACTER*(*) CUNIT	*OUT: unit of each axis + data*
INTEGER PNTR	*OUT: pointer to mapped data*
INTEGER NO	*OUT: file id of frame*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: map complete data frame for reading or updating and read the stand ard descriptors.

Algorithm: use STDRDx and STFMAP to map and to read the standard descriptors.

Remarks: The char. arrays IDENT and CUNIT have to be initialized before the call to STIGET, otherwise nothing may be returned!
For a detailed description of the parameters , see text for STF... and STD... routines

---

Subroutine *STIPUT*

STIPUT(NAME,DATTYP,IOMODE,FILTYP, NAXIS,NPIX,START,STEP, IDENT,CUNIT,PNTR,NO,STAT)

CHARACTER*(*) NAME	*IN: name of frame*
INTEGER DATTYP	*IN: data type*
INTEGER IOMODE	*IN: opening mode, F_O_MODE or F_X_MODE*
INTEGER FILTYP	*IN: file type*
INTEGER NAXIS	*IN: no. of dimensions*
INTEGER NPIX(1)	*IN: no. of pixels per axis*
DOUBLE PRECISION START(1)	*IN: start values*
DOUBLE PRECISION STEP(1)	*IN: stepsizes*
CHARACTER*(*) IDENT	*IN: ASCII identification for frame*
CHARACTER*(*) CUNIT	*IN: unit of each axis + data*
INTEGER PNTR	*OUT: pointer to mapped data*
INTEGER NO	*OUT: file id of frame*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Create new image frame, map it and store standard descriptors

Algorithm: call STFCRE, STFMAP and STDWRx routines

Remarks: for a detailed description of the parameters , see text for STF... and STD... routines

---

Module *Version [3.00] (15-Jan-1992)* **Module STK**

Author: K. Banse ESO - Garching *Language:* F77

Classification: standard interfaces, keyword data base

Environment:
- VMS and UNIX
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DEF.INC
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DAT.INC

Comments:

This module contains the routines to interact with the keywords of MIDAS.
The functions provided are:

- Find keywords and get information about them (STKFND, STKINF)
- Prompt for keywords (STKPRC, STKPRD, STKPRI, STKPRR)
- Read contents of keywords (STKRDC, STKRDD, STKRDI, STKRDR)
- Write contents of keywords (STKWRC, STKWRD, STKWRI, STKWRR)

Main arguments used by the routines are:

KEY name of keyword, max. 8 chars. long and must begin with an alphabetic char.

FELM first element to be accessed, the very first element is indicated by 1

MAXS maximum number of elements required.

ACTS actual number of elements accessed, value returned by the routine.

KUN unit of keyword, currently not implemented - just used as a placeholder for now.

KNUL no. of null values in keyword, currently not implemented - just used as a placeholder for now.

STAT return status.

Subroutine *STKFND*

STKFND(KEY,TYPE,NOELM,BYTELM,STAT)

CHARACTER*(*) KEY	*IN: keyword name (max. 8 chars)*
CHARACTER*(*) TYPE	*OUT: type of keyword*
INTEGER NOELM	*OUT: no. of elements in keyword*
INTEGER BYTELM	*OUT: bytes per element*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: find keyword 'KEY' and provide info about it

Algorithm: use low level functions to find key + related info

Remarks: returned TYPE = I for integer, = R for real, = D for double precision,
= C for character keywords; if key not found TYPE = ' ' (a blank) is returned

Subroutine *STKINF*

STKINF(NPOS,FNO,BUF,NUMBUF,STAT)

INTEGER NPOS	*IN: position of keyword in data base, beginning with 1*
INTEGER FNO	*IN: info flag: 1 = name, 2 = type, 3 = size*
CHARACTER*(*) BUF	*OUT: char. info buffer (FNO = 1, 2)*
INTEGER NUMBUF	*OUT: numerical info buffer (FNO = 3)*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Provide info about keyword at position 'NPOS'

Algorithm: go through the keywords directory and provide the desired info
for 'FNO' = 1,2 the character buffer is filled, else the numerical output buffer

---

Subroutine *STKPRC*

STKPRC(PROMPT,KEY,NOELM,FELM,MAXS,ACTS,VALS,KUN,KNUL,STAT)

CHARACTER*(*) PROMPT	*IN: prompt string*
CHARACTER*(*) KEY	*IN: keyword name (max. 8 chars)*
INTEGER NOELM	*IN: no. of bytes per data value*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of elements to get*
INTEGER ACTS	*OUT: actual no. of elements returned*
CHARACTER*(*) VALS	*OUT: buffer for data values*
INTEGER KUN	*OUT: units (not used yet...)*
INTEGER KNUL	*OUT: no. of null values (not used yet...)*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Display prompt string and read character keyword data from terminal

Algorithm: 'ACTS' will be set to the actual no. of values returned.

Remarks: 'NOELM' larger than 1 enables you to use character arrays, where each data value is in effect a char. string of 'NOELM' bytes.

Subroutine *STKPRD*

STKPRD(PROMPT,KEY,FELM,MAXS,ACTS,VALS,KUN,KNUL,STAT)

CHARACTER*(*) PROMPT	*IN: prompt string*
CHARACTER*(*) KEY	*IN: keyword name (max. 8 chars)*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of elements to get*
INTEGER ACTS	*OUT: actual no. of elements returned*
DOUBLE PRECISION VALS(1)	*OUT: buffer for data values*
INTEGER KUN	*OUT: units (not used yet...)*
INTEGER KNUL	*OUT: no. of null values (not used yet...)*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Display prompt string and read double precision keyword data from terminal

Algorithm: 'ACTS' will be set to the actual no. of values returned.

Subroutine *STKPRI*

STKPRI(PROMPT,KEY,FELM,MAXS,ACTS,VALS,KUN,KNUL,STAT)

CHARACTER*(*) PROMPT	*IN: prompt string*
CHARACTER*(*) KEY	*IN: keyword name (max. 8 chars)*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of elements to get*
INTEGER ACTS	*OUT: actual no. of elements returned*
INTEGER VALS(1)	*OUT: buffer for data values*
INTEGER KUN	*OUT: units (not used yet...)*
INTEGER KNUL	*OUT: no. of null values (not used yet...)*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Display prompt string and read integer keyword data from terminal

Algorithm: 'ACTS' will be set to the actual no. of values returned.

Subroutine *STKPRR*

STKPRR(PROMPT,KEY,FELM,MAXS,ACTS,VALS,KUN,KNUL,STAT)

CHARACTER*(*) PROMPT	*IN: prompt string*
CHARACTER*(*) KEY	*IN: keyword name (max. 8 chars)*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of elements to get*
INTEGER ACTS	*OUT: actual no. of elements returned*
REAL VALS(1)	*OUT: buffer for data values*
INTEGER KUN	*OUT: units (not used yet...)*
INTEGER KNUL	*OUT: no. of null values (not used yet...)*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Display prompt string and read real keyword data from terminal

Algorithm: 'ACTS' will be set to the actual no. of values returned.

Subroutine *STKRDC*

STKRDC(KEY,NOELM,FELM,MAXS,ACTS,VALS,KUN,KNUL,STAT)

CHARACTER*(*) KEY	*IN: keyword name (max. 8 chars)*
INTEGER NOELM	*IN: no. of bytes per data value*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of elements to get*
INTEGER ACTS	*OUT: actual no. of elements returned*
CHARACTER*(*) VALS	*OUT: buffer for data values*
INTEGER KUN	*OUT: units (not used yet...)*
INTEGER KNUL	*OUT: no. of null values (not used yet...)*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Read character keyword.

Algorithm: 'ACTS' will be set to the actual no. of values returned.

Remarks: 'NOELM' larger than 1 enables you to use character arrays, where each data value is in effect a char. string of 'NOELM' bytes.

Subroutine *STKRDD*

STKRDD(KEY,FELM,MAXS,ACTS,VALS,KUN,KNUL,STAT)

CHARACTER*(*) KEY	*IN: keyword name (max. 8 chars)*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of elements to get*
INTEGER ACTS	*OUT: actual no. of elements returned*
DOUBLE PRECISION VALS(1)	*OUT: buffer for data values*
INTEGER KUN	*OUT: units (not used yet...)*
INTEGER KNUL	*OUT: no. of null values (not used yet...)*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Read double precision keyword.

Algorithm: 'ACTS' will be set to the actual no. of values returned.

Subroutine *STKRDI*

STKRDI(KEY,FELM,MAXS,ACTS,VALS,KUN,KNUL,STAT)

CHARACTER*(*) KEY	*IN: keyword name (max. 8 chars)*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of elements to get*
INTEGER ACTS	*OUT: actual no. of elements returned*
INTEGER VALS(1)	*OUT: buffer for data values*
INTEGER KUN	*OUT: units (not used yet...)*
INTEGER KNUL	*OUT: no. of null values (not used yet...)*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Read integer keyword.

Algorithm: 'ACTS' will be set to the actual no. of values returned.

Subroutine *STKRDR*

STKRDR(KEY,FELM,MAXS,ACTS,VALS,KUN,KNUL,STAT)

CHARACTER*(*) KEY	*IN: keyword name (max. 8 chars)*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of elements to get*
INTEGER ACTS	*OUT: actual no. of elements returned*
REAL VALS(1)	*OUT: buffer for data values*
INTEGER KUN	*OUT: units (not used yet...)*
INTEGER KNUL	*OUT: no. of null values (not used yet...)*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Read real keyword.

Algorithm: 'ACTS' will be set to the actual no. of values returned.

Subroutine *STKWRC*

STKWRC(KEY,NOELM,VALS,FELM,MAXS,KUN,STAT)

CHARACTER*(*) KEY	*IN: keyword name (max. 8 chars)*
INTEGER NOELM	*IN: no. of bytes per data value*
CHARACTER*(*) VALS	*IN: buffer for data values*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of elements to write*
INTEGER KUN	*IN: units (not used yet...)*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Write character keyword.

Remarks: 'NOELM' larger than 1 enables you to use character arrays, where each data value is in effect a char. string of 'NOELM' bytes.

Subroutine *STKWRD*

STKWRD(KEY,VALS,FELM,MAXS,KUN,STAT)

CHARACTER*(*) KEY	*IN: keyword name (max. 8 chars)*
DOUBLE PRECISION VALS(1)	*IN: buffer for data values*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of elements to write*
INTEGER KUN	*IN: units (not used yet...)*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Write double precision keyword.

Subroutine *STKWRI*

STKWRI(KEY,VALS,FELM,MAXS,KUN,STAT)

CHARACTER*(*) KEY	*IN: keyword name (max. 8 chars)*
INTEGER VALS(1)	*IN: buffer for data values*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of elements to write*
INTEGER KUN	*IN: units (not used yet...)*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Write integer keyword.

Subroutine *STKWRR*

STKWRR(KEY,VALS,FELM,MAXS,KUN,STAT)

CHARACTER*(*) KEY	*IN: keyword name (max. 8 chars)*
REAL VALS(1)	*IN: buffer for data values*
INTEGER FELM	*IN: first element*
INTEGER MAXS	*IN: no. of elements to write*
INTEGER KUN	*IN: units (not used yet...)*
INTEGER STAT	*OUT: status (0 = o.k.)*

Purpose: Write real keyword.

Module *Version [3.00] (15-Jan-1994)* **Module STST**

Author: K. Banse ESO - Garching *Language:* F77

Classification: standard interfaces, MIDAS environment, text

Environment:
- VMS and UNIX
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DEF.INC
- INCLUDE MID_DISK:[MIDAS.'version'.INCL]ST_DAT.INC

Comments:

This module contains the routines to link into the MIDAS environment and to display text.
The functions provided are:

- Link to and unlink from the MIDAS environment (STSPRO, STSEPI)
- Display (and log) text (STTPUT, STTDIS)

Subroutine *STSPRO*

STSPRO(PROG)

CHARACTER∗(∗) PROG	*IN: program name*

Purpose: Get into the MIDAS environment.

Algorithm: Read keywords data base, initialize File Control Table and open logfile

Remarks: Set the program name to the string '-1' to run the program outside a MIDAS session (this implies that there will be no keywords available...)

Subroutine *STSEPI*

STSEPI

Purpose: Close down MIDAS environment and stop.

Subroutine *STTPUT*

STTPUT(TEXT,STAT)

CHARACTER∗(∗) TEXT	*IN: text string*
INTEGER STAT	*OUT: return status (0 = o.k.)*

Purpose: Display text and store it in MIDAS logfile.

Subroutine *STTDIS*

STTDIS(TEXT,FLAG,STAT)

CHARACTER∗(∗) TEXT	*IN: text string*
INTEGER BC	*IN: backspace count*
INTEGER STAT	*OUT: return status (0 = o.k.)*

Purpose: Display a given character string as it is; do not (!) store it in the logfile

Algorithm: backspace count = 0, write line normally
= 1, ..., 80, do that many backspaces, so the next message may overwrite

Appendix B

Table Interfaces

Module	*Version 1.0 (28-Feb-1991)*	**Module TCA**

Author: ESO-IPG, Garching *Language:* C

Classification: Table (3-D) Interfaces

Source file: $MIDASHOME/$MIDVERS/libsrc/tbl/tca.c

Environment:
- Midas
- #include <tblsys.h> /* Table System parameters */
- #include <tbldef.h> /* Symbols used for Tables */
- #include <midas_def.h> /* General MIDAS Symbols */
- #include <tblerr.h> /* List of Table Errors */
- #include <atype.h> /* Character classification */
- #include <macrogen.h> /* Classical macros */
- #include <math.h> /* System Library */

Comments:

This module contains the routines to handle the access to table elements, which *can be arrays* . Access to elements is done by column, row and index numbers; an array of elements is specified by column, row, index and length. The length is taken as the maximum possible when this argument is zero. element arrays in the table can be integer (I*1, i*2 or I*4), float, double precision or character string data types, as defined during the creation of the columns. Implicit conversion of the element value is done automatically if the input/output is done with a routine not corresponding to the data type.
The functions provided by this module are:

- Delete an array element (TCADEL),
- read an extract of an array element from the table (TCARD*x*),
- map an array element of the table (TCAMAP),
- write an extract of an array element into the table (TCAWR*x*),
- unmap an element of the table (TCAUNM),

Main arguments used by the routines are:

col sequential column number. It is an integer number provided by the system when a new column is created, or it is defined by the user for already existing columns.

index starting position in the element array. The first element is numbered 1.

items How many elements of the array are concerned. The value 0 stands for *all*

null null flag. This value is 1 if the element is undefined, 0 otherwise.

row sequential row number. It is an integer defining the row number in the table or the symbols LAST, FIRST, NEXT and PREVIOUS defined in the system file 'midas_def.h' in the directory 'MID_INCLUDE'. The sequence number is the physical sequence number (by default) or the sequence corresponding to the index of the reference column.

tid table identifier. It is an integer number provided by the system when the table is created or opened.

value element value. The type depends on the routine name. Implicit conversion is done if this type does not correspond to the column data type.

History: [1.0] 14-Dec-1990: Creation (Francois Ochsenbein)

Function *TCAMAP*

int TCAMAP(tid, row, col, addr)

int tid	*IN : table id*
int row	*IN : row number*
int col	*IN : column number*
char **addr	*OUT: element adress*

Purpose: Return the address of the COMPLETE element array

Returns: status

Function *TCAUNM*

int TCAUNM(tid, address)

int tid	*IN : table id*
char *address	*IN: column address*

Purpose: Unmap a part of the file that was mapped.

Returns: status (-1 if not mapped)

Function *TCAEDC*

int TCAEDC(tid, abin, col, index, items, buffer)

int tid	*IN : table id*
char *abin	*IN : values to edit*
int col	*IN : column number*
int index	*IN : index number*
int items	*IN : how many*
char *buffer	*OUT: edited value*

Purpose: Edit the element array "abin". The function is similar to TCARDC, but has the value in memory instead of in table (typically the result of TCAMAP).

Returns: status

Remarks: No null flag here (null values have buffer[0] = 0). Output buffer assumed to be large enough (size is returned by TBFGET)

Function *TCATRC*

int TCATRC(tid, abin, col, index, items, buffer)

int tid	*IN : table id*
char *abin	*OUT: value to edit*
int col	*IN : column number*
int index	*IN : index number*
int items	*IN : how many*
char *buffer	*IN : edited value*

Purpose: Interpret (TRanslate) the character value in buffer. This function is similar to TCAWRC, but converts the value to memory (no table access)

Returns: status

Remarks: No null flag here (null values have buffer[0] = 0).

Function *TCARDC*

int TCARDC(tid, row, col, index, items, value)

int tid	*IN : table id*
int row	*IN : row number*
int col	*IN : column number*
int index	*IN : index number*
int items	*IN : how many*
char *value	*OUT: actual value*

Purpose: Reads table element as a character string. Arrays are edited with a comma between elements.

Returns: status (error and non-selected)

Algorithm: Map first the element, then use TCAEDC to convert to char.

Function *TCARDD*

int TCARDD(tid, row, col, index, items, value)

int tid	*IN : table id*
int row	*IN : row number*
int col	*IN : column number*
int index	*IN : index number*
int items	*IN : how many*
double *value	*OUT: actual values*

Purpose: Reads table element as a double precision variable.

Returns: status

Function *TCARDI*

int TCARDI(tid, row, col, index, items, value)

int tid	*IN : table id*
int row	*IN : row number*
int col	*IN : column number*
int index	*IN : index number*
int items	*IN : how many*
int *value	*OUT: actual value*

Purpose: Reads table element as integer value.

Returns: status

Function *TCARDR*

int TCARDR(tid, row, col, index, items, value)

int tid	*IN : table id*
int row	*IN : row number*
int col	*IN : column number*
int index	*IN : index number*
int items	*IN : how many*
float *value	*OUT: actual value*

Purpose: Reads table element as float.

Returns: status

Function *TCADEL*

int TCADEL(tid, row, col, index, items)

int tid	*IN : table id*
int row	*IN : row number*
int col	*IN : column number*
int index	*IN : index number*
int items	*IN : how many*

Purpose: Deletes table elements.

Returns: status

Algorithm: Writes a NULL value in the table.

Function *TCAWRC*

int TCAWRC(tid, row, col, index, items, value)

int tid	*IN : table id*
int row	*IN : row number*
int col	*IN : column number*
int index	*IN : index number*
int items	*IN : how many*
char ∗value	*IN : actual value*

Purpose: Writes table element, character string format.

Returns: status

Function *TCAWRD*

int TCAWRD(tid, row, col, index, items, value)

int tid	*IN : table id*
int row	*IN : row number*
int col	*IN : column number*
int index	*IN : index number*
int items	*IN : how many*
double ∗value	*IN : actual value*

Purpose: Writes table element, double precision argument.

Returns: status

Function *TCAWRI*

int TCAWRI(tid, row, col, index, items, value)

int tid	*IN : table id*
int row	*IN : row number*
int col	*IN : column number*
int index	*IN : index number*
int items	*IN : how many*
int ∗value	*IN : actual value*

Purpose: Writes table element, integer variable as argument.

Returns: status

Function *TCAWRR*

int TCAWRR(tid, row, col, index, items, value)

int tid	*IN : table id*
int row	*IN : row number*
int col	*IN : column number*
int index	*IN : index number*
int items	*IN : how many*
float *value	*IN : actual value*

Purpose: Writes table element, float as input argument.

Returns: status

Function *TCASRC*

int TCASRC(tid, row, col, index, items, value, next)

int tid	*IN : table id*
int row	*IN : starting row number*
int col	*IN : column number*
int index	*IN : index number*
int items	*IN : how many items to compare*
char *value	*IN : Comparison vector*
int *next	*OUT: found row number*

Purpose: Search the first row that matches a set of values provided in character form. Search element in character string format. Search is performed on the table *tid*, examining rows from *row* on. It compares *items* elements of the column *col* from *index* with the array *value*, and stops when a match occurs. The result *next* is -1 if nothing appropriate was found.

Returns: status

Function *TCASRD*

int TCASRD(tid, row, col, index, items, value, next)

int tid	*IN : table id*
int row	*IN : starting row number*
int col	*IN : column number*
int index	*IN : index number in col*
int items	*IN : How many to compare*
double *value	*IN : Comparison vector*
int *next	*OUT: found row number*

Purpose: Search the first row that matches a set of values provided in double precision.

Returns: status

Function *TCASRI*

int TCASRI(tid, row, col, index, items, value, next)

int tid	*IN : table id*
int row	*IN : starting row number*
int col	*IN : column number*
int index	*IN : index number in col*
int items	*IN : How many to compare*
int ∗value	*IN : Comparison vector*
int ∗next	*OUT: found row number*

Purpose: Search the first row that matches a set of values provided in integer.

Returns: status

Function *TCASRR*

int TCASRR(tid, row, col, index, items, value, next)

int tid	*IN : table id*
int row	*IN : starting row number*
int col	*IN : column number*
int index	*IN : index number in col*
int items	*IN : How many to compare*
float ∗value	*IN : Comparison vector*
int ∗next	*OUT: found row number*

Purpose: Search the first row that matches a set of values provided in double precision.

Returns: status

External references:

<tblsys.h> <tbldef.h> <midas_def.h> <tblerr.h> <atype.h> <macrogen.h> <math.h>

Module	*Version 3.4 (28-Feb-1991)*	**Module TCC**

Author: J.D.Ponz IPG-ESO - Garching *Language:* C

Classification: table interface (Design 2.0)

Source file: $MIDASHOME/$MIDVERS/libsrc/tbl/tcc.c

Environment:
- #include <tblsys.h> /* Table System parameters */
- #include <tbldef.h> /* Symbols used for Tables */
- #include <tblerr.h> /* List of Table Errors */
- #include <atype.h> /* Character classification */
- #include <macrogen.h> /* Classical macros */

Comments:

This module contains the routines to handle columns in a table. Access to the table is done through the table identifier, returned by the routines TCTOPN and TCTINI. Access to a column is done through the column number, given by the routines TCCINI, TCCSER and TCLSER.
The following main functions are defined in this module:

- create a column in the table (TCCINI),
- delete a column from the table (TCCDEL),
- map a column (TCCMAP),
- unmap a column (TCCUNM),
- table sorting according to column values (TCCSRT),
- search the column number (TCCSER).

Main arguments used by the routines are:

alen array length. Integer number to define the number of items in a multiple column. In the case of character strings 'alen' defines the maximum length of the string.

colref reference to the column by label or by number. It is a character string containing the ':label' if the column is referenced by label, or '#n' if the column is referenced by its sequential number.

column sequential column number. It is an integer number provided by the system when a new column is created or defined by the user.

dtype column data type. The system file 'midas_def.h' in 'MID_INCLUDE:' provides the symbols D_xx_FORMAT where 'xx' is 'I1', 'I2' or 'I4' for integer data, 'R4' for float, 'R8' for double precision and 'C' for character.

form column format. Character string up to 6 characters. Display format associated to the column. The format is defined as in standard FORTRAN.

label column label. Character string (up to 16 characters) provided by the use when the column is created. Valid characters are letters, digits and the underscore symbol '_'. In the current version labels are NOT case independent.

tid table identifier. It is an integer number provided by the system when the table is created or opened.

unit column unit. Character string (up to 16 characters) to define the units of the column.

History: [1.0] 25-Mar-1987 Implementation JDP
[1.1] 20-Oct-1987 Modify calling sequence
[1.2] 23-Nov-1987 New SC calling sequence
[1.3] 11-Apr-1988 Include new routines without type conversion. Character string always ending with a '\0'. New SC calling sequence.
[1.4] 26-Sep-1988 Modify calling sequence in TCCINI to include data types and array length.
[3.0] 01-Jul-1990 New Version with Arrays / Elementary IO Added TCCSEL routine (selection of Columns)
[3.1] 26-Sep-1990 Removed bug in TCCSRT
[3.2] 06-Dec-1990 Force New Version as soon as something modified.
[3.3] 18-Dec-1990 FO: Take into account the number of selected rows.
[3.4] 23-Feb-1991 Sort always writes NULL values at the end.

Function *TCCDEL*

int TCCDEL(tid, column, ncol)

int tid	*IN: table id number*
int column	*IN: column number*
int *ncol	*OUT:actual number of columns*

Purpose: Delete column in a table

Returns: status

Algorithm: Just modify Descriptors.

Remarks: This implementation FORCES the table version to TBL_VERSION.

Function *TCCINI*

int TCCINI(tid, dtype, alen, form, unit, label, column)

int tid	*IN: table id number*
int dtype	*IN: column data type (Midas)*
int alen	*IN: no. of array items*
char *form	*IN: column form*
char *unit	*IN: column unit*
char *label	*IN: column label*
int *column	*OUT:column number*

Purpose: Initializes table column. Elements in the column are initialized as undefined (null values).

Returns: Status

Algorithm: Check if column has been already defined, otherwise add the new column descriptors. The routine provides an overflow mechanism when the new column exceeds the allocated space.

Function *TCCMAP*

int TCCMAP(tid, column, address)

int tid	*IN: table id number*
int column	*IN: column number*
char **address	*OUT:column address*

Purpose: Return the address of the column in the table.

Returns: status

Algorithm: Finds the position of the physical column and get address via the virtual memory mechanism.
Note: This routine only works on tables stored by columns (transposed format, parameter F_TRANS in TCTINI).

Function *TCCSEL*

int TCCSEL(tid, text, max_cols, cols, flags, found_cols)

int tid	*IN : table id*
char *text	*IN : textual designation*
int max_cols	*IN: Size of cols & flags arrays*
int *cols	*OUT: column numbers*
int *flags	*OUT: column flags (in brackets)*
int *found_cols	*OUT: How many columns were found*

Purpose: Searchs a set of columns column by references (their name).

Returns: Status

Algorithm: A set of columns has the general syntax col[(flag)][,col]...
where "col" may be #number or :label. The Sequence column may be designated by # or :

Function *TCCSER*

int TCCSER(tid, colref, column)

int tid	*IN : table id*
char *colref	*IN : column reference*
int *column	*OUT: column number*

Purpose: Searchs column by reference (its name). The column may be designated as #number or :name or name The Sequence column may be designated by # or :

Returns: Status

Algorithm: Sequential search for the column label. The routine returns the column number as the last argument or -1 if the column is not found.

Function *TCCSRT*

int TCCSRT(tid, nc, column, sortfl)

int tid	*IN : table id*
int nc	*IN : number of columns*
int column	*IN : column numbers*
int sortfl	*IN : sort flag*

Purpose: Sort table.

Returns: Status

Algorithm: Use Quick sort algorithm to sort the table data in ascending (sort flag > 0) or descending (sort flag < 0) sequence.

Function *TCCUNM*

int TCCUNM(tid, address)

int tid	*IN: table id number*
char ∗address	*IN: column address*

Purpose: Unmap the part of a file that was mapped before.

Returns: status (-1 if address not found)

Algorithm: Identical to TCEUNM

External references:

<tblsys.h> <tbldef.h> <tblerr.h> <atype.h> <macrogen.h>

Module	*Version 3.3 (24-Feb-1991)*	**Module TCD**

Author: IPG-ESO Garching *Language:* C

Classification: table interface (Design version 2.0)

Source file: $MIDASHOME/$MIDVERS/libsrc/tbl/tcd.c

Environment:
- #include <tblsys.h> /* Table System parameters */
- #include <tbldef.h> /* Symbols used for Tables */
- #include <macrogen.h> /* Classical macros */
- #include <atype.h> /* Character classification */
- #include <midas_def.h> /* General MIDAS Symbols */

Comments:

This module contains the routines to handle standard table descriptors:

- General table information giving the current number of rows and columns, the number of the sorted column (0 by default), the column acting as refence to access rows by value (0 by default) and the physical dimension of the table. The routine TCIGET gives this information.
- Table data types. The structure of one column may be retrieved with TCBGET(data type, number of items).
- Format to define the editing format. The following operations are implemented: Get (TCFGET), put (TCFPUT).
- Reference column. Defines the column used as reference for accessing the table elements by value. Possible operations are define reference column (TCKPUT) and get reference column (TCKGET).
- Label. Defines the name associated with the column. The label is used as a variable name associated with the column in programs and procedures. Possible operations are get label (TCLGET), put label (TCLPUT) and search column with a given label (TCLSER). The label is initialized by TCCINI.

  Case sensitivity for label names may be turned on via *TCOSET ("label_case", 1)* or off via *TCOSET ("label_case", 0).*
- Various *options* may be retrieved (TCOGET) or setup (TCOSET). TCOERR writes an error.
- Unit. Defines the units associated with a column. Possible operations are get unit (TCUGET) and put unit (TCUPUT). Units are initialized by TCCINI.
- Selection flag. Defines the current subset selected in the table. The functions included are: get selection (TCSGET), put selection on/off (TCSPUT), count selected items (TCSCNT) and initialize selection (TCSINI).

Main arguments used by the routines are:

column sequential column number. It is an integer number provided by the system when a new column is created or defined by the user.

dtype column data type. The system file midas_def.h in MID_INCLUDE provides the symbols D_xx_FORMAT where 'xx' is 'I1','I2' or 'I4' for integers, 'R4' for float, 'R8' for double precision and 'C' for characters

form column display format. It is a character string (up to 6 characters) defining a FORTRAN compatible format.

label column label. Character string (up to 16 characters) defined by the user when the column is created. Valid characters are letters, digits and the underscore '_'. In the current version labels are NOT case independent.

row sequential row number. It is an integer number defining the row number. The terms row and record are equivalent in this context.

tid table identifier. It is an integer number provided by the system when the table is created or opened.

unit column unit. It is a character string (up to 16 characters) to define the units of the column.

History:
[1.0] 25-Mar-1987 Implementation J.D. Ponz
[1.1] 25-Aug-1987 Include TCSPUT
[1.2] 20-Oct-1987 Calling sequence
[1.3] 23-Nov-1987 New SC calling sequence
[1.4] 11-Apr-1988 New SC calling sequence
[1.5] 11-Jan-1989 Include transposed file format
[1.6] 11-Jun-1989 New routines TCSCNT, TCDGET
[1.7] 14-Oct-1989 ibase = 1 in TCFSET (not yet implemented)
[1.8] 02-Jan-1990 TCLSER modified (null terminator)
[3.0] 01-Jul-1990 New Version with Arrays / Elementary IO New entry points TCBGET / TCOSET / TCOGET
[3.1] 28-Oct-1990 Added TCFED (Edition bin -> char) and TCFDV (Conversion char->bin)
[3.2] 07-Dec-1990 Added TCVERS (Version)
[3.3] 18-Dec-1990 Selected rows are stored in TBLCONTR descriptor. Added TCSSET routine to write the descriptor.

Function *TCBGET*

TCBGET(tid, column, dtype, items, bytes)

int tid	*IN : table ident*
int column	*IN : column number*
int *dtype	*OUT: data type*
int *items	*OUT: array size*
int *bytes	*OUT: storage required*

Purpose: Reads Binary storage characteristics of Column. The routine returns the data type (one of the D_xx_FORMAT), the number of items (arrays), and the number of bytes required for the binary storage.

Returns: status

Function *TCDGET*

TCDGET(tid, store)

int tid	*IN : table id*
int *store	*OUT: physical format on disk*

Purpose: Reads table storage format (F_TRANS or F_RECORD)

Returns: status

Function *TCFGET*

TCFGET(tid, column, form, len, dtype)

int tid	*IN : table ident*
int column	*IN : column number*
char *form	*OUT: column format*
int *len	*OUT: field length*
int *dtype	*OUT: data type*

Purpose: Reads column format. The routine returns the column format, the length of the column field when edited (bytes), and the data type.

Returns: status

Function *TCFPUT*

TCFPUT(tid, column, form)

int tid	*IN : table id*
int column	*IN : column number*
char *form	*IN : column format*

Purpose: Writes column format. Defines or changes the format of the column. The format is defined in FORTRAN compatible form.

Returns: status

Function *TCIGET*

TCIGET(tid, column, row, nsort, allcol, allrow)

int tid	*IN : table id*
int *column	*OUT: number of column*
int *row	*OUT: number of rows*
int *nsort	*OUT: sorted column*
int *allcol	*OUT: number of allocated words per record*
int *allrow	*OUT: number of allocated rows*

Purpose: Reads table size information.

Returns: status

Function *TCIPUT*

TCIPUT(tid, column, row)

int tid	*IN : table id*
int column	*IN: number of columns*
int row	*IN: number of rows*

Purpose: Writes number of table columns and rows. This routine has to be used in connection with mapping facility.

Returns: status

Remarks: Should this really be allowed ????

Function *TCKGET*

TCKGET(tid, column)

int tid	*IN : table id*
int ∗column	*OUT: number of column*

Purpose: Gets the sequential number of the reference column.

Returns: status

Function *TCKPUT*

TCKPUT(tid, column)

int tid	*IN : table id*
int column	*IN : column number*

Purpose: Defines the column to be used as reference to access rows by value.

Returns: status

Function *TCLGET*

TCLGET(tid, column, label)

int tid	*IN : table id*
int column	*IN : column number*
char ∗label	*OUT: label name*

Purpose: Reads column label.

Returns: status

Function *TCLPUT*

TCLPUT(tid, column, label)

int tid	*IN : table id*
int column	*IN : column number*
char ∗label	*IN : label name*

Purpose: Writes column label.

Returns: status

Function *TCLSER*

TCLSER(tid, label, column)

int tid	*IN : table id*
char *label	*IN : label name*
int *column	*OUT: column number*

Purpose: Searchs the sequential number of the column having the given label, case insensitive. Column is set to -1 when name is not found.

Returns: status.

Function *TCOERR*

TCOERR (tid, row, col, status, text)

int tid	*IN : table id*
int row	*IN: row number*
int col	*IN: column number*
int status	*IN: Status*
char *text	*IN: Text to add*

Purpose: Write an error

Returns: Status

Function *TCOGET*

TCOGET(op_name, value)

char *op_name	*IN : Name of option*
int *value	*OUT: Option's value*

Purpose: Gets various options;

Returns: Status

Remarks: Valid parameters are: "label_case" sensitivity to case of labels "display_error" to display the errors

---

Function *TCOSET*

TCOSET(op_name, value)

char *op_name	*IN : Name of option*
int value	*IN : Option's value*

Purpose: Set various options.

Returns: status

Remarks: See TCOGET for possible names of op_name.

Function *TCSGET*

TCSGET(tid, row, value)

int tid	*IN : table id*
int row	*IN : row number*
int *value	*OUT: selection flag (TRUE if selected)*

Purpose: Reads row selection flag.

Returns: status

Algorithm: Check if ALL selection flag in on. Otherwise, read selection flag.

Function *TCSPUT*

TCSPUT(tid, row, value)

int tid	*IN : table id*
int row	*IN : row number*
int *value	*IN: selection flag*

Purpose: Writes row selection flag: Writes the value (TRUE/FALSE) to indicate selection/not-selection of the row in the table.

Returns: status

Function *TCSCNT*

TCSCNT(tid,count)

int tid	*IN : table id*
int *count	*out: number of selected entries*

Purpose: Count selected entries.

Returns: status

Algorithm: Sequential scan of the selection flags.

Function *TCSINI*

TCSINI(tid)

int tid	*IN : table id*

Purpose: Initializes selection flag.

Returns: status

Function *TCSINF*

TCSINF(tid, line)

int tid	*IN : table id*
char *line	*OUT: selection info*

Purpose: Get selection information.

Returns: status

Function *TCSSET*

TCSSET(tid, line)

int tid	*IN : table id*
char *line	*IN : selection info*

Purpose: Set selection information. Note that the number of selected rows is not modified, unless line is blank which asks for an ALL selection.

Returns: status

Function *TCUGET*

TCUGET(tid, column, unit)

int tid	*IN : table id*
int column	*IN : column number*
char *unit	*OUT: unit name*

Purpose: Reads column unit.

Returns: status

Function *TCUPUT*

TCUPUT(tid, column, unit)

int tid	*IN : table id*
int column	*IN : column number*
char *unit	*IN : unit name*

Purpose: Writes column units.

Returns: status

Function *TCVERS*

int TCVERS(tid)

int tid	*IN : table id*

Purpose: Reads table Version

Returns: ERR_NORMAL if Correct Version / -n for Obsolete Versions

---

External references:

<tblsys.h> <tbldef.h> <macrogen.h> <atype.h> <midas_def.h>

Module *Version 3.3 (28-Feb-1991)* **Module TCE**

Author: IPG-ESO Garching *Language:* C

Classification: table interface.

Source file: $MIDASHOME/$MIDVERS/libsrc/tbl/tce.c

Environment:
- #include <tblsys.h> /* Table System parameters */
- #include <tbldef.h> /* Symbols used for Tables */
- #include <midas_def.h> /* General MIDAS Symbols */
- #include <tblerr.h> /* List of Table Errors */
- #include <atype.h> /* Character classification */
- #include <macrogen.h> /* Classical macros */
- #include <math.h> /* System Library */

Comments:

This module contains the routines to handle table elements. Access to elements is done by column and row numbers. Elements in the table can be integer, float, double precision and character string data types, as defined during the creation of the columns. Implicit conversion of the element value is done automatically if the input/output is done with a routine not corresponding to the data type.
The functions provided by this module are:

- Delete an element (TCEDEL),
- read an element from the table (TCERDx),
- map an element of the table (TCEMAP),
- write an element into the table (TCEWRx),
- search for an element value (TCESRx).
- unmap an element of the table (TCEUNM),

Main arguments used by the routines are:

column sequential column number. It is an integer number provided by the system when a new column is created, or it is defined by the user for already existing columns.

null null flag. This value is 1 if the element is undefined, 0 otherwise.

row sequential row number. It is an integer defining the row number in the table. The sequence number is the physical sequence number (by default) or the sequence corresponding to the index of the reference column.

tid table identifier. It is an integer number provided by the system when the table is created or opened.

value element value. The type depends on the routine name. Implicit conversion is done if this type does not correspond to the column data type. This variable is filled only if the value is defined.

In case of arrays, **only the first element** is returned for non-character data.

History:
[1.0] 25-Mar-1987 Implementation J.D. Ponz
[1.1] 21-Oct-1987 New calling sequence
[1.2] 11-Apr-1988 New calling sequence

[1.3] 26-Sep-1988 Use pointers for TCEWRx
[1.4] 16-Jan-1989 Include record file organization
[3.0] 01-Jul-1990 New Version with Arrays / Elementary IO
[3.1] 27-Sep-1990 Added TCEUNM
[3.2] 28-Oct-1990 Added TCEEDC (Edit, similar to TCERDC) and TCETRC (similar to TCEWRC)
[3.3] 18-Dec-1990 FO: Take into account the number of selected rows.

Function *TCEDEL*

TCEDEL(tid, row, column)

int tid	*IN : table id*
int row	*IN : row number*
int column	*IN : column number*

Purpose: Deletes table element.

Returns: status

Algorithm: Writes a NULL value in the table.

Function *TCEMAP*

TCEMAP(tid, row, column, address, null)

int tid	*IN: table id number*
int row	*IN: row number*
int column	*IN: column number*
char **address	*OUT: element adress*
int *null	*OUT: 1 if element null*

Purpose: Return the address of the element (which can be an array) in the table.

Returns: status

Remarks: The selection element cannot be mapped...

Function *TCEEDC*

TCEEDC(tid, abin, column, buffer)

int tid	*IN : table id*
char *abin	*IN : value to edit*
int column	*IN : column number*
char *buffer	*OUT: edited value*

Purpose: Read an element referred by abin. The function is similar to TCERDC, but has the value in memory instead of in table.

Returns: status

Remarks: No null flag here (null values have buffer[0] = 0). Output buffer assumed to be large enough (size is returned by TBFGET)

Function *TCERDC*

TCERDC(tid, row, column, value, null)

int tid	*IN : table id*
int row	*IN : row number*
int column	*IN : column number*
char ∗value	*OUT: actual value*
int ∗null	*OUT: null flag*

Purpose: Reads table element as a character string.

Returns: status (error and non-selected)

Algorithm: Finds the element position and convert data type if required. Arrays are edited with a comma between elements.

Function *TCERDD*

TCERDD(tid,row,column,value,null)

int tid	*IN : table id*
int row	*IN : row number*
int column	*IN : column number*
double ∗value	*OUT: actual value*
int ∗null	*OUT: null flag*

Purpose: Reads table element as a double precision variable.

Returns: status (error and non-selected)

Algorithm: Finds the element position and convert data type if required. Only the FIRST value is got for arrays.

Function *TCERDI*

TCERDI(tid,row,column,value,null)

int tid	*IN : table id*
int row	*IN : row number*
int column	*IN : column number*
int ∗value	*OUT: actual value*
int ∗null	*OUT: null flag*

Purpose: Reads table element as integer value.

Returns: status (error and non-selected)

Algorithm: Finds the element position and convert data type if required. Only the FIRST value is got for arrays.

Function *TCERDR*

TCERDR(tid,row,column,value,null)

int tid	*IN : table id*
int row	*IN : row number*
int column	*IN : column number*
float *value	*OUT: actual value*
int *null	*OUT: null flag*

Purpose: Reads table element as float.

Returns: status (error and non-selected)

Algorithm: Finds the element position and convert data type if required. Only the FIRST value is got for arrays.

Function *TCETRC*

TCETRC(tid, abin, column, buffer)

int tid	*IN : table id*
char *abin	*OUT: value to edit*
int column	*IN : column number*
char *buffer	*IN : edited value*

Purpose: Interpret (TRanslate) the character value in buffer. This function is similar to TCEWRC, but converts the value to memory (no table access)

Returns: status

Remarks: No null flag here (null values have buffer[0] = 0).

Function *TCEWRC*

TCEWRC(tid,row,column,value)

int tid	*IN : table id*
int row	*IN : row number*
int column	*IN : column number*
char *value	*IN : actual value*

Purpose: Writes table element, caracter string format.

Returns: status

Algorithm: Finds the element position and convert data type if required. Element in an array must be separated by ,

---

Function *TCEWRD*

TCEWRD(tid,row,column,value)

int tid	*IN : table id*
int row	*IN : row number*
int column	*IN : column number*
double *value	*IN : actual value*

Purpose: Writes table element, double precision argument.

Returns: status

Algorithm: Finds the element position and convert data type if required. Only the FIRST value is written for arrays.

Function *TCEWRI*

TCEWRI(tid,row,column,value)

int tid	*IN : table id*
int row	*IN : row number*
int column	*IN : column number*
int *value	*IN : actual value*

Purpose: Writes table element, integer variable as argument.

Returns: status

Algorithm: Finds the element position and convert data type if required. Only the FIRST value is written for arrays.

Function *TCEWRR*

TCEWRR(tid,row,column,value)

int tid	*IN : table id*
int row	*IN : row number*
int column	*IN : column number*
float *value	*IN : actual value*

Purpose: Writes table element, float as input argument.

Returns: status

Algorithm: Finds the element position and convert data type if required. Only the FIRST value is written for arrays.

Function *TCESRC*

TCESRC(tid, column, value, start, len, first, next)

int tid	*IN : table id*
int column	*IN : column number*
char ∗value	*IN : value to search*
int start	*IN : starting position in field*
int len	*IN : how many bytes to compare*
int first	*IN : starting row*
int ∗next	*OUT: found row number, -1 if not found*

Purpose: Search element in character string format.

Returns: status

Algorithm: Search element in the table, using binary or sequential search algorithm.
Note: A zero value for error is used for location on non-character columns.

Function *TCESRD*

TCESRD(tid,column,value,verror,first,next)

int tid	*IN : table id*
int column	*IN : column number*
double value	*IN : value to search*
double verror	*IN : tolerance*
int first	*IN : starting row number*
int ∗next	*OUT: found row number, -1 if not found*

Purpose: Search element in double precision format.

Returns: status

Algorithm: Search element in the table, using binary or sequential search algorithm.
Note: The routine can be used only with columns containing numerical data, implicit conversion of arguments is done if required.

Function *TCESRI*

TCESRI(tid, column, value, verror, first, next)

int tid	*IN : table id*
int column	*IN : column number*
int value	*IN : value to search*
int verror	*IN : tolerance*
int first	*IN : starting row number*
int ∗next	*OUT: found row number, -1 if not found*

Purpose: Search element in integer format.

Returns: status

Algorithm: Search element in the table, using binary or sequential search algorithm.

Note: The routine can be used only with columns containing numerical data, implicit conversion of arguments is done if required.

Function *TCESRR*

TCESRR(tid, column, value, verror, first, next)

int tid	*IN : table id*
int column	*IN : column number*
float value	*IN : value to search*
float verror	*IN : tolerance*
int first	*IN : starting row number*
int *next	*OUT: found row number, -1 if not found*

Purpose: Search element, floating point format.

Returns: status

Algorithm: Since, in C, argument in float is converted to double, this routine is identical to TCESRD.

Function *TCEUNM*

TCEUNM(tid, address)

int tid	*IN: table id number*
char *address	*IN: column address*

Purpose: Unmap a part of the file that was mapped.

Returns: status (-1 if not mapped)

Algorithm: this routine is identical to TCESRD.

External references:

<tblsys.h> <tbldef.h> <midas_def.h> <tblerr.h> <atype.h> <macrogen.h> <math.h>

Module *Version 3.0 (24-Feb-1991)* **Module TCM**

Author: J.D.Ponz IPG-ESO Garching *Language:* C

Classification: table interface (Design 2.0)

Source file: $MIDASHOME/$MIDVERS/libsrc/tbl/tcm.c

Environment:
- #include <tblsys.h> /* Table System parameters */
- #include <tbldef.h> /* Symbols used for Tables */
- #include <tblerr.h> /* List of Table Errors */

Comments: This module contains the routines handling the table structure as a matrix. The following operations can be performed:
- Allocate memory for the matrix (TCMALL)
- Release memory associated to the matrix (TCMFRE)
- Copy (selected) table entries into the matrix, with implicit type conversion (TCMGET)
- Copy the matrix into the (selected entries of the) table (TCMPUT)
- Obtain the bit pattern used for the representation of NULL values in memory (TCMNUL).

History:
[1.0] 25-Mar-1988 Implementation JDP
[3.0] 01-Jul-1990 New Version with Arrays / Elementary IO

Function *TCMALL*

TCMALL(type, ncol, nrow, pointer)

char *type	*IN: column type*
int ncol	*IN: number of columns*
int nrow	*IN: number of rows*
int *pointer	*OUT: pointer to the allocated matrix*

Purpose: Allocates memory for the matrix.

Returns: Status

Algorithm: The routine returns the pointer of the allocated memory for the matrix with 'ncol' columns and 'nrow' rows. The actual space allocated depends on the data type defined by the first argument. Up to 16 different memory areas can be allocated.

Function *TCMFRE*

TCMFRE(pointer)

int *pointer	*IN: pointer to the allocated matrix*

Purpose: Releases the memory associated to the pointer.

Returns: Status

Algorithm: The routine uses the pointer to identify the allocated memory and releases it.

Function *TCMGET*

TCMGET(tid, pointer, nac, nar)

int tid	*IN : table id*
int *pointer	*IN : pointer to the allocated matrix*
int *nac	*OUT: actual number of columns*
int *nar	*OUT: actual number of rows*

Purpose: Copies the table information into the matrix.

Returns: Status

Algorithm: The information in the table is converted into the matrix internal format, only selected columns and rows are considered. NULL values are also tranfered, use TCMNUL to find out the bit pattern used to represent NULLs in memory. The routine returns the actual number of columns and rows copied. The table must be open (TCTOPN or TCTINI).

Function *TCMPUT*

TCMPUT(tid, pointer, nac, nar)

int tid	*IN : table id*
int *pointer	*IN : pointer to the allocated matrix*
int nac	*IN : actual number of columns*
int nar	*IN : actual number of rows*

Purpose: Copies the matrix data into the table.

Returns: Status

Algorithm: The information in the matrix is converted into the table internal format, only selected columns and rows are considered. NULL values are also tranfered. The table must be open (TCTOPN or TCTINI).

Function *TCMCON*

TCMCON(tblsel, tdtrue, tdfalse)

float *tblsel	*OUT: selection flag on*
double *tdtrue	*OUT: double precision true*
double *tdfalse	*OUT: double precision false*

Purpose: Obtain the representation in memory of some system constants.

Returns: Status

Algorithm: The routine gives the selection flag and the true and false values in double precision.

Function *TCMNUL*

TCMNUL(inull, rnull, dnull)

int *inull	*OUT: integer null value*
float *rnull	*OUT: float null value*
double *dnull	*OUT: double null value*

Purpose: Obtain the representation in memory of NULL values.

Returns: Status

Algorithm: Depending on the system, the representation of NULL values in double and single precision are returned.

External references:

<tblsys.h> <tbldef.h> <tblerr.h>

Module *Version 3.2 (24-Feb-1991)* **Module TCR**

Author: IPG-ESO Garching *Language:* C

Classification: table interface (old table system)

Source file: $MIDASHOME/$MIDVERS/libsrc/tbl/tcr.c

Environment:
- #include <tblsys.h> /* Table System parameters */
- #include <tbldef.h> /* Symbols used for Tables */
- #include <tblerr.h> /* List of Table Errors */
- #include <atype.h> /* Character classification */
- #include <macrogen.h> /* Classical macros */
- #include <math.h> /* System Library */

Comments:

This module contains the routines to handle table rows. Access to elements is done by column and row numbers. Elements in the table can be integer, float, double precision and character string data types, as defined during the creation of the columns. Implicit conversion of the element value is done automatically if the input/output is done with a routine not corresponding to the data type.
The functions provided by this module are:

- Delete several elements in a row (TCRDEL),
- Read several row elements from the table (TCRRDx),
- Write several row elements into the table (TCRWRx).

Main arguments used by the routines are:

column sequential column number. Integer array defining the set of columns to be considered.

null null flag. It is an array with a value TRUE if the corresponding element is undefined, FALSE otherwise.

row sequential row number. The sequence number is the physical sequence number (by default) or the sequence corresponding to the index of the reference column.

tid table identifier. It is an integer number provided by the system when the table is created or opened.

value row values. The type depends on the routine name. Implicit conversion is done if this type does not correspond to the column data type.

History:
[1.0] 25-Mar-1987 Implementation J.D. Ponz
[1.1] 21-Oct-1987 Modification of the calling sequence
[1.2] 11-Apr-1988 Modification of the calling sequence
[1.3] 11-Jan-1989 Include record file organization
[1.4] 16-Oct-1989 indexing (ifirst-1)
[3.0] 01-Jul-1990 New Version with Arrays / Elementary IO Added Routine TCRSEL
[3.1] 27-Sep-1990 Added TCRMAP TCRUMP
[3.2] 18-Dec-1990: Take into account the number of selected rows.

Function *TCRDEL*

TCRDEL(tid, row)

int tid	*IN : table id*
int row	*IN : row number*

Purpose: Deletes all element in the row.

Returns: Status

Algorithm: uses TCEDEL.

Function *TCRRDC*

TCRRDC(tid, row, nc, column, value, null)

int tid	*IN : table id*
int row	*IN : row number*
int nc	*IN : number of cols.*
int *column	*IN : column numbers*
char *value	*OUT: buffer for actual value*
int *null	*OUT: null flag*

Purpose: Reads table row as a character string.

Returns: status

Algorithm: Access each single row element with implicit type conversion.

Function *TCRRDD*

TCRRDD(tid, row, nc, column, value, null)

int tid	*IN : table id*
int row	*IN : row number*
int nc	*IN : number of cols.*
int *column	*IN : column numbers*
double *value	*OUT: buffer for actual values*
int *null	*OUT: buffer for null flags*

Purpose: Reads table row as a double precision array.

Returns: status

Algorithm: Access each single row element with implicit type conversion.

Function *TCRRDI*

TCRRDI(tid, row, nc, column, value, null)

int tid	*IN : table id*
int row	*IN : row number*
int nc	*IN : number of cols.*
int *column	*IN : column numbers*
int *value	*OUT: buffer for actual values*
int *null	*OUT: buffer for null flags*

Purpose: Reads table row as in integer array.

Returns: status

Algorithm: Access each single row element with implicit type conversion.

Function *TCRRDR*

TCRRDR(tid, row, nc, column, value, null)

int tid	*IN : table id*
int row	*IN : row number*
int nc	*IN : number of cols.*
int *column	*IN : column numbers*
float value	*OUT: buffer for actual values*
int null	*OUT: buffer for null flags*

Purpose: Reads table row as floating point array.

Returns: status

Algorithm: Access each single row element with implicit type conversion.

Function *TCRSEL*

TCRSEL(tid, text, max_ranges, lower_bounds, upper_bounds, found_ranges)

int tid	*IN : table id*
char *text	*IN : textual designation*
int max_ranges	*IN: Size of lower& upper bound arrays*
int *lower_bounds	*OUT: lower bounds of found ranges*
int *upper_bounds	*OUT: upper bounds of found ranges*
int *found_ranges	*OUT: How many ranges were found*

Purpose: Select Ranges defined over the rows.

Returns: Status

Algorithm: A set of rows has the general syntax range[,range]...
where "range" is @lower_row[..[upper_row]]
Example: @1..10,@20,@100.. for rows number 1 to 10, 20 and over 100

Function *TCRWRC*

TCRWRC(tid, row, nc, column, value)

int tid	*IN : table id*
int row	*IN : row number*
int nc	*IN : number of cols.*
int *column	*IN : column numbers*
char *value	*IN : values to write*

Purpose: Writes table row

Returns: status

Algorithm: Access each row element with implicit type conversion.

Function *TCRWRD*

TCRWRD(tid, row, nc, column, value)

int tid	*IN : table id*
int row	*IN : row number*
int nc	*IN : number of cols.*
int *column	*IN : column numbers*
double *value	*IN : array with values*

Purpose: Writes table row from a double precision array.

Returns: status

Algorithm: Access each row element with implicit type conversion.

Function *TCRWRI*

TCRWRI(tid, row, nc, column, value)

int tid	*IN : table id*
int row	*IN : row number*
int nc	*IN : number of cols.*
int *column	*IN : column numbers*
int *value	*IN : data array*

Purpose: Writes table row from an integer array of values.

Returns: status

Algorithm: Access each row element with implicit type conversion.

Function *TCRWRR*

TCRWRR(tid, row, nc, column, value)

int tid	*IN : table id*
int row	*IN : row number*
int nc	*IN : number of cols.*
int *column	*IN : column numbers*
float *value	*IN : data array*

Purpose: Writes table row from a floating point data array.

Returns: status

Algorithm: Access each row element with implicit type conversion.

External references:

<tblsys.h> <tbldef.h> <tblerr.h> <atype.h> <macrogen.h> <math.h>

Module *Version 3.6 (25-Feb-1991)* **Module TCT**

Author: IPG-ESO Garching *Language:* C

Classification: table interface

Source file: $MIDASHOME/$MIDVERS/libsrc/tbl/tct.c

Environment:
- #include <tblsys.h> /* Table System parameters */
- #include <tbldef.h> /* Symbols used for Tables */
- #include <midas_def.h> /* General MIDAS Symbols */
- #include <tblerr.h> /* List of Table Errors */
- #include <macrogen.h> /* Classical macros */

Comments:

This module contains the routines to handle tables as a whole. Table initialization and open table functions return the table identification tid, used by other table routines.
Tables are physically stored on disk in two different formats: by records, corresponding to the natural way of storing sequentially the rows, and transposed, where all the values of a single variable – column – are stored together. It is the responsibility of the user to decide the physical format when initializing the table file.
The functions provided by this module are:

- Create or Initialize a table (TCTINI)
- Open an existing table (TCTOPN)
- Map an opened table to virtual memory (TCTMAP)
- Unmap a mapped table (TCTUNM): write modified parts, unmap mapped pieces
- Close a table (TCTCLO).

Main arguments used by the routines are:

allcol number of words per record allocated physically in the table file.

allrow number of records (rows) allocated physically in the table file.

mode file opening mode. There are several modes to open a table: F_I_MODE input (TBTOPN), F_O_MODE output (TBTINI), F_IO_MODE input/output (TBTOPN), F_X_MODE scratch (TBTINI), F_D_MODE descriptors only (TBTOPN). These symbols are provided in the file 'midas_def.h' in the directory 'MID_INCLUDE'.

The access mode may be forced to mapping with the F_MAP_FORCE option, or to elementary i/o mode with the F_EIO_FORCE option. The default is EIO mode for tables larger than TBL_EIO_LIMIT bytes, a configuration parameter which may be changed with TCOSET

name table file name. It is a character string defining the file name of the table as follows: '[path_name]table_name[.ext]', where 'path_name' is the directory path name, defaulted to the working directory, 'table_name' is the name of the file and 'ext' is the file extension, defaulted to 'tbl'.

storage physical file format. Defined as F_RECORD if the table is stored by records, or F_TRANS for transposed format. These symbols are provided in the file 'midas_def.h' in the directory 'MID_INCLUDE'.

tid table identifier. It is an integer number provided by the system when the table is created or opened.

History:
[1.0] 25-Mar-1987 Definition J.D. Ponz
[1.1] 21-Oct-1987 Modification of calling sequence
[1.2] 23-Nov-1987 New SC calling sequence
[1.3] 11-Apr-1988 New SC calling sequence
[3.0] 01-Jul-1990 New Version with Arrays / Elementary IO
[3.1] 27-Sep-1990 Added TCTFLU TCTMAP
[3.2] 17-Oct-1990 Added TCTID (Find ID from Name)
[3.3] 29-Oct-1990 Corrected bug in transformation old fmt=>new
[3.4] 06-Dec-1990 Keep Number identical to SC routines. Force change for new NULL values
[3.5] 18-Dec-1990: Added descriptor containaing number of selected rows.
[3.6] 24-Feb-1991: Always close the table, even in read-only mode

---

Function *TCTCLO*

int TCTCLO(tid)

int tid *IN : table id*

Purpose: Closes table file .

Returns: status

Algorithm: First Unmap, then release used memory.

Function *TCTCRV*

int TCTCRV(name, refer_name, mode)

char *name	*IN : Name of view to create*
char *refer_name	*IN : Name of related table*
int mode	*IN : creation mode (future)*

Purpose: Create a view (a table with only a Selection Column) taking columns from another (existing!) table.

Returns: Status

Algorithm: The view is ONLY CREATED. It must later be opened ...

Function *TCTUNM*

int TCTUNM(tid)

int tid *IN : table id*

Purpose: Unmap (Flush) table file

Returns: status

Algorithm: Writes back modified parts and releases used memory.

Function *TCTID*

int TCTID(name)

char *name	*IN: Name to look for*

Purpose: Find number from Name

Returns: Corresponding tid / -1 if not found

Function *TCTINI*

int TCTINI(name, storage, mode, allcol, allrow, tid)

char *name	*IN : table name*
int storage	*IN : physical structure on disk*
int mode	*IN : openning mode*
int allcol	*IN : number of words per record alloc.*
int allrow	*IN : number of rows allocated*
int *tid	*MOD: table identifier (try to keep it)*

Purpose: Initializes table file on disk and in memory

Returns: Status

Algorithm: Creates output file and opens it according to the desired storage option and mode. The storage option can be by records (F_RECORD) or transposed (F_TRANS). The openning mode can be F_O_MODE for output or F_X_MODE for scratch.
The mode may include modifiers F_MAP_FORCE to force the mapping, F_EIO_FORCE to force an elementary i/o mode, F_ALL_FORCE to force exact numbers of lines / cols. In this last case (F_ALL_FORCE), the number of physical columns to allocate may be given in the 2 leftmost bytes of mode, i.e. mode = (physical_columns*2**16 + F_ALL_FORCE + mode)

Function *TCTMAP*

int TCTMAP(tid, addr)

int tid	*IN : table id*
char **addr	*OUT: Where table Mapped*

Purpose: Maps complete Table

Returns: status

Algorithm: Use SCFMAP

Function *TCTOPN*

int TCTOPN(name,mode,tid)

char ∗name	*IN : table name*
int mode	*IN : openning mode*
int ∗tid	*MOD: table identifier (try to keep it)*

Purpose: Reads table file from disk.

Returns: Status

Algorithm: Opens table file according to the desired mode. The openning mode can be F_I_MODE for input, F_D_MODE for descriptors only, F_IO_MODE for update.
The mode may include modifiers F_MAP_FORCE to force the mapping, or F_EIO_FORCE to force an elementary i/o mode.

External references:

<tblsys.h> <tbldef.h> <midas_def.h> <tblerr.h> <macrogen.h>

Module *Version [1.00] (26-Apr-1989)* **Module TBC**

Author: J.D.Ponz ESO - Garching *Language:* F77

Classification: table interfaces.

Source file: MID_DISK:['MIDASHOME'.'MIDVERS'.LIBSRC.FTOC]TBC.FOR

Environment: • FORTRAN 77 and C standards

Comments: Contains layer between FORTRAN TB interfaces and the TZ interfaces. Version using the length of character strings as an explicit argument. Module related to table columns.

History: [[1.00]] 880408: created

Subroutine *TBCDEL*

TBCDEL(TID, COLUMN, NC, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : column number*
INTEGER NC	*OUT: actual number of columns*
INTEGER STATUS	*OUT: status*

Purpose: Delete column in a table

Subroutine *TBCINI*

TBCINI(TID, TYP, ALEN, FORM, UNIT, LABEL, COLUMN, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER TYP	*IN : column type*
INTEGER ALEN	*IN : no. of items*
CHARACTER*(*) FORM	*IN : column format*
CHARACTER*(*) UNIT	*IN : column unit*
CHARACTER*(*) LABEL	*IN : column label*
INTEGER COLUMN	*OUT: column number*
INTEGER STATUS	*OUT: status*

Purpose: Initialize a column in the table. The column type can be D_xx_FORMAT where 'xx' is 'I1', 'I2' or 'I4' for integer data, 'R4' for float, 'R8' for double precision and 'C' for character.

Subroutine *TBCMAP*

TBCMAP(TID, COLUMN, ADDRES, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : column number*
INTEGER ADDRES	*OUT: column address*
INTEGER STATUS	*OUT: status*

Purpose: Map column in a table

Subroutine *TBCSER*

TBCSER(TID, COLREF, COLUMN, STATUS)

INTEGER TID	*IN : table identifier*
CHARACTER*(*) COLREF	*IN : column reference*
INTEGER COLUMN	*OUT: column number*
INTEGER STATUS	*OUT: status*

Purpose: Search for column by reference (':label' or '#number')

Subroutine *TBCSRT*

TBCSRT(TID, NC, COLUMN, SFLAG, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER NC	*IN : number of columns to sort*
INTEGER COLUMN(NC)	*IN : column numbers*
INTEGER SFLAG(NC)	*IN : sort flag (>0 asc., <0 desc.)*
INTEGER STATUS	*OUT: status*

Purpose: Sort table according to the ascending or descending order of the columns.

External references:

MID_DISK:['MIDASHOME'.'MIDVERS'.INCL]ST_DEF.INC
MID_DISK:['MIDASHOME'.'MIDVERS'.INCL]ST_DAT.INC

Module	*Version [1.00] (26-Apr-1989)*	**Module TBD**

Author: J.D.Ponz ESO - Garching *Language:* F77

Classification: table interfaces.

Source file: MID_DISK:['MIDASHOME'.'MIDVERS'.LIBSRC.FTOC]TBD.FOR

Environment: • FORTRAN 77 and C standards

Comments: Contains layer between FORTRAN TB interfaces and the TZ interfaces. Version including the length of characters as explicit argument in the calling sequence. Module related to table descriptors.

History: [[1.00]] 880408: created

Subroutine *TBFGET*

TBFGET(TID, COLUMN, FORM, LFIELD, TYP, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : column number*
CHARACTER*(*) FORM	*OUT: column format*
INTEGER LFIELD	*OUT: field length*
INTEGER TYP	*OUT: column type*
INTEGER STATUS	*OUT: status*

Purpose: Reads column format

Subroutine *TBFPUT*

TBFPUT(TID, COLUMN, FORM, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : column number*
CHARACTER*(*) FORM	*IN : column format*
INTEGER STATUS	*OUT: status*

Purpose: Writes column format

Subroutine *TBIGET*

TBIGET(TID, COLUMN, ROW, NSC, ACOL, AROW, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*OUT: number of defined columns*
INTEGER ROW	*OUT: number of defined rows*
INTEGER NSC	*OUT: sorted column*
INTEGER ACOL	*OUT: number of words allocated per row*
INTEGER AROW	*OUT: number of rows allocated*
INTEGER STATUS	*OUT: status*

Purpose: Reads size parameters

Subroutine *TBIPUT*

TBIPUT(TID, COLUMN, ROW, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : number of defined columns*
INTEGER ROW	*IN : number of defined rows*
INTEGER STATUS	*OUT: status*

Purpose: Updates size parameters. This routine is only used in connection to the mapping of columns.

Subroutine *TBKGET*

TBKGET(TID, COLUMN, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*OUT: index of reference column*
INTEGER STATUS	*OUT: status*

Purpose: Get index of reference column

Subroutine *TBKPUT*

TBKPUT(TID, COLUMN, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : index of reference column*
INTEGER STATUS	*OUT: status*

Purpose: Defines index of reference column

Subroutine *TBLGET*

TBLGET(TID, COLUMN, LABEL, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : column number*
CHARACTER*(*) LABEL	*OUT: column label*
INTEGER STATUS	*OUT: status*

Purpose: Reads column label

Subroutine *TBLPUT*

TBLPUT(TID, COLUMN, LABEL, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : column number*
CHARACTER*(*) LABEL	*IN : column label*
INTEGER STATUS	*OUT: status*

Purpose: Writes column label

Subroutine *TBLSER*

TBLSER(TID, LABEL, COLUMN, STATUS)

INTEGER TID	*IN : table identifier*
CHARACTER*(*) LABEL	*IN : column label*
INTEGER COLUMN	*OUT: column number*
INTEGER STATUS	*OUT: status*

Purpose: Search for column by label. Column is set to -1 when the label is not found

---

Subroutine *TBSGET*

TBSGET(TID, ROW, VALUE, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
LOGICAL VALUE	*OUT: selection flag*
INTEGER STATUS	*OUT: status*

Purpose: Reads row selection flag

Subroutine *TBSPUT*

TBSPUT(TID, ROW, VALUE, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
LOGICAL VALUE	*IN : selection flag*
INTEGER STATUS	*OUT: status*

Purpose: Writes row selection flag

Subroutine *TBSINI*

TBSINI(TID, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER STATUS	*OUT: status*

Purpose: Initializes row selection flag

Subroutine *TBUGET*

TBUGET(TID, COLUMN, UNIT, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : column number*
CHARACTER*(*) UNIT	*OUT: column unit*
INTEGER STATUS	*OUT: status*

Purpose: Reads column unit

Subroutine *TBUPUT*

TBUPUT(TID, COLUMN, UNIT, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : column number*
CHARACTER*(*) UNIT	*IN : column unit*
INTEGER STATUS	*OUT: status*

Purpose: Writes column unit

External references:

MID_DISK:['MIDASHOME'.'MIDVERS'.INCL]ST_DEF.INC
MID_DISK:['MIDASHOME'.'MIDVERS'.INCL]ST_DAT.INC

Module	*Version [1.02] (26-Apr-1989)*	**Module TBE**

Author: J.D.Ponz ESO - Garching *Language:* F77

Classification: table interfaces.

Source file: MID_DISK:['MIDASHOME'.'MIDVERS'.LIBSRC.FTOC]TBE.FOR

Environment: • FORTRAN 77 and C standards

Comments: Contains layer between FORTRAN TB interfaces and the TZ interfaces. Version including the length of characters as explicit argument in the calling sequence. Module related to table elements.

History: [[1.00]] 880408: created
[[1.01]] 881012: temporary names for tbesrX
[[1.02]] 890112: Include routines TBESRx

Subroutine *TBEGET*

TBEGET(TID, TYPE, ROW, COLUMN, VALUE, NULL, STATUS)

INTEGER TID	*IN : table identifier*
CHARACTER*(*) TYPE	*IN : data type*
INTEGER ROW	*IN : row number*
INTEGER COLUMN	*IN : column number*
INTEGER VALUE(1)	*OUT: table element*
LOGICAL NULL	*OUT: null flag*
INTEGER STATUS	*OUT: status*

Purpose: Reads table element variable type (INCLUDED FOR COMPATIBILITY)

Subroutine *TBEPUT*

TBEPUT(TID, TYPE, ROW, COLUMN, VALUE, STATUS)

INTEGER TID	*IN : table identifier*
CHARACTER*(*) TYPE	*IN : argument type*
INTEGER ROW	*IN : row number*
INTEGER COLUMN	*IN : column number*
INTEGER VALUE(1)	*IN : table element*
INTEGER STATUS	*OUT: status*

Purpose: Writes table element (INCLUDED FOR COMPATIBILITY)

Subroutine *TBEDEL*

TBEDEL(TID, ROW, COLUMN, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER COLUMN	*IN : column number*
INTEGER STATUS	*OUT: status*

Purpose: Deletes table element (set to null)

Subroutine *TBERDC*

TBERDC(TID, ROW, COLUMN, VALUE, NULL, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER COLUMN	*IN : column number*
CHARACTER*(*) VALUE	*OUT: table element*
LOGICAL NULL	*OUT: null flag*
INTEGER STATUS	*OUT: status*

Purpose: Reads table element as a character string

Subroutine *TBERDD*

TBERDD(TID, ROW, COLUMN, VALUE, NULL, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER COLUMN	*IN : column number*
DOUBLE PRECISION VALUE	*OUT: table element*
LOGICAL NULL	*OUT: null flag*
INTEGER STATUS	*OUT: status*

Purpose: Reads table element as a double precision variable

Subroutine *TBERDI*

TBERDI(TID, ROW, COLUMN, VALUE, NULL, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER COLUMN	*IN : column number*
INTEGER VALUE	*OUT: table element*
LOGICAL NULL	*OUT: null flag*
INTEGER STATUS	*OUT: status*

Purpose: Reads table element as an integer variable

Subroutine *TBERDR*

TBERDR(TID, ROW, COLUMN, VALUE, NULL, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER COLUMN	*IN : column number*
REAL VALUE	*OUT: table element*
LOGICAL NULL	*OUT: null flag*
INTEGER STATUS	*OUT: status*

Purpose: Reads table element as a real variable

Subroutine *TBEWRC*

TBEWRC(TID, ROW, COLUMN, VALUE, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER COLUMN	*IN : column number*
CHARACTER*(*) VALUE	*IN : table element*
INTEGER STATUS	*OUT: status*

Purpose: Writes table element as a character string

Subroutine *TBEWRD*

TBEWRD(TID, ROW, COLUMN, VALUE, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER COLUMN	*IN : column number*
DOUBLE PRECISION VALUE	*IN : table element*
INTEGER STATUS	*OUT: status*

Purpose: Writes table element as a double precision variable

Subroutine *TBEWRI*

TBEWRI(TID, ROW, COLUMN, VALUE, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER COLUMN	*IN : column number*
INTEGER VALUE	*IN : table element*
INTEGER STATUS	*OUT: status*

Purpose: Writes table element as an integer variable

Subroutine *TBEWRR*

TBEWRR(TID, ROW, COLUMN, VALUE, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER COLUMN	*IN : column number*
REAL VALUE	*IN : table element*
INTEGER STATUS	*OUT: status*

Purpose: Writes table element as a real variable

Subroutine *TBESRC*

TBESRC(TID, COLUMN, VALUE, START, NB, FIRST, NEXT, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : column number*
CHARACTER*(*) VALUE	*IN : string of characters*
INTEGER START	*IN : starting field position*
INTEGER NB	*IN : number of bytes in the field*
INTEGER FIRST	*IN : row number to start searching*
INTEGER NEXT	*OUT: row number of the element found*
INTEGER STATUS	*OUT: status*

Purpose: Search character string in a column.

Subroutine *TBESRD*

TBESRD(TID, COLUMN, VALUE, ERROR, FIRST, NEXT, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : column number*
DOUBLE PRECISION VALUE	*IN : value to search*
DOUBLE PRECISION ERROR	*IN : tolerance in the search*
INTEGER FIRST	*IN : row number to start searching*
INTEGER NEXT	*OUT: row number of the element found*
INTEGER STATUS	*OUT: status*

Purpose: Search element in a column, double precision version.

Subroutine *TBESRI*

TBESRI(TID, COLUMN, VALUE, ERROR, FIRST, NEXT, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : column number*
INTEGER VALUE	*IN : value to search*
INTEGER ERROR	*IN : error in the search*
INTEGER FIRST	*IN : row number to start searching*
INTEGER NEXT	*OUT: row number of the element found*
INTEGER STATUS	*OUT: status*

Purpose: Search element in a column (integer variable)

Subroutine *TBESRR*

TBESRR(TID, COLUMN, VALUE, ERROR, FIRST, NEXT, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER COLUMN	*IN : column number*
REAL VALUE	*IN : value to search*
REAL ERROR	*IN : error in the search*
INTEGER FIRST	*IN : row number to start searching*
INTEGER NEXT	*OUT: row number of the element found*
INTEGER STATUS	*OUT: status*

Purpose: Search element in a column, real variable

Module	*Version [1.00] (26-Apr-1989)*	**Module TBM**

Author: J.D.Ponz ESO - Garching *Language:* F77

Classification: table interfaces.

Source file: MID_DISK:['MIDASHOME'.'MIDVERS'.LIBSRC.FTOC]TBM.FOR

Environment: • FORTRAN 77 and C standards

Comments: Contains layer between FORTRAN TB interfaces and the TZ interfaces. Version including the length of characters as explicit argument in the calling sequence. Module related to memory mapping in the context of tables.

History: [[1.00]] 880408: created

Subroutine *TBMCON*

TBMCON(TBLSEL, TDTRUE, TDFALS)

REAL TBLSEL	*OUT: selection flag*
DOUBLE PRECISION TDTRUE	*OUT: double precision true*
DOUBLE PRECISION TDFALS	*OUT: double precision false*

Purpose: Get some constants

Subroutine *TBMNUL*

TBMNUL(TINULL, TRNULL, TDNULL)

INTEGER TINULL	*OUT: integer null*
REAL TRNULL	*OUT: real null*
DOUBLE PRECISION TDNULL	*OUT: double precision null*

Purpose: Get null value representation

Module	*Version [1.00] (26-Apr-1989)*	**Module TBR**

Author: J.D.Ponz ESO - Garching *Language:* F77

Classification: table interfaces.

Source file: MID_DISK:['MIDASHOME'.'MIDVERS'.LIBSRC.FTOC]TBR.FOR

Environment: • FORTRAN 77 and C standards

Comments: Contains layer between FORTRAN TB interfaces and the TZ interfaces. Version including the length of characters as explicit argument in the calling sequence. Module related to table rows.

History: [[1.00]] 880408: created

Subroutine *TBRGET*

TBRGET(TID, TYPE, ROW, NC, COLUMN, VALUE, NULL, STATUS)

INTEGER TID	*IN : table identifier*
CHARACTER*(*) TYPE	*IN : data type*
INTEGER ROW	*IN : row number*
INTEGER NC	*IN : number of columns*
INTEGER COLUMN(NC)	*IN : column number*
INTEGER VALUE(1)	*OUT: table element*
LOGICAL NULL(NC)	*OUT: null flag*
INTEGER STATUS	*OUT: status*

Purpose: Reads table row variable type (Implemented for compatibility mode)

Subroutine *TBRPUT*

TBRPUT(TID, TYPE, ROW, NC, COLUMN, VALUE, STATUS)

INTEGER TID	*IN : table identifier*
CHARACTER*(*) TYPE	*IN : argument type*
INTEGER ROW	*IN : row number*
INTEGER NC	*IN : number of columns*
INTEGER COLUMN(NC)	*IN : column number*
INTEGER VALUE(1)	*IN : table element*
INTEGER STATUS	*OUT: status*

Purpose: Writes table row (Implemented for compatibility mode)

Subroutine *TBRDEL*

TBRDEL(TID, ROW, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER STATUS	*OUT: status*

Purpose: Deletes table row (set to null)

Subroutine *TBRRDC*

TBRRDC(TID, ROW, NC, COLUMN, VALUE, NULL, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER NC	*IN : number of columns*
INTEGER COLUMN(NC)	*IN : column index*
CHARACTER*(*) VALUE	*OUT: table row*
LOGICAL NULL(NC)	*OUT: null flag*
INTEGER STATUS	*OUT: status*

Purpose: Reads table row as a character string

Subroutine *TBRRDD*

TBRRDD(TID, ROW, NC, COLUMN, VALUE, NULL, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER NC	*IN : number of columns*
INTEGER COLUMN(NC)	*IN : column index*
DOUBLE PRECISION VALUE(NC)	*OUT: table row*
LOGICAL NULL(NC)	*OUT: null flag*
INTEGER STATUS	*OUT: status*

Purpose: Reads table row as a double precision variable

Subroutine *TBRRDI*

TBRRDI(TID, ROW, NC, COLUMN, VALUE, NULL, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER NC	*IN : number of columns*
INTEGER COLUMN(NC)	*IN : column index*
INTEGER VALUE(NC)	*OUT: table row*
LOGICAL NULL(NC)	*OUT: null flag*
INTEGER STATUS	*OUT: status*

Purpose: Reads table row as an integer variable

Subroutine *TBRRDR*

TBRRDR(TID, ROW, NC, COLUMN, VALUE, NULL, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER NC	*IN : number of columns*
INTEGER COLUMN(NC)	*IN : column index*
REAL VALUE(NC)	*OUT: table row*
LOGICAL NULL(NC)	*OUT: null flag*
INTEGER STATUS	*OUT: status*

Purpose: Reads table row as a real variable

Subroutine *TBRWRC*

TBRWRC(TID, ROW, NC, COLUMN, VALUE, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER NC	*IN : number of columns*
INTEGER COLUMN(NC)	*IN : column index*
CHARACTER*(*) VALUE	*IN : table row*
INTEGER STATUS	*OUT: status*

Purpose: Writes table row as a character string

Subroutine *TBRWRD*

TBRWRD(TID, ROW, NC, COLUMN, VALUE, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER NC	*IN : number of columns*
INTEGER COLUMN(NC)	*IN : column index*
DOUBLE PRECISION VALUE(NC)	*IN : table row*
INTEGER STATUS	*OUT: status*

Purpose: Writes table row as a double precision variable

Subroutine *TBRWRI*

TBRWRI(TID, ROW, NC, COLUMN, VALUE, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER NC	*IN : number of columns*
INTEGER COLUMN(NC)	*IN : column index*
INTEGER VALUE(NC)	*IN : table row*
INTEGER STATUS	*OUT: status*

Purpose: Writes table row as an integer variable

Subroutine *TBRWRR*

TBRWRR(TID, ROW, NC, COLUMN, VALUE, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER ROW	*IN : row number*
INTEGER NC	*IN : number of columns*
INTEGER COLUMN(NC)	*IN : column index*
REAL VALUE(NC)	*IN : table row*
INTEGER STATUS	*OUT: status*

Purpose: Writes table row as a real variable

Module	*Version [1.00] (26-Apr-1989)*	**Module TBT**

Author: J.D.Ponz ESO - Garching *Language:* F77

Classification: table interfaces.

Source file: MID_DISK:['MIDASHOME'.'MIDVERS'.LIBSRC.FTOC]TBT.FOR

Environment: • FORTRAN 77 and C standards

Comments: Contains layer between FORTRAN TB interfaces and the TZ interfaces. Version using the length of character strings as an explicit argument. Module related to the table handling.

History: [[1.00]] 880408: created

Subroutine *TBTCLO*

TBTCLO(TID, STATUS)

INTEGER TID	*IN : table identifier*
INTEGER STATUS	*OUT: status*

Purpose: Closes table file

Subroutine *TBTINI*

TBTINI(NAME, STORE, MODE, ACOL, AROW, TID, STATUS)

CHARACTER*(*) NAME	*IN : table name*
INTEGER STORE	*IN : physical structure on disk (0/1)*
INTEGER MODE	*IN : opening mode*
INTEGER ACOL	*IN : number of words per row*
INTEGER AROW	*IN : number of rows*
INTEGER TID	*OUT: table identifier*
INTEGER STATUS	*OUT: status*

Purpose: Initialize and open table file. STORE defines the physical allocation on disk as : F_RECORD table stored per records, F_TRANS table stored in transposed form. MODE defines the mode of file access as : F_O_MODE for output or F_X_MODE for scratch.

Subroutine *TBTOPN*

TBTOPN(NAME, MODE, TID, STATUS)

CHARACTER*(*) NAME	*IN : table name*
INTEGER MODE	*IN : opening mode*
INTEGER TID	*OUT: table identifier*
INTEGER STATUS	*OUT: status*

Purpose: Opens table file. MODE defines the mode of file access as F_I_MODE for input or F_IO_MODE for update, F_D_MODE for descriptors only. only.

External references:

MID_DISK:['MIDASHOME'.'MIDVERS'.INCL]ST_DEF.INC
MID_DISK:['MIDASHOME'.'MIDVERS'.INCL]ST_DAT.INC

Appendix C

Graphics Interfaces

Module *Version 1.0 (27-Oct-1993)* **Module PCA**

Author: R.M. van Hees ESO-IPG, Garching *Language:* C

Classification: High Level Graphics Interfaces

Source file: $MIDASHOME/$MIDVERS/libsrc/plot/pca.c

Environment:
- Midas and AGL
- #include <agl.h> /* Prototypes for AGL application programs */
- #include <midas_def.h> /* Prototypes for MIDAS interfaces */
- #include <plot_def.h> /* Symbols used by the PLT interfaces */

Comments: The module contains the routines to draw a coordinate box. The main arguments used by the routines are:

xmnmx floating point array holding minimum and maximum value along the X-axis,

ymnmx floating point array holding minimum and maximum value along the Y-axis,

labelx character array to be drawn as label along the X-axis

labely character array to be drawn as label along the Y-axis

xwcfrm floating point array with the definition of the X-axis: amin, amax, abig ticksteps, asmall tickstep

ywcfrm floating point array with the definition of the Y-axis: amin, amax, abig ticksteps, asmall tickstep

AGLopt character array with AGL options, see AGL reference manual.

History: [1.0] 23-Sep-1993: Creation (R.M. van Hees)

Function *PCAXES*

void PCAXES(xmnmx, ymnmx, labelx, labely, AGLopt)

float *xmnmx	*IN : minimum and maximum value along the X-axis*
float *ymnmx	*IN : minimum and maximum value along the Y-axis*
char *labelx	*IN : label along X-axis*
char *labely	*IN : label along Y-axis*
char *AGLopt	*IN : AGL option string, see AGL ref. manual (AG_AXES)*

Purpose: Computes tick marks, and draws axes including tick marks and labels

Algorithm: PCAXES sets the clipping area so that it encloses the box defined by the axes. In practice this means that you can call PCDATA and PCHIST to plot your data within the drawn frame. The clipping area is NOT taken from the MIDAS keywords! The character dimension is not affected by PCAXES. PCAXES uses the AGL routine AG_AXES.

Function *PCFRAM*

void PCFRAM(xwcfrm, ywcfrm, labelx, labely)

float *xwcfrm	*IN : definition of the X axis*
float *ywcfrm	*IN : definition of the Y axis*
char *labelx	*IN : label along the X axis*
char *labely	*IN : label along the Y axis*

Purpose: Draws a standard MIDAS coordinate frame including tick marks and labels

Algorithm: PCFRAM draws a standard MIDAS frame including tick marks and labels. Font, character size, line width and text thickness are defined by the current MIDAS setting. PCFRAM uses the AGL routine AG_AXIS.

External references:

<agl.h>	<midas_def.h>	<plot_def.h>
<stdio.h>	<string.h>	
<math.h>		

Module *Version 1.0 (27-Oct-1993)* **Module PCD**

Author: R.M. van Hees ESO-IPG, Garching *Language:* C

Classification: High Level Graphics Interfaces

Source file: $MIDASHOME/$MIDVERS/libsrc/plot/pcd.c

Environment:
- Midas and AGL
- #include <agl.h> /* Prototypes for AGL application programs */
- #include <plot_def.h> /* Symbols used by the PLT interfaces */
- #include <midas_def.h> /* Prototypes for MIDAS interfaces */

Comments: The module contains the routine for plotting data points by lines, symbols, and histogram. In addition the module takes care of plotting text strings. The main arguments used by the routines are:

stype Integer number defining the symbol type used, see SET/GRAPHIC options,

ltype Integer number defining the line type used, see SET/GRAPHIC options,

binmod Integer number: (0) bin mode OFF; (1) bin mode ON

fopt floating point array, [0] histogram type selector: (= 0) simple staircase, (= 1) staircase steps joined to X-axis, (> 1) boxes with width starting from 0 (simple line) and increasing with mode value in steps of small character width, [1] space between the filling lines, [2] angle of the filling lines w.r.t. X-axis.

ipos Integer number defining the centering position: (= 1) starts at (xc, yc), (= 2) ends at (xc, yc), or centred on (xc, yc)

History: [1.0] 23-Sep-1993: Creation (R.M. van Hees)

Function *PCDATA*

void PCDATA(stype, ltype, binmod, x_data, y_data, y_off, nrdata)

int stype	*IN : symbol type*
int ltype	*IN : line type*
int binmod	*IN : bin mode*
float *x_data	*IN : X values of the data points*
float *y_data	*IN : Y values of the data points*
float y_off	*IN : offset along the Y-axis*
int nrdata	*IN : number of points to be plotted*

Purpose: Plot a set of points connected with lines, or indicated by symbols

Algorithm: The data points are connected with lines if ltype is not equal to zero. Symbols are only plotted when ltype equals zero. A graphic device must be defined. Coordinates refer to the current active coordinate system

---

Function *PCHIST*

void PCHIST(nbins, cl, rfr, fopt)

int nbins	*IN : number of data points*
float *cl	*IN : values along the X-axis*
float *rfr	*IN : number of occurrences per bin*
float fopt[ ]	*IN : histogram options*

Purpose: PCHIST draws a histogram with an option to fill

Algorithm: A graphic device must be defined. Coordinates refer to the current active coordinate system. PCHIST uses the AGL routines AG_HIST and AG_FILL.

Function *PCTEXT*

void PCTEXT(text, xc, yc, angle, chsiz, ipos)

char *text	*IN : text to be drawn*
float xc	*IN : string position in X*
float yc	*IN : string position in Y*
float angle	*IN : text angle, in degrees w.r.t. x-axis*
float chsiz	*IN : character expansion factor*
int ipos	*IN : centering parameter*

Purpose: PCTEXT draws a text in the current active viewport

Algorithm: Coordinates refer to the current active coorniate system. The routine makes use of the character size as stored in the graphic keywords, which is multiplied by CHSIZ

External references:

<agl.h> <plot_def.h> <midas_def.h> <stdio.h> <string.h>

Module *Version 1.0 (27-Oct-1993)* **Module PCG**

Author: R.M. van Hees ESO-IPG, Garching *Language:* C

Classification: High Level Graphics Interfaces

Source file: $MIDASHOME/$MIDVERS/libsrc/plot/pcg.c

Environment:
- Midas and AGL
- #include <agl.h> /* Prototypes for AGL application programs */
- #include <midas_def.h> /* Prototypes for MIDAS interfaces */

Comments: The module contains routine(s) to manipulate the graphics cursor. The main arguments used by the routines are:

xval input: required initial x position of the locator
output: final locator x position

yval input: required initial y position of the locator
output: final locator y position

key key code (device dependent)

History: [1.0] 23-Sep-1993: Created by R.M. van Hees

Function *PCGCUR*

int PCGCUR(xval, yval, key)

float *xval	*IN/OUT : x value of the locator position*
float *yval	*IN/OUT : y value of the locator position*
int *key	*OUT : key code*

Purpose: retrieve interactively positions from a graphic device

Algorithm: PCGCUR gets a graphics cursor and returns coordinates refering to the current active coordinate system. It returns an error message if the cursor is outside the clipping area.
PCGCUR uses AGL routine AG_VLOC

Returns: zero if the final position is within the clipping area

---

External references:

<agl.h> <midas_def.h> <stdio.h>

Module	*Version 1.0 (27-Oct-1993)*	**Module PCK**

Author: R.M. van Hees ESO-IPG, Garching *Language:* C

Classification: High Level Graphics Interfaces, graphic keywords

Source file: $MIDASHOME/$MIDVERS/libsrc/plot/pck.c

Environment:
- Midas
- #include <midas_def.h> /* Prototypes for MIDAS interfaces */

Comments: The module PCK contains routines to interact with the graphic keywords of MIDAS. The graphic keywords contain values which can be read or written by the user, using a parameter name. For the definition of these name see section 5.2. Using parameter names offers a greater flexibility than using the standard interfaces directly, since the contents of the keywords may change, while the link between the parameter name and its value is saved.
The functions provided are:

- read a parameter from a keyword (PCKRDC, PCKRDI, PCKRDR)
- write a parameter to a keyword (PCKWRC, PCKWRI, PCKWRR)
- set the graphic keywords to default (PCKDEF)

The main arguments used by the routines are:

cpar,ipar,rpar name of a parameter in a keyword, see section 5.2

maxvals maximum number of elements required

actvals actual number of elements accessed, value returned by the routine

cval,ival,rval buffer for the value(s) of the parameter

History: [1.0] 23-Sep-1993: Created by R.M. van Hees

Function *PCKRDC*

void PCKRDC(cpar, maxvals, actvals, cval)

char *cpar	*IN : character parameter to be found*
int maxvals	*IN : no. of elements to get*
int *actvals	*OUT : actual no. of elements returned*
char *cval	*OUT : value of the parameter*

Purpose: reads character values from keyword PLCSTAT, PLCMETA or PLCDATA

Function *PCKRDI*

void PCKRDI(ipar, maxvals, actvals, ival)

char *ipar	*IN : integer parameter to be found*
int maxvals	*IN : no. of elements to get*
int *actvals	*OUT : actual no. of elements returned*
int *ival	*OUT : value of the parameter*

Purpose: reads integer values from keyword PLISTAT

Function *PCKRDR*

void PCKRDR(rpar, maxvals, actvals, rval)

char *rpar	*IN : real parameter to be found*
int maxvals	*IN : no. of elements to get*
int *actvals	*OUT : actual no. of elements returned*
float *rval	*OUT : value of the parameter*

Purpose: reads real values from keyword PLRSTAT or PLRGRAP

Function *PCKWRC*

void PCKWRC(cpar, cval)

char *cpar	*IN : name of the character parameter*
char *cval	*IN : value of the parameter*

Purpose: writes character values in keyword PLCSTAT, PLCMETA or PLCDATA

Function *PCKWRI*

void PCKWRI(ipar, nrval, ival)

char *ipar	*IN : name of the integer parameter*
int nrval	*IN : no. of elements to write, if zero then set IPAR to default*
int *ival	*IN : value for the parameter*

Purpose: writes integer values in keyword PLISTAT

Function *PCKWRR*

void PCKWRR(rpar, nrval, rval)

char *rpar	*IN : name of the real parameter*
int nrval	*IN : no. of elements to write, if zero then set RPAR to default*
float *rval	*IN : value of the parameter*

Purpose: writes real values in keyword PLRSTAT or PLRGRAP

Function *PCKDEF*

void PCKDEF(void)

Purpose: restores the default MIDAS graphic setting

External references:

<midas_def.h> <stdio.h> <string.h>

Module	*Version 1.1 (29-Oct-1993)*	**Module PCO**

Author: R.M. van Hees ESO-IPG, Garching *Language:* C

Classification: High Level Graphics Interfaces

Source file: $MIDASHOME/$MIDVERS/libsrc/plot/pco.c

Environment:
- Midas and AGL
- #include <agl.h> /* Prototypes for AGL application programs */
- #include <osparms.h> /* Symbols used by the OS interfaces */
- #include <plot_def.h> /* Symbols used by the PLT interfaces */
- #include <midas_def.h> /* Prototypes for MIDAS interfaces */

Comments: The module contains the routine for opening and closing a graphics device The main arguments used by the routines are:

devnam Name of the graphics device to open. The devic must be present in the 'agldevs.dat' file containing all legal devices.

plname Name of the meta (plot) file to open or to append

access Integer number to set the access mode to cursor (= −1), over plot (= 1) or new plot (= 0)

plmode Integer number to set the plot mode: (0) no axes and no info, (1) draw axes, labels and some info, (2) draw axes, labels and full info, (−1) use and return the current MIDAS setting.

History: [1.0] 23-Sep-1993: FORTRAN to C (R.M. van Hees) [1.1] 29-Oct-1993: made device and meta-file optional, RvH

Function *PCOPEN*

void PCOPEN(devnam, plname, access, plmode)

char *devnam	*IN : name of device to be activated*
char *plname	*IN : name of meta file*
int access	*IN : access mode*
int *plmode	*IN/OUT : plot mode*

Purpose: sets up graphics device according to MIDAS settings

Algorithm: PCOPEN activates a given device (DEVNAM), or the current (by MIDAS) defined device if no device name is given. A given device has to be present in the "AGLDEVS.dat" file.
PCOPEN does not open, or append to, a meta file if PLNAME equals "NO"ne. If no name, or a file name (with or without ".plt" extention) is given PCOPEN will open or append to a meta file (see access mode). Default for the name of the meta file is the name of the activated device extended with MIDAS session number. A meta file will always get the extension ".plt", any other extension will be replaced.
PCOPEN reads the MIDAS graphics keywords, opens this device (on graphics or image display) according to the MIDAS settings. The coordinate box has to be defined *before* the call to PCOPEN, by using the routine PCKWRR (PTKWRR) to write t

he parameters XWNDL and YWNDL or the parameters XAXES or YAXES. The parameters SCALES and OFFSETS (see PCKWRR or PTKWRR) are always used by PCOPEN(!). They have to be set in the MIDAS procedure calling your program (see procedure “plscoff”) or in your program if you do not want to use the previous setting. Note that you will always overrule them with SET/GRAPHIC.

Function *PCCLOS*

void PCCLOS()

Purpose: close plot file, and terminate graphic operations

External references:

<agl.h>	<osparms.h>	<plot_def.h>	<midas_def.h>
<stdio.h>	<string.h>	<stdlib.h>	<math.h>

Module	*Version 1.0 (27-Oct-1993)*	**Module PTA**

Author: R.M. van Hees ESO-IPG, Garching *Language:* Fortran 77

Classification: High Level Graphics Interfaces

Environment:
- Midas and AGL
- INCLUDE MID_DISK:[MIDAS.'version'.INC]ST_DEF.INC
- INCLUDE MID_DISK:[MIDAS.'version'.INC]ST_DAT.INC

Comments: This module contains the routines to draw a coordinate box. Main arguments used by the routines are:

XMNMX real array holding minimum and maximum value along the X-axis,

YMNMX real array holding minimum and maximum value along the Y-axis,

LABELX character array to be drawn as label along the X-axis

LABELY character array to be drawn as label along the Y-axis

XWCFRM real array with the definition of the X-is: amin, amax, abig ticksteps, asmall tickstep

YWCFRM real array with the definition of the Y-axis: amin, amax, abig ticksteps, asmall tickstep

AGLOPT character array with AGL options, see AGL reference manual.

History: [1.0] 23-Sep-1993: Creation (R.M. van Hees)

Subroutine *PTAXES*

PTAXES(XMNMX, YMNMX, LABELX, LABELY, AGLOPT)

REAL XMNMX(2)	*IN : minimum and maximum value along the X-axis*
REAL YMNMX(2)	*IN : minimum and maximum value along the Y-axis*
CHARACTER*(*) LABELX	*IN : label along X-axis*
CHARACTER*(*) LABELY	*IN : label along Y-axis*
CHARACTER*(*) AGLOPT	*IN : option string, see AGL ref. manual (AG_AXES)*

Purpose: Computes tick marks, and draws axes including tick marks and labels

Algorithm: PTAXES sets the clipping area so that it encloses the box defined by the axes. In practice this means that you can call PTDATA and PTHIST to plot your data within the drawn frame. The character dimension is not affected by PTAXES. PTAXES uses the AGL routine AG_AXES.

Subroutine *PTFRAM*

PTFRAM(XWCFRM, YWCFRM, LABELX, LABELY)

REAL XWCFRM(4)	*IN : definition of the X axis*
REAL YWCFRM(4)	*IN : definition of the Y axis*
CHARACTER*(*) LABELX	*IN : label along the X axis*
CHARACTER*(*) LABELY	*IN : label along the Y axis*

Purpose: Draws a frame including tick marks and labels

Algorithm: PTFRAM draws a standard MIDAS frame including tick marks and labels. Font, character size, line width and text thickness are defined by the current MIDAS setting. PTFRAM uses the AGL routine AG_AXIS.

Module *Version 1.0 (27-Oct-1993)* **Module PTD**

Author: R.M. van Hees ESO-IPG, Garching *Language:* Fortran 77

Classification: High Level Graphics Interfaces

Environment:
- Midas and AGL
- INCLUDE MID_DISK:[MIDAS.'version'.INC]ST_DEF.INC
- INCLUDE MID_DISK:[MIDAS.'version'.INC]ST_DAT.INC

Comments:

This module contains the routines for plotting data points by lines, symbols, and histogram. In addition the module takes care of plotting text strings. The main arguments used by the routines are:

STYPE Integer number defining the symbol type used, see SET/GRAPHIC options,

LTYPE Integer number defining the line type used, see SET/GRAPHIC options,

BINMOD Integer number: (0) bin mode OFF; (1) bin mode ON

FOPT real array,
(1) histogram type selector: (= 0) simple staircase, (= 1) staircase steps joined to X-axis, (> 1) boxes with width starting from 0 (simple line) and increasing with mode value in steps of small character width,
(2) space between the filling lines,
(3) angle of the filling lines w.r.t. X-axis.

IPOS Integer number defining the centering position: (= 1) starts at (xc, yc), (= 2) ends at (xc, yc), or centred on (xc, yc)

History: [1.0] 23-Sep-1993: Creation (R.M. van Hees)

Subroutine *PTDATA*

PTDATA(STYPE,LTYPE,BINMOD,XDATA, YDATA, YOFF, NRDATA)

INTEGER STYPE	*IN : symbol type*
INTEGER LTYPE	*IN : line type*
INTEGER BINMOD	*IN : bin mode*
REAL XDATA(NRDATA)	*IN : X values of the data points*
REAL YDATA(NRDATA)	*IN : Y values of the data points*
REAL YOFF	*IN : offset along the Y-axis*
INTEGER NRDATA	*IN : number of points to be plotted*

Purpose: Plot a set of points connected with lines, or indicated by symbols

Algorithm: The data points are connected with lines if ltype is not equal to zero. Symbols are only plotted when ltype equals zero. A graphic device must be defined. Coordinates refer to the current active coordinate system

Subroutine *PTHIST*

PTHIST(NBINS, CL, RFR, FOPT)

INTEGER NBINS	*IN : number of data points*
REAL CL(NBINS)	*IN : values along the X-axis*
REAL RFR(NBINS)	*IN : number of occurrences per bin*
REAL FOPT(3)	*IN : array with the histogram options*

Purpose: PTHIST draws a histogram with an option to fill

Algorithm: A graphic device must be defined. Coordinates refer to the current active coordinate system. PTHIST uses the AGL routines AG_HIST and AG_FILL

Subroutine *PTTEXT*

PTTEXT(TEXT,XC,YC,ANGLE,CHSIZ,IPOS)

CHARACTER*(*) TEXT	*IN : text to be drawn*
REAL XC	*IN : string position in X*
REAL YC	*IN : string position in Y*
REAL ANGLE	*IN : text angle, in degrees w.r.t. x-axis*
REAL CHSIZ	*IN : character expansion factor*
INTEGER IPOS	*IN : centering parameter*

Purpose: PTTEXT draws a text in the current active viewport

Algorithm: Coordinates refer to the current active coordinate system. The routine makes use of the character size as stored in the graphic keywords, which is multiplied by CHSIZ

Module *Version 1.0 (27-Oct-1993)* **Module PTG**

Author: R.M. van Hees ESO-IPG, Garching *Language:* FORTRAN 77

Classification: High Level Graphics Interfaces

Environment:
- Midas and AGL
- INCLUDE MID_DISK:[MIDAS.'version'.INC]ST_DEF.INC
- INCLUDE MID_DISK:[MIDAS.'version'.INC]ST_DAT.INC

Comments: The module contains routine(s) to manipulate the graphics cursor. The main arguments used by the routines are:

XVAL input: required inital x position of the locator,
output: final locator position

YVAL input: required inital y position of the locator,
output: final locator position

KEY returned key code (device dependent)

History: [1.0] 23-Sep-1993: Created by R.M. van Hees

Subroutine *PTGCUR*

PTGCUR(XVAL,YVAL,KEY,IERR)

REAL XVAL	*IN/OUT : x position of the locator position*
REAL YVAL	*IN/OUT : y position of the locator position*
INTEGER KEY	*OUT : key code*
INTEGER IERR	*OUT : error code*

Purpose: retrieve interactively positions from a graphic device

Algorithm: PTGCUR gets a graphics cursor and returns coordinates referring to the current active coordinate system. It returns an error message if the cursor is outside the clipping area. PTGCUR uses the AGL routine AG_VLOC

Returns: zero if the final position is within the clipping area

Module *Version 1.0 (27-Oct-1993)* **Module PTK**

Author: R.M. van Hees ESO-IPG, Garching *Language:* FORTRAN 77

Classification: High Level Graphics Interfaces, graphic keywords

Environment:
- Midas
- INCLUDE MID_DISK:[MIDAS.'version'.INC]ST_DEF.INC
- INCLUDE MID_DISK:[MIDAS.'version'.INC]ST_DAT.INC

Comments: The module PTK contains routines to interact with the graphic keywords of MIDAS. The graphic keywords contain values which can be read or written by the user, using a parameter name. For the definition of these name see section 5.2. Using parameter names offers a greater flexibility than using the standard interfaces directly, since the contents of the keywords may change, while the link between the parameter name and its value is saved.
The functions provided are:

- read a parameter from a keyword (PTKRDC, PTKRDI, PTKRDR)
- write a parameter to a keyword (PTKWRC, PTKWRI, PTKWRR)
- set the graphic keywords to default (PTKDEF)

The main arguments used by the routines are:

CPAR,IPAR,RPAR name of a parameter in a keyword, see section 5.2

MAXVALS maximum number of elements required

ACTVALS actual number of elements accessed, returned by the routine

CVAL,IVAL,RVAL buffer for the value(s) of the parameter

History: [1.0] 23-Sep-1993: Created by R.M. van Hees

Subroutine *PTKRDC*

PTKRDC(CPAR,MAXVALS,ACTVALS,CVAL)

CHARACTER*(*) CPAR	*IN : character parameter to be found*
INTEGER MAXVALS	*IN : no. of elements to get*
INTEGER ACTVALS	*OUT : actual no. of elements returned*
CHARACTER*(*) CVAL	*OUT : value of the parameter*

Purpose: reads character values from keyword PLCSTAT, PLCMETA or PLCDATA

Subroutine *PTKRDI*

PTKRDI(IPAR, MAXVALS, ACTVALS, IVAL)

CHARACTER*(*) IPAR	*IN : integer parameter to be found*
INTEGER MAXVALS	*IN : no. of elements to get*
INTEGER ACTVALS	*OUT : actual no. of elements returned*
INTEGER IVAL(*)	*OUT : value of the parameter*

Purpose: reads integer values from keyword PLISTAT

Subroutine *PTKRDR*

PTKRDR(RPAR, MAXVALS, ACTVALS, RVAL)

CHARACTER*(*) RPAR	*IN : real parameter to be found*
INTEGER MAXVALS	*IN : no. of elements to get*
INTEGER ACTVALS	*OUT : actual no. of elements returned*
REAL RVAL(*)	*OUT : value of the parameter*

Purpose: reads real values from keyword PLRSTAT or PLRGRAP

Subroutine *PTKWRC*

PTKWRC(CPAR, CVAL)

CHARACTER*(*) CPAR	*IN : name of the character parameter*
CHARACTER*(*) CVAL	*IN : value of the parameter*

Purpose: writes character values in keyword PLCSTAT, PLCMETA or PLCDATA

Subroutine *PTKWRI*

PTKWRI(IPAR, NRVAL, IVAL)

CHARACTER*(*) IPAR	*IN : name of the integer parameter*
INTEGER NRVAL	*IN : no. of elements to write, if zero then set IPAR to default*
INTEGER IVAL(*)	*IN : value for the parameter*

Purpose: writes integer values in keyword PLISTAT

Subroutine *PTKWRR*

PTKWRR(RPAR, NRVAL, RVAL)

CHARACTER*(*) RPAR	*IN : name of the real parameter*
INTEGER NRVAL	*IN : no. of elements to write, if zero then set RPAR to default*
REAL RVAL(*)	*IN : value of the parameter*

Purpose: writes real values in keyword PLRSTAT or PLRGRAP

Subroutine *PTKDEF*

PTKDEF

Purpose: restores the default MIDAS graphic setting

Module *Version 1.1 (29-Oct-1993)* **Module PTO**

Author: R.M. van Hees ESO-IPG, Garching *Language:* FORTRAN 77

Classification: High Level Graphics Interfaces

Environment:
- Midas and AGL
- INCLUDE MID_DISK:[MIDAS.'version'.INC]ST_DEF.INC
- INCLUDE MID_DISK:[MIDAS.'version'.INC]ST_DAT.INC

Comments: The main arguments used by the routines are:

DEVNAM Name of the graphics deivce to open

PLNAME Name of the meta (plot) file to open or to append

ACCESS Integer number to set the access mode to cursor ($= -1$), over plot ($= 1$) or new plot ($= 0$)

PLMODE Integer number to set the plot mode: (0) no axes and no info, (1) draw axes, labels and some info, (2) draw axes, labels and full info, (-1) use and return the current MIDAS setting.

History: [1.0] 23-Sep-1993: FORTRAN to C (R.M. van Hees) [1.1] 29-Oct-1993: made device and meta-file optional, RvH

Subroutine *PTOPEN*

PTOPEN(DEVNAM,PLNAME,ACCESS,PLMODE)

CHARACTER *(*) DEVNAM	*IN : name of device to be activated*
CHARACTER *(*) PLNAME	*IN : name of meta file*
INTEGER ACCESS	*IN : access mode*
INTEGER PLMODE	*IN/OUT : plot mode*

Purpose: sets up graphics device according to MIDAS settings

Algorithm: PTOPEN activates a given device (DEVNAM), or the current (by MIDAS) defined device if no device name is given. A given device has to be present in the "AGLDEVS.dat" file.
PTOPEN does not open, or append to, a meta file if PLNAME equals "NO"ne. If no name, or a file name (with or without ".plt" extention) is given PTOPEN will open or append to a meta file (see access mode). Default for the name of the meta file is the name of the activated device extended with MIDAS session number. A meta file will always get the extension ".plt", any other extension will be replaced.
PTOPEN reads the MIDAS graphics keywords, opens the device (hardcopy, graphics or an image display) according to the keyword settings. The coordinate box has to be defined *before* the call to PCOPEN, by using the routine PCKWRR to write t he parameters XWNDL and YWNDL or the parameters XAXES or YAXES.

Subroutine *PTCLOS*

PTCLOS()

Purpose: close plot file, and terminate graphic operations

Appendix D

Communication Interfaces

Module *Version [1.01] (15-Feb-1996)* **Module XCX**

Author: K. Banse, ESO - Garching *Language:* C

Classification: standard interfaces, communication

Environment:
- UNIX
- #include <midas_def.h>
- #include <xconndef.h>

Comments:

This module contains the routines to open and close a connection to a background Midas as well as routines for getting the status and other info about the background Midas.

The functions provided are:

- Open a connection via ASCII files or sockets (XCZOPN, XCXOPN)
- Start a background Midas and open a connection via ASCII files or sockets (XCZINI, XCXINI)
- Close a connection (XCXCLO)
- Stop a background Midas and close the connection (XCXSTP)
- check status of background Midas (XCXSTA)
- get file descriptor of socket used in communication with background Midas (XCXGFD)

Connections between a program (or a Midas session) and a *background Midas* are done via ASCII files or sockets. Using ASCII files limits the connections to the same host or different hosts sharing disks but can bridge different host systems, like e.g. Unix and Basic based systems.
Connections based on sockets can be established on the same host or over a network but are limited to Unix systems supporting sockets.
Main arguments used by these routines are:

midasunit unit of *background* Midas

cid connection id of *background* Midas

Important: The environment variable MID_WORK must be set and point to the same directory as the MID_WORK variable of the *background* Midas.

Function *XCZOPN*

XCZOPN(midasunit,myunit,direc,cid)

char *midasunit	*IN: unit of background Midas*
char *myunit	*IN: pseudo Midas unit of calling program*
char *direc	*IN: directory for send/receive files*
int *cid	*OUT: connection id of background Midas*

Purpose: Open a connection to a Midas session using files.

Remarks: 'direc' = '\0', indicates we use MID_WORK directory

Returns: return status (0 = o.k.)
= -1; problems with file communication
= -90; invalid input to this interface
= -92; background Midas not running
= -99; variable MID_WORK not set

Function *XCXOPN*

XCXOPN(midasunit,host,cid)

char *midasunit	*IN: unit of background Midas*
char *host	*IN: system name of host where background Midas is running*
int *cid	*OUT: connection id of background Midas*

Purpose: Open a connection to a Midas session using sockets.

Remarks: 'host' = '\0', indicates local host

Returns: return status (0 = o.k.)
= -1; problems with socket communication
= -90; invalid input to this interface
= -92; background Midas not running
= -99; variable MID_WORK not set

Function *XCZINI*

XCZINI(midasunit,myunit,direc,maxtim,cid)

char *midasunit	*IN: unit of background Midas*
char *myunit	*IN: pseudo Midas unit of calling program*
char *direc	*IN: directory for send/receive files*
int maxtim	*IN: max. no. of seconds to wait until background Midas is ready*
int *cid	*OUT: connection id of background Midas*

Purpose: Start a background Midas and open a connection to that session using files.

Remarks: 'direc' = '\0', indicates we use MID_WORK directory

Returns: return status (0 = o.k.)
= -1; problems with file communication
= -2; requested Midas session could not be started
= -3; requested Midas session not ready after 'maxtim' seconds
= -90; invalid input to this interface
= -99; variable MID_WORK not set

Function *XCXINI*

XCXINI(midasunit,host,maxtim,cid)

char *midasunit	*IN: unit of background Midas*
char *host	*IN: system name of host where background Midas is running*
int maxtim	*IN: max. no. of seconds to wait until background Midas is ready*
int *cid	*OUT: connection id of background Midas*

Purpose: Start a background Midas and open a connection to that session usi ng sockets.

Remarks: 'host' = '\0', indicates local host

Returns: return status (0 = o.k.)
= -1; problems with socket communication
= -2; requested Midas session could not be started
= -3; requested Midas session not ready after 'maxtim' seconds
= -90; invalid input to this interface
= -99; variable MID_WORK not set

Function *XCXCLO*

XCXCLO(cid)

int cid *IN: connection id of background Midas*

Purpose: Close connection to background Midas.

Returns: return status (0 = o.k.)
= -1; problems with socket/file communication
= -90; invalid input to this interface

Function *XCXSTP*

XCXSTP(cid)

int cid *IN: connection id of background Midas*

Purpose: Stop background Midas and close connection to it.

Returns: return status (0 = o.k.)
= -1; problems with socket/file communication
= -90; invalid input to this interface

Function *XCXSTA*

XCXSTA(cid)

int cid *IN: connection id of background Midas*

Purpose: Check status of background Midas.

Returns: return status:
= 0; background Midas idle
= 1; background Midas busy
= -1; problems with socket/file communication
= -90; invalid input to this interface

Function *XCXGFD*

XCXGFD(cid,fid)

int cid	*IN: connection id of background Midas*
int *fid	*OUT: file descr. of socket used for connection*

Purpose: Get file descriptor of socket used in communication with background Midas.

Returns: return status (0 = o.k.)
= -1; problems with socket communication
= -90; invalid input to this interface

Module *Version [1.00] (15-Dec-1994)* **Module XCC**

Author: K. Banse, ESO - Garching *Language:* C

Classification: standard interfaces, communication

Environment:
- UNIX
- #include <midas_def.h>
- #include <xconndef.h>

Comments:

This module contains the routines to send a command to background Midas and to wait for return from a command which was sent to background Midas before.

The functions provided are:

- Send a command to background Midas (XCCSND)
- Wait until a command which has been sent to background Midas finishes (XCCWAI)

ASCII files or sockets are used for the communication and the transfer of data, depending on how the connection to the background Midas has been opened.

Main arguments used by the routines are:

cid connection id of background Midas

retstat returns Midas status from command execution in the background Midas if communication o.k., else the communication error code itself

---

Function *XCCSND*

XCCSND(cid,comline,secs,retstat)

int cid	*IN: connection id of background Midas*
char *comline	*IN: command to be executed by background Midas*
int secs	*IN: max. no. of seconds to wait*
int retstat[2]	*OUT: Midas status from Background Midas (if we wait)*

Purpose: Send a command to background Midas with or without waiting for its termination.

Remarks: 'secs' = 0, indicates no wait
'secs'= -1; indicates that we wait forever

Returns: return status (0 = o.k.)
= 2; time out, 'secs' seconds have passed without return
= -1; problems with socket/file communication
= -90; invalid input to this interface

Function *XCCWAI*

XCCWAI(cid,retstat)

int cid	*IN: connection id of background Midas*
int retstat[2]	*OUT: return status from Background Midas*

Purpose: Wait until a command is executed which has been sent to background Midas (without wait) before.

Returns: return status (0 = o.k., retstat holds status from background Midas)
= 2; time out, 'secs' seconds have passed without return
= -1; problems with socket/file communication
= -90; invalid input to this interface

Module	*Version [1.00] (15-Jan-1995)*	**Module XCK**

Author: K. Banse, ESO - Garching *Language:* C

Classification: standard interfaces, communication

Environment:
- UNIX
- #include <midas_def.h>
- #include <xconndef.h>

Comments:

This module contains the routines to read and write keywords of a background Midas.

The following keyword access routines can ONLY be used if the connection to the background Midas has been established via sockets, i.e. via XCXOPN.

The functions provided are:

- Read contents of keywords of background Midas (XCKRDC, XCKRDD, XCKRDI, XCKRDR)
- Write contents of keywords of background Midas (XCKWRC, XCKWRD, XCKWRI,XCKWRR)

Main arguments used by the routines are:

cid connection id of background Midas

key name of keyword, max. 8 chars. long and must begin with an alphabetic char.

felem first element to be accessed, the very first element is indicated by 1 (not by 0 as in C-arrays) for compatibility with the relevant SCKRDx and SCKWRx interfaces

maxvals maximum number of elements required

actvals actual number of elements accessed, value returned by the routines

unit unit of keyword, currently not implemented - just used as a placeholder for now

null no. of null values in keyword, currently not implemented - just used as a placeholder for now

kstat status from the corresponding SCKRDx or SCKWRx routine executed in the background Midas

Function *XCKRDC*

XCKRDC(cid,key,noelm,felem,maxvals,actvals,values,unit,null,kstat)

int cid	*IN: connection id of background Midas*
char *key	*IN: keyword name (max. 8 chars.)*
int noelm	*IN: no. of bytes per data value*
int felem	*IN: first data item to be read*
int maxvals	*IN: no. of elements to get*
int *actvals	*OUT: actual no. of elements returned*
char *values	*OUT: buffer for data values*
int *unit	*OUT: address of unit-pointer*
int *null	*OUT: no. of null values in keyword*
int *kstat	*OUT: status from SCKRDC in background Midas*

Purpose: Read data from character keyword of background Midas.

Returns: return status (0 = o.k.)
= 1; background Midas busy
= -1; problems with socket communication
= -90; invalid input to this interface
= -91; only non-socket communication established

Algorithm: 'actvals' will be set to the actual no. of values returned.

Remarks: 'noelm' > 1 enables you to use character arrays, where each data value is in effect a char. string of 'noelm' bytes

---

Function *XCKRDD*

XCKRDD(cid,key,felem,maxvals,actvals,values,unit,null,kstat)

int cid	*IN: connection id of background Midas*
char *key	*IN: keyword name (max. 8 chars.)*
int felem	*IN: first data item to be read*
int maxvals	*IN: no. of elements to get*
int *actvals	*OUT: actual no. of elements returned*
double *values	*OUT: buffer for data values*
int *unit	*OUT: address of unit-pointer*
int *null	*OUT: no. of null values in keyword*
int *kstat	*OUT: status from SCKRDD in background Midas*

Purpose: Read data from double precision keyword of background Midas.

Returns: return status (0 = o.k.)
= 1; background Midas busy
= -1; problems with socket communication
= -90; invalid input to this interface
= -91; only non-socket communication established

Algorithm: 'actvals' will be set to the actual no. of values returned.

Function *XCKRDI*

XCKRDI(cid,key,felem,maxvals,actvals,values,unit,null,kstat)

int cid	*IN: connection id of background Midas*
char *key	*IN: keyword name (max. 8 chars.)*
int felem	*IN: first data item to be read*
int maxvals	*IN: no. of elements to get*
int *actvals	*OUT: actual no. of elements returned*
int *values	*OUT: buffer for data values*
int *unit	*OUT: address of unit-pointer*
int *null	*OUT: no. of null values in keyword*
int *kstat	*OUT: status from SCKRDI in background Midas*

Purpose: Read data from integer keyword of background Midas.

Returns: return status (0 = o.k.)
= 1; background Midas busy
= -1; problems with socket communication
= -90; invalid input to this interface
= -91; only non-socket communication established

Algorithm: 'actvals' will be set to the actual no. of values returned.

Function *XCKRDR*

XCKRDR(cid,key,felem,maxvals,actvals,values,unit,null,kstat)

int cid	*IN: connection id of background Midas*
char *key	*IN: keyword name (max. 8 chars.)*
int felem	*IN: first data item to be read*
int maxvals	*IN: no. of elements to get*
int *actvals	*OUT: actual no. of elements returned*
float *values	*OUT: buffer for data values*
int *unit	*OUT: address of unit-pointer*
int *null	*OUT: no. of null values in keyword*
int *kstat	*OUT: status from SCKRDR in background Midas*

Purpose: Read data from real keyword of background Midas.

Returns: return status (0 = o.k.)
= 1; background Midas busy
= -1; problems with socket communication
= -90; invalid input to this interface
= -91; only non-socket communication established

Algorithm: 'actvals' will be set to the actual no. of values returned.

Function *XCKWRC*

XCKWRC(cid,key,noelm,values,felem,maxvals,unit,kstat)

int cid	*IN: connection id of background Midas*
char *key	*IN: keyword name (max. 8 chars.)*
int noelm	*IN: no. of bytes per data value*
char *values	*IN: buffer with data values*
int felem	*IN: first data item to be written*
int maxvals	*IN: no. of elements to write*
int *unit	*IN: unit-pointer*
int *kstat	*OUT: status from SCKWRC in background Midas*

Purpose: Write data into character keyword of background Midas.

Returns: return status (0 = o.k.)
= 1; background Midas busy
= -1; problems with socket communication
= -90; invalid input to this interface
= -91; only non-socket communication established

Remarks: 'noelm' > 1 enables you to use character arrays, where each data value is in effect a char. string of 'noelm' bytes

---

Function *XCKWRD*

XCKWRD(cid,key,values,felem,maxvals,unit,kstat)

int cid	*IN: connection id of background Midas*
char *key	*IN: keyword name (max. 8 chars.)*
double *values	*IN: buffer with data values*
int felem	*IN: first data item to be written*
int maxvals	*IN: no. of elements to write*
int *unit	*IN: unit-pointer*
int *kstat	*OUT: status from SCKWRD in background Midas*

Purpose: Write data into double precision keyword of background Midas.

Returns: return status (0 = o.k.)
= 1; background Midas busy
= -1; problems with socket communication
= -90; invalid input to this interface
= -91; only non-socket communication established

Function *XCKWRI*

XCKWRI(cid,key,values,felem,maxvals,unit,kstat)

int cid	*IN: connection id of background Midas*
char *key	*IN: keyword name (max. 8 chars.)*
int *values	*IN: buffer with data values*
int felem	*IN: first data item to be written*
int maxvals	*IN: no. of elements to write*
int *unit	*IN: unit-pointer*
int *kstat	*OUT: status from SCKWRI in background Midas*

Purpose: Write data into integer keyword of background Midas.

Returns: return status (0 = o.k.)
= 1; background Midas busy
= -1; problems with socket communication
= -90; invalid input to this interface
= -91; only non-socket communication established

Function *XCKWRR*

XCKWRR(cid,key,values,felem,maxvals,unit,kstat)

int cid	*IN: connection id of background Midas*
char *key	*IN: name of keyword*
float *values	*IN: buffer with data values*
int felem	*IN: first data item to be written*
int maxvals	*IN: no. of elements to write*
int *unit	*IN: unit-pointer*
int *kstat	*OUT: status from SCKWRR in background Midas*

Purpose: Write data into real keyword of background Midas.

Returns: return status (0 = o.k.)
= 1; background Midas busy
= -1; problems with socket communication
= -90; invalid input to this interface
= -91; only non-socket communication established

Appendix E

Standard Descriptors and Column Names

E.1 Descriptors

E.1.1 Bulk Data Frames

The Descriptors `NAXIS, NPIX, START, STEP, CUNIT, IDENT` and `LHCUTS` **must** be present, in order to define the data as an image frame.

`[IDENT]` Character Descriptor of up to 72 characters, to contain the identifier of the frame or object.

`[NAXIS]` Integer Descriptor containing the number of dimensions (or axes) of the data.

`[NPIX]` Integer Descriptor of `NAXIS` elements containing the size (i.e. number of pixels) of each axis.

`[START]` Double precision Descriptor of `NAXIS` elements containing the world coordinate value of the first pixel of each axis. **Note:** the value refers to the center of the pixel (*e.g.* sensitivity weighted mean).

`[STEP]` Double precision Descriptor of `NAXIS` elements containing the distance or step between pixels on each axis expressed in units of the world coordinate system corresponding to that axis.

`[CUNIT]` Character Descriptor of `16*(NAXIS+1)` characters containing the unit of the data itself and the units of the world coordinates of each axis. Here is specified, in what units the world coordinates and data are given. It is recommended to use units defined in the IAU Style Manual.

[LHCUTS] Real Descriptor of 4 elements containing the user cuts and physical minimum and maximum of the data frame. **Note:** some applications may use elements 5 and 6 for storage of special cuts values.

The descriptors described above are also called the *standard descriptors* of an image (cf. the SCDCOP and STDCOP interfaces).

A number of other descriptor names are reserved and may be present. These descriptors may **not** be used for any other purpose than the one defined here. They are:

[REFPIX] Double precision Descriptor of NAXIS elements containing location of the frame reference point in pixel coordinates where the value 1.0 corresponds to the center of the first pixel. This Descriptor is created by the FITS reader.

[ROTA] Double precision Descriptor of NAXIS elements describes the rotation of the coordinate system in degrees and corresponds to the FITS 'CROTAn' keyword. This descriptor is disregarded if the CTMATRIX descriptor exits. This Descriptor is created by the FITS reader.

[CTYPE] Character Descriptor of 16*NAXIS characters defines the type of coordinate system used for the axes and corresponds to the FITS 'CTYPEn' keyword. This Descriptor may be created by the FITS reader.

[CTMATRIX] Double precision Descriptor of NAXIS*NAXIS elements defines the coordinate transformation matrix for the axes. The matrix elements are stored column by column. This Descriptor may be created by the FITS reader.

[O_TIME] Double precision Descriptor with information on time, duration etc. of an exposure.

1. epoch at start of exposure (years) (**note:** the epoch 1992.0 corresponds to 1992-01-01 00:00 UT),
2. month of the epoch (January:1) (**note:** must be zero if element 1 is a fraction of year),
3. day of the epoch (**note:** must be zero if element 1 is a fraction of year),
4. Modified Julian Date (MJD) at start of exposure (days),
5. Universal Time (UT) at start of exposure (hours),
6. Local mean Sidereal Time (LST) at start of exposure (hours),
7. exposure time in seconds.

This Descriptor may be created by the FITS reader.

[O_POS] Double precision Descriptor of 3 elements with information on the position of the object observed.

1. right ascension in degrees!,

2. declination in degrees, and
3. equinox of coordinates.

This Descriptor may be created by the FITS reader.

[RADECSYS] Character Descriptor of 16 characters defining the type of celestial coordinate system used *e.g.* FK4 or FK5. This Descriptor may be created by the FITS reader.

[O_AIRM] Single precision Descriptor with mean airmass of exposure. This Descriptor may be created by the FITS reader.

[OBSERVER] Character Descriptor with the name of the observer. This Descriptor may be created by the FITS reader.

[TELESCOP] Character Descriptor with the name of the telescope used for acquisition. This Descriptor may be created by the FITS reader.

[INSTRUME] Character Descriptor with the name of the instrument used for acquisition. This Descriptor may be created by the FITS reader.

[FILENAME] Character Descriptor with the original file name. This Descriptor may be created by the FITS reader.

[HISTORY] Character Descriptor containing the history of the data. This Descriptor may be created by the FITS reader.

[HISTORY_UPDA] Integer Descriptor containing the update flag for descriptor HISTORY of the same frame. If set to 1, descriptor HISTORY is updated. If set to 0, descriptor HISTORY is not updated.

[COMMENT] Character Descriptor with comments relating to the data. This Descriptor may be created by the FITS reader.

[ESO_LOG] Character Descriptor with the observing log. This Descriptor may be created by the FITS reader.

[HIERARCH] Character Descriptor containing hierarchial keyword information for the frame. This Descriptor may be created by the FITS reader.

[NULLVALUE] Integer Descriptor of 8 elements defining the bit-patterns of the NULL values used. Its value **must not** be changed. This Descriptor may be created by the FITS reader.

[DISPLAY_DATA] Integer descriptor of 9 elements holding information about the way the frame was loaded into the display. First 2 elements contain scaling values, negative values indicate expansion, positive values indicate shrinking of the image (note, that for historic reasons the sign is the opposite of what you use in the LOAD/IMAGE command). The next two values are the center pixels in x,y. DISPLAY_DATA(5) =

1 or 0 depending upon if the frame was averaged or not when loaded with scaling factors ¡ 0. The values of DISPLAY_DATA(6,7,8,9) are the frame and screen pixels (x,y) for a given fixpoint.

Descriptors starting with underscore '_' are used for special storage of acquisition data. Such descriptors should not be created by users. ESO has defined a number of hierarchical keywords which are used by the Archive for storage of information relating to the data acquisition *e.g.* instrument and telescope parameters. This information is stored in descriptors starting with '_E' followed by characters defining the logical hierarchical level. The level field is separated from the lowest level with a '_'

E.1.2 Tables

Table files use a number of standard descriptors to define their content. These descriptors **must not** be modified by the user. They are defined as follows:

`[TBLCONTR]` Integer Descriptor of 10 elements with the following control information

1	- no. of allocated columns (no. of words per row)
2	- no. of allocated rows
3	- no. of defined columns (NDC)
4	- no. of defined rows
5	- reference column
6	- sorted column
7	- physical table format on disk
8	- table version
9	- items reserved for column descriptors
10	- number of selected rows

`[TLABLxxx]` Character Descriptor with 41 bytes containing for each column the information : Label (1 to 16), Unit (17 to 32) and Format (33 to 41). `'xxx'` is the column number in the range `001, ..., NDC`

`[TBLENGTH]` Integer Descriptor with NDC elements which defining for each column its type and its number of items

`[TBLOFFST]` Integer Descriptor with NDC elements defining the offset in bytes of each column from the beginning of the table.

`[TSELTABL]` Character Descriptor of 64 characters containing the selection information ("-") if all table selected)

The descriptors TBLENGTH, TBLOFFST, TBLCONTR and TSELTABL are also called the *standard descriptors* of a table (cf. the SCDCOP and STDCOP interfaces).

In addition these required standard table descriptors, a table may contain a number of other descriptors. The following reserved descriptor names may only be used with the meaning defined here:

[IDENT] Character Descriptor of up to 72 characters, to contain the identifier of the table.

[FILENAME] Character Descriptor with the original file name. This Descriptor may be created by the FITS reader.

[NULLVALUE] Integer Descriptor of 8 elements defining the bit-patterns of the NULL values used. Its value **must not** be changed. This Descriptor may be created by the FITS reader.

[AUTHOR] Character Descriptor identifying the person who compiled the table information. This Descriptor may be created by the FITS reader.

[REFEREC] Character Descriptor containing a biliographic reference to where the data have been published. This Descriptor may be created by the FITS reader.

E.2 Label Names of Columns

The usage of standard label names for table columns makes the exchange of information in tables much easier. Thus, it is recommended to use the label names in the table below. A more complete list is being prepared.

Currently some applications do not use these recommended column labels. Conformity to these label names will be made mandatory for future MIDAS releases.

Label	Format	Unit	Content
:IDENT	C32	-	Identifier of object
:WAVE	R8	nm	Wavelength
:FREQ	R8	Hz	Frequency
:FLUX	R4	-	Flux (undefined units)
:FLUX_W	R4	-	Flux (wavelength bins)
:FLUX_F	R4	-	Flux (frequency bins)
:BIN_W	R4	nm	Wavelength bin size
:BIN_F	R4	Hz	Frequency bin size
:ION	C16	-	Name of ion
:ORDER	I4	-	Order no. of echelle spectrum
:MAG	R4	-	Flux in magnitude units
:MAG_x	R4	-	Flux for color 'x' in magnitude units
:SKY	R4	-	Mean sky value
:PEAK	R4	-	Fitted peak value of object (-background)
:TRANS	R4	-	Transmission of filters
:RESP	R4	-	Relative Response of detectors
:RA	R8	degree	Right Ascension
:DEC	R8	degree	Declination
:EPOCH	R4	year	Epoch
:JD	R8	day	Julian Date
:MJD	R8	day	Modified Julian Date
:PM_RA	R4	"/year	Proper motion in Right Ascension
:PM_DEC	R4	"/year	Proper motion in Declination
:ST	R8	hour	Local Sidereal time
:UT	R8	hour	Universal time
:TIME	R8	hour	Time (undefined units)
:RV	R4	km/s	Radial velocity
:Z_RED	R4	-	Redshift expressed as z

Table E.1: Recommended label name for table columns.

Label	Format	Unit	Content
:XCEN	R8	-	Center coordinate of object in X
:YCEN	R8	-	Center coordinate of object in Y
:XERR	R4	-	R.M.S error in X coordinate
:YERR	R4	-	R.M.S error in X coordinate
:XFWHM	R4	-	Full Width Half Maximum of object in X
:YFWHM	R4	-	Full Width Half Maximum of object in Y
:X	R4	-	World coordinate of cursor in X
:Y	R4	-	World coordinate of cursor in Y
:VALUE	R4	-	Pixel value at cursor position
:VALUE1	R4	-	Pixel value at cursor #1 position
:VALUE2	R4	-	Pixel value at cursor #2 position
:XSTART	R4	-	Start coordiate of area in X
:YSTART	R4	-	Start coordiate of area in Y
:XEND	R4	-	End coordiate of area in X
:YEND	R4	-	End coordiate of area in Y

Table E.2: Recommended label name for table columns (cont.).

Appendix F

Application Program Examples

F.1 Examples Using Image Data

F.1.1 An Application in FORTRAN

The following source of the program `SMOOTH` serves as an example of an application program ready to be run inside the MIDAS environment. This program smooths an image with a box filter.
This program is tied into MIDAS via the following procedure, `smooth.prg`:

```
!+++++++++++++++++++++++++++++++++++++++++++++++++++++
!
!.COPYRIGHT   (C) 1990 European Southern Observatory
!.IDENT       smooth.prg
!.AUTHOR      Klaus Banse,  ESO - Garching
!.KEYWORDS    Smoothing, Filtering
!.PURPOSE     smooth an image bu box-car filtering
!.VERSION     1.0    Creation    900129
!
!-----------------------------------------------------
!
DEFINE/PARAM P1 ? IMA "Enter input frame: "
DEFINE/PARAM P2 ? IMA "Enter output frame: "
DEFINE/PARAM P3 1.,0. N "Enter radius, threshold: "
DEFINE/PARAM P4 Y C "Enter center_flag, Y(es) or N(o): "
WRITE/KEYWORD IN_A {P1}
WRITE/KEYWORD OUT_A {P2}
WRITE/KEYWORD INPUTR/R/1/2 {P3}
WRITE/KEYWORD HISTORY Smooth
WRITE/KEYWORD DEFAULT {P4}
```

```
RUN smooth.exe
```

To run the application execute the procedure via

```
Midas 001> @@ smooth inframe outframe rad,thresh centr_flag
```

The program is written in standard Fortran 77 code with the exceptions which are taken care of by the ESO provided preprocessor. This preprocessor is not needed on a VAX/VMS machine (see chapter 7 for details).

```
C @(#)smooth.for        6.3 (ESO-IPG) 1/17/94 17:53:36
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C.IDENTIFICATION
C  program SMOOTH                        version 5.00       900129
C  K. Banse                              ESO - Garching
C
C.KEYWORDS
C  smoothing, average
C
C.PURPOSE
C  smooth an image by averaging in the neighborhood
C
C.ALGORITHM
C  get the names of input + output frames from Keywords IN_A + IN_B,
C  get the radius + threshold for the averaging algorithm from the
C  Keyword INPUTI and calculate new points via:
C
C  B(n) = SUM A(j), j in neighbourhood of given radius around n, if
C  SUM - A(n) > threshold*A(n), else B(n) = A(n)
C
C.INPUT/OUTPUT
C  the following Keywords are used:
C
C  IN_A/C/1/60      input frame
C  OUT_A/C/1/60     output frame
C  DEFAULT/C/1/1    default = Y, include center pixel in calculation
C                   of SUM
C  INPUTR/R/1/2     radius ( >0 ), threshold ( in [0,1] ) for
C                   averaging algorithm
C
C--------------------------------------------------------------------
C
      PROGRAM SMOOTH
C
      IMPLICIT NONE
C
      INTEGER       NAXISA,NPIXA(2),IAV,STAT,IRAD
      INTEGER       IMNOA,IMNOC,PNTRA,PNTRC
      INTEGER       KNULL,KUNIT(1)                !KNULL, KUNIT currently only
      INTEGER       MADRID(1)                     !placeholders...!
C
```

```
      CHARACTER*60  FRAMEA,FRAMEC
      CHARACTER     CUNITA*64,IDENTA*72,DEFAUL*1
C
      DOUBLE PRECISION    STEPA(2),STARTA(2)      !descr. START + STEP are
C                                                 !now double precision
C
      REAL          THRESH,INPUTR(2),CUTS(4)
C
C  the following INCLUDE statements may look strange to a Unix user,
C  but the ESO preprocessor takes care of the syntax...
C
      INCLUDE 'MID_INCLUDE:ST_DEF.INC'            !to get the definitions
C                                                 !for the ST interfaces...
C
C  very important COMMON block VMR for the pointer emulation in Fortran 77 !!!
      COMMON  /VMR/ MADRID
C
      INCLUDE 'MID_INCLUDE:ST_DAT.INC'            !to get the data values
C                                                 !for the definitions above...
C
      DATA    IDENTA /' '/                 !initialization of IDENTA, CUNITA
      DATA    CUNITA /' '/                 !is necessary for STIGET ...
C
C  set up MIDAS environment + enable automatic error abort
C
      CALL STSPRO('SMOOTH')
C
C  get name of input frame and map it as an image with pixels
C  in floating point format
C  use the definitions from the include file ST_DEF.INC (e.g. D_R4_FORMAT)
C
      CALL STKRDC('IN_A',1,1,60,IAV,FRAMEA,KUNIT,KNULL,STAT)
      CALL STIGET(FRAMEA,D_R4_FORMAT,F_I_MODE,F_IMA_TYPE,
     +            2,NAXISA,NPIXA,STARTA,STEPA,
     +            IDENTA,CUNITA,PNTRA,IMNOA,STAT)
C
C  a call to STETER will abort the program ...
C
      IF (NAXISA.NE.2)
     +   CALL STETER(1,'input frame must have 2 dimensions...')
C
C  get name of output frame and create and map it
C  Note, that this does not fill in any data values
C
```

```
      CALL STKRDC('OUT_A',1,1,60,IAV,FRAMEC,KUNIT,KNULL,STAT)
      CALL STIPUT(FRAMEC,D_R4_FORMAT,F_O_MODE,F_IMA_TYPE,
     +            NAXISA,NPIXA,STARTA,STEPA,
     +            IDENTA,CUNITA,PNTRC,IMNOC,STAT)
C
C  since we use STIGET and STIPUT we cannot have calls of STFGET or STFPUT...
C
C  get flag, if center pixel should be included
C  get the radius + threshold for the averaging algorithm
C
      CALL STKRDC('DEFAULT',1,1,1,IAV,DEFAUL,KUNIT,KNULL,STAT)
      CALL STKRDR('INPUTR',1,2,IAV,INPUTR,KUNIT,KNULL,STAT)
      IRAD = NINT(INPUTR(1))
      THRESH = INPUTR(2)
C
C  Now call the subroutine to do the averaging.
C  Note, that the address of the input and output array are passed to the
C  subroutine using the array MADRID
C
      CALL AVERAG(MADRID(PNTRA),MADRID(PNTRC),NPIXA(1),NPIXA(2),
     +            IRAD,THRESH,DEFAUL)
C
C  update descr. HISTORY with contents of keyword HISTORY + P1, P2, ...
C  and copy all non-standard descriptors from input to output frame
C
      IDENTA(1:) = ' '
      CALL DSCUPT(IMNOA,IMNOC,IDENTA(1:1),STAT)
C
C  release files, update keywords and exit
C  STSEPI closes all open images, only then are the data values written
C  into the result image file
C
      CALL STSEPI
C
      END



      SUBROUTINE AVERAG(A,B,NX,NY,IRAD,THRESH,ALL)
C
C  smooth 2-dim image by taking the average of surrounding points,
C  if it differs more than THRESH from original value
C
```

```
      IMPLICIT NONE
C
      INTEGER     NX,NY,IRAD
      INTEGER     N,NN,NL,NRL,NP,OFF,OFFA,DIFX,INDX
C
      REAL        A(1),B(1),THRESH,TEST,COUNT,W
C
      CHARACTER   ALL*1
C
C  initialize COUNT
C  to number of pixels in neighbourhood of center
      COUNT = FLOAT(IRAD*(IRAD+1)*4) + 1.
C
C  first IRAD lines are not treated
C
      DO 1000 N=1,IRAD*NX
         B(N) = A(N)
1000  CONTINUE
C
C  main loop over inner lines
C
      DO 5000 NL=IRAD+1,NY-IRAD
         OFF = (NL-1)*NX
C
C  leave first IRAD values of each line as they are
C
         DO 2000 N=OFF+1,OFF+IRAD
            B(N) = A(N)
2000     CONTINUE
C
C  average in the inner part
C
         DO 4000 NP=IRAD+1,NX-IRAD
            INDX = OFF + NP              !pixel to work on
            W = A(INDX)                  !save original pixel value
            TEST = 0.0                   !clear sum
C                                        !sum over IRAD lines above
            DO 3000 NRL=1,IRAD              !and below current line
               DIFX = NRL * NX
               OFFA = INDX - DIFX - IRAD
               DO 2500 NN=OFFA,OFFA+IRAD*2
                  TEST = TEST + A(NN)
2500           CONTINUE
C
```

```
              OFFA = OFFA + (2*DIFX)
              DO 2800 NN=OFFA,OFFA+IRAD*2
                 TEST = TEST + A(NN)
2800          CONTINUE
3000        CONTINUE
C
            DO 3100 NN=INDX-IRAD,INDX+IRAD    !sum over current line
              TEST = TEST + A(NN)
3100        CONTINUE
C
C  compare average of neighbourhood with pixel in question
C
            IF (ALL.NE.'Y') THEN
              TEST = (TEST-W) / (COUNT-1.)
            ELSE
              TEST = TEST/COUNT
            ENDIF
C
            IF (ABS (TEST-W).GT.ABS (THRESH*W)) THEN
              B(INDX) = TEST
            ELSE
              B(INDX) = W
            ENDIF
4000      CONTINUE
C
C  leave last IRAD values of each line as they are
C
          DO 4100 NN=OFF+NX-IRAD+1,OFF+NX
            B(NN) = A(NN)
4100      CONTINUE
C
5000   CONTINUE
C
C  end of main loop - last lines are not treated
C
       DO 6000 N=(NY-IRAD)*NX+1,NX*NY
          B(N) = A(N)
6000   CONTINUE
C
C  that's it folks
C
       RETURN
       END
```

F.1.2 An Application in C

The following program `copymi` is written in C and copies an image from one data format to another. It is the application which implements the MIDAS `COPY/II` command. In this example also the usage of the SCFGET and SCFPUT routines instead of SCIGET and SCIPUT is demonstrated.
This program is tied into MIDAS via the following procedure, `copy.prg`:

```
!+++++++++++++++++++++++++++++++++++++++++++++++++++++++
!
!.COPYRIGHT   (C) 1989 European Southern Observatory
!.IDENT       copy.prg
!.AUTHOR      Klaus Banse,  ESO - Garching
!.KEYWORDS    data conversion, copying
!.PURPOSE     copy images (and convert data formats)
!.VERSION     1.0    Creation    890414
!
!-------------------------------------------------------
!
DEFINE/PARAM P1 ? IMA "Enter source frame: "
DEFINE/PARAM P2 ? IMA "Enter destination frame: "
DEFINE/PARAM P3 R4 C "Enter destination format: "
DEFINE/PARAM P5 20000 N "Enter mapping buffer size: "
WRITE/KEYWORD IN_A {P1}
WRITE/KEYWORD OUT_A {P2}
WRITE/KEYWORD INPUTI/I/20/1 {P5}
RUN MID_EXE:copymi
```

To run the application execute the procedure via
`Midas 001> @@ copy inframe outframe I2`

```
/* @(#)copymi.c 6.4 (ESO-IPG) 10/19/93 16:56:29 */
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.COPYRIGHT    (c)  1988   European Southern Observatory
.IDENT        copymi.c
.LANGUAGE     C
.AUTHOR       K. Banse   ESO/IPG
.KEYWORDS     Format Conversion
.PURPOSE      Read data frame in input format, convert it to output format
              and store in output frame
.COMMENT      Formats supported are the MIDAS formats:
              I1, I2, R4
.VERSION      1.00  890414:   Creation (adapted from COPYMI.FOR)  KB
.VERSION      1.10  890512:   use SCFGET, SCFPUT to make it faster
.VERSION      1.20  890628:   use SCKGETC and add "delete option"
.VERSION      1.30  910228:   copy also LHCUTS
.VERSION      1.40  910417:   use SCDCOP to copy all descriptors in one go
.VERSION      1.50  911004:   take care of over/underflow
.VERSION      2.00  930929:   long int -> int, unit -> *int
--------------------------------------------------------------------*/

#include   <midas_def.h>

main()

{

char      infile[64], outfile[64], cbuf[12];
char      *pntra, *pntrb;
char      dely[4];

int  unit;
int  maxdim, naxis, npix[6], imnoa, imnob, imnow1, imnow2;
int  felm, nochunk, mapsize, size;
int  info[5], null, nval;

int  i, n, stat, fmtin, fmtout;
int  uflow, oflow;

float   cuts[4];

double  start[6], step[6];
```

```
SCSPRO("copymi");

stat = SCKGETC("IN_A",1,60,&nval,infile);   /* get input frame   */
stat = SCKGETC("OUT_A",1,60,&nval,outfile);   /* get output frame  */
stat = SCKGETC("P3",1,10,&nval,cbuf);  /* get output format */
stat = SCKGETC("P4",1,4,&nval,dely);   /* get delete flag   */

stat = SCKRDI("INPUTI",20,1,&nval,&mapsize,&unit,&null);  /* mapping size */

stat = SCFINF(infile,2,info); /* get source format  */
fmtin =  info[1];
fmtout = check_fmt(cbuf);
if (fmtout < 0) SCETER(2,"invalid data format...");

stat = SCFOPN(infile,info[1],0,F_IMA_TYPE,&imnoa);
stat = SCDRDI(imnoa,"NAXIS",1,1,&nval,&naxis,&unit,&null);
maxdim = 6; /* current limit  */
if (naxis > maxdim) SCETER(3,"Too many dimensions...");

stat = SCDRDI(imnoa,"NPIX",1,naxis,&nval,npix,&unit,&null);
stat = SCDRDD(imnoa,"START",1,naxis,&nval,start,&unit,&null);
stat = SCDRDD(imnoa,"STEP",1,naxis,&nval,step,&unit,&null);
stat = SCDRDR(imnoa,"LHCUTS",1,4,&nval,cuts,&unit,&null);

size = 1; /*  get total size of frame  */
for (n=0; n< naxis; n++)
   size *= npix[n];

stat = SCFCRE(outfile,fmtout,F_O_MODE,F_IMA_TYPE,size,&imnob);

stat = SCDWRI(imnob,"NAXIS",&naxis,1,1,&unit);
stat = SCDWRI(imnob,"NPIX",npix,1,naxis,&unit);
stat = SCDWRD(imnob,"START",start,1,naxis,&unit);
stat = SCDWRD(imnob,"STEP",step,1,naxis,&unit);

uflow = 0; oflow = 0;


/*  calculate no. of chunks needed  */

if (size <= mapsize)
   {
   mapsize = size;
   nochunk = 1;
```

```
   }

else
   nochunk = ((size-1) / mapsize) + 1;


/*  allocate virtual memory for in/out frame buffers  */

stat = SCFCRE("work1",fmtin,F_X_MODE,F_IMA_TYPE,mapsize,&imnow1);
stat = SCFMAP(imnow1,F_X_MODE,1,mapsize,&nval,&pntra);

if (fmtin != fmtout)
   {
   stat = SCFCRE("work2",fmtout,F_X_MODE,F_IMA_TYPE,mapsize,&imnow2);
   stat = SCFMAP(imnow2,F_X_MODE,1,mapsize,&nval,&pntrb);
   }
else
   pntrb = pntra;


/*  now we do the conversion   */

felm = 1;
for (i=0; i<nochunk; i++)
   {
   stat = SCFGET(imnoa,felm,mapsize,&nval,pntra);
   if (fmtin != fmtout)
      conv_pix(pntrb,pntra,fmtout,fmtin,nval);
   stat = SCFPUT(imnob,felm,nval,pntrb);
   felm += nval;
   }


/*  handle over, underflow  */

if (fmtout == D_I1_FORMAT)
   {
   if (cuts[2] < 0) uflow = 1;
   if (cuts[3] > 255) oflow = 1;
   }
else if (fmtout == D_I2_FORMAT)
   {
   if (cuts[2] < -32768) uflow = 1;
   if (cuts[3] > 32767) oflow = 1;
```

```
   }
else if (fmtout == D_UI2_FORMAT)
   {
   if (cuts[2] < 0) uflow = 1;
   if (cuts[3] > 65534) oflow = 1;
   }
if (uflow == 1)
   SCTPUT("Warning: Format conversion results in underflow...");
if (oflow == 1)
   SCTPUT("Warning: Format conversion results in overflow...");


stat = SCDCOP(imnoa,imnob,1,cbuf); /*  copy all descriptors */


/* if delete flag set, delete the input frame  */

if ((dely[0] == 'D') || (dely[0] == 'd')) SCFDEL(infile);

SCSEPI();
}


------------------------------

The code of the functions called by the main program is omitted.
The interested reader is referred to the actual MIDAS code.

------------------------------
```

F.2 Examples Using Table Data

F.2.1 A Table Application in Fortran

The following example is a copy of a program in the ROMAFOT stellar photometry package. It reads the intermediate table, obtained after various fitted and verification command in the ROMAFOT package, and creates the final registration table, containing the final results. The program is tied into the MIDAS environment via the following procedure `regist.prg`:

```
!+++++++++++++++++++++++++++++++++++++++++++++++++++++++
!.IDENTIFICATION: REGIST.PRG
!.PURPOSE: Privat register command for the ROMAFOT package
!.USE: @@ regist int_tab reg_tab [wnd_opt] [obj_opt]
!.AUTHOR: Rein H. Warmels, ESO Garching
!.VERSION: 890614 RHW Creation
!.VERSION: 890803 RHW Implementation of table file system
!.VERSION: 900515 RHW Copied to own directory as an example
! ------------------------------------------------------
DEFINE/PARAM P1 ?  TBL "Enter input intermediate table:  "
DEFINE/PARAM P2 ?  TBL "Enter output registration table:  "
DEFINE/PARAM P3 A C "Enter window option:  "
DEFINE/PARAM P4 N C "Enter object option:  "
!
WRITE/KEYWORD IN_A {P1}
WRITE/KEYWORD OUT_A {P2}
WRITE/KEYWORD INPUTC/C/1/1 {P3}
WRITE/KEYWORD INPUTC/C/2/1 {P4}
!
RUN REGIST
```

To run the application execute the procedure via
`Midas 001> @@ regist int_tab reg_tab [wnd_opt] [obj_opt]`

The program is written in standard Fortran 77 code with the exceptions which are taken care of by the ESO provided preprocessor. This preprocessor is not needed on a VAX/VMS machine! In the code you will find a reference to common block `MID_INCLUDE:RFOTDECL.INC` which contains all relevant variables. In order to keep this example within reasonable limits this common block is not listed.

```
      PROGRAM REGIST
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C.IDENTIFICATION: RFOTREGIST
C.PURPOSE: Compute the absolute quantities and store the results
C          in the final MIDAS table
C.AUTHOR:  R. Buonanno, G. Buscema, C. Corsi, I. Ferraro, G. Iannicola
C          Osservatorio Astronomico di Roma
C.VERSION: 16.09.87 RB  First running version at ESO (outside MIDAS)
C          17.12.87 RHW Installation in MIDAS
C          24.10.88 RB  New version
C          14.06.89 RHW Rewritten for the portable MIDAS version
C                       Inclusion of MIDAS tables
C----------------------------------------------------------------------
      IMPLICIT       NONE
      INTEGER        NMAX
      INTEGER        NCPAR
      INTEGER        NRREG
      PARAMETER      (NMAX=256)
      PARAMETER      (NCPAR=12)
      PARAMETER      (NRREG=10000)
C ***
      INTEGER        ICPAR(NCPAR)
      INTEGER        IDENT(NRREG)
      INTEGER        IROW, ICIDN, IREGI, IGRP
      INTEGER        ISTAT, IAC, IAV
      INTEGER        I, IS, IH, IC, IK, IVN
      INTEGER        KUN, KNUL, KONT
      INTEGER        IPX, IPY
      INTEGER        K7
      INTEGER        LF9
      INTEGER        MADRID(1)
      INTEGER        NRINT,NCINT
      INTEGER        NACINT,NARINT,NSINT
      INTEGER        NOBJ, NGRP, NSR
      INTEGER        NCP, NHL, NCOM
      INTEGER        NMAX
      INTEGER        NCPAR, NRREG, NCREG
      INTEGER        REGTYP, REGCOL
      INTEGER        TIDINT
      INTEGER        TIDREG
C ***
      REAL           ALTMIN
      REAL           B, BETA
      REAL           BU(NMAX)
```

```
      REAL           DATR(NCPAR,NRREG)
      REAL           D1, D2, D3, D4, D6, D7
      REAL           FOG, FAT, FOND, FL
      REAL           GRE
      REAL           P(NMAX)
      REAL           SIGMA, SOFOT, SAT
      REAL           PP1, PP2
      REAL           SQM(NMAX), SIQ(NMAX)
      REAL           U
      REAL           V
      REAL           VOL
C ***
      CHARACTER*80   STRING
      CHARACTER*60   INTFIL
      CHARACTER*60   REGFIL
      CHARACTER*16   LABEL(NCPAR),REGLAB
      CHARACTER*16   UNIT(NCPAR),REGUNI
      CHARACTER*16   REGFOR
      CHARACTER*16   FORMR4,FORMI4
      CHARACTER*1    SST, CAR
C ***
      INCLUDE        'MID_INCLUDE:RFOTDECL.INC'  ! romafot common block
      INCLUDE        'MID_INCLUDE:ST_DEF.INC'    ! ST definitions
      COMMON         /VMR/MADRID(1)              ! DON'T FORGET !!!
      INCLUDE        'MID_INCLUDE:ST_DAT.INC'    ! ST assignments
C ***
      DATA           ICIDN/1/
      DATA           FORMI4/'I6'/
      DATA           FORMR4/'E12.4'/
      DATA     LABEL /'X       ', 'Y       ', 'INT     ', 'BG      ',
     2               'MAG1    ', 'MAG2    ', 'MAG3    ', 'MAG_CNV ',
     3               'SIGMA   ', 'BETA    ', 'SIQ     ', 'CHI_SQ  '/
      DATA     UNIT  /'PIXEL   ', 'PIXEL   ', '        ', '        ',
     2               'MAG.    ', 'MAG.    ', 'MAG.    ', 'MAG.    ',
     3               '        ', '        ', '        ', '        '/
C ***
 9001  FORMAT('*** INFO: Window ',I5,' not fitted')
 9002  FORMAT('*** INFO: Window ',I5,' not registered (no conv. ',
     *        'or already regist.)')
C
C *** start the code
      CALL STSPRO('REGISTER')
C
C *** get the input intermediate table and output registration table
```

```
      CALL STKRDC('IN_A',1,1,60,IAC,INTFIL,KUN,KNUL,ISTAT)
      CALL STKRDC('OUT_A',1,1,60,IAC,REGFIL,KUN,KNUL,ISTAT)
C
C ***
      CALL TBTOPN(INTFIL,F_I_MODE,TIDINT,ISTAT)    ! open interm. table
      CALL TBIGET(TIDINT,NCINT,NRINT,NSINT,NACINT,NARINT,ISTAT)  ! info
      IF (NRINT.EQ.0) THEN                              ! no rows in table
         STRING = '*** FATAL: There are no data in the intermediate '//
     2            'table'
         CALL STETER(9,STRING)
      ENDIF
C
C *** create the registration table
      NCREG = NCPAR + 1
      CALL TBTINI(REGFIL,0,F_O_MODE,NCREG,NRREG,TIDREG,ISTAT)
C
      REGTYP = D_I4_FORMAT
      REGFOR = FORMI4
      REGUNI = '   '
      REGLAB = 'IDENT'
      CALL TBCINI(TIDREG,REGTYP,1,REGFOR,REGUNI,
     2                   REGLAB,REGCOL,ISTAT)     ! create ident column
C
      DO 101 I = 1,NCPAR                            ! loop through columns
         ICPAR(I) = I + 1
         REGTYP   = D_R4_FORMAT
         REGFOR   = FORMR4
         REGUNI   = UNIT(I)
         REGLAB   = LABEL(I)
         CALL TBCINI(TIDREG,REGTYP,1,REGFOR,REGUNI,
     2                      REGLAB,REGCOL,ISTAT)   ! create the columns
  101 CONTINUE
C
C *** read the window and object option
      CALL STKRDC('INPUTC',1,1,1,IAV,CAR,KUN,KNUL,ISTAT)
      CALL STKRDC('INPUTC',1,2,1,IAV,SST,KUN,KNUL,ISTAT)
C
C *** read the table descriptor of the intermedaite table
      CALL INTDRD(TIDINT,NGRP,NOBJ,NSR,SAT,FAT,SIGMA,BETA,SOFOT,
     2                   ALTMIN,FOG)
C
C *** do the work
      KONT = 0
      IROW = 1
```

```
      IGRP = 1
C
 1001 CONTINUE                                     ! loop through the rows
         CALL INTWRD(TIDINT,IROW,NCP,NHL)                ! read the groups
         D1   = PARINT(1)
         D2   = PARINT(2)
         V    = PARINT(3)
         B    = PARINT(4)
         U    = PARINT(5)
         D3   = PARINT(6)
         D4   = PARINT(7)
         P(1) = PARINT(8)
         P(2) = PARINT(9)
         P(3) = PARINT(10)
         BETA = PARINT(11)
         D7   = PARINT(12)
         D6   = PARINT(13)
         FL   = PARINT(14)
C
         DO 1011 IS = 1,NCP                          ! copy the components
            P((IS-1)*4+4) = FITCMP((IS-1)*6+1)
            P((IS-1)*4+5) = FITCMP((IS-1)*6+2)
            P((IS-1)*4+6) = FITCMP((IS-1)*6+3)
            P((IS-1)*4+7) = FITCMP((IS-1)*6+4)
            SQM(IS)       = FITCMP((IS-1)*6+5)
            SIQ(IS)       = FITCMP((IS-1)*6+6)
 1011    CONTINUE
C
         DO 1012 IH = 1,NHL                              ! copy the holes
            BU((IH-1)*3+1) = FITHOL((IH-1)*3+1)
            BU((IH-1)*3+2) = FITHOL((IH-1)*3+2)
            BU((IH-1)*3+3) = FITHOL((IH-1)*3+3)
 1012    CONTINUE
C
         IREGI = 0
         IF (CAR.EQ.'A') THEN
            IREGI = 1
         ENDIF
         IF (IREGI.EQ.1) THEN
            IPX  = INT(D1)
            IPY  = INT(D2)
            NCOM = 0
            LF9  = 0
C
```

```
            DO 1013 IC = 1,NCP
               K7 = (IC-1)*4 +4
               IVN = FLGCMP(IC)
               IF (IVN.EQ.1 .OR. CAR.EQ.'A' .OR. IVN.EQ.2) THEN
                  IF (IVN.EQ.1 .OR. (IVN.EQ.2 .AND. SST.EQ.''Y')) THEN
C ***
                  CALL STTPUT('HERE WE WILL COMPUTE THE FINAL'//
     2                        ' RESULTS,ISTAT)
C
C *** copy back to register array
                  IDENT(KONT)   = IDNGRP*100+(IDNCMP(IC)-100)
                  DATR(1,KONT)  = PP1
                  DATR(2,KONT)  = PP2
                  DATR(3,KONT)  = P(K7)
                  DATR(4,KONT)  = FOND
                  DATR(5,KONT)  = V
                  DATR(6,KONT)  = B
                  DATR(7,KONT)  = U
                  DATR(8,KONT)  = VOL
                  DATR(9,KONT)  = P(K7+3)
                  DATR(10,KONT) = BETA
                  DATR(11,KONT) = SIQ(IC)
                  DATR(12,KONT) = SQM(IC)
               END IF
 1013       CONTINUE
C
         ELSE
            IF (GRE.EQ.0) THEN
               WRITE (STRING,9001) IC
            ELSE
               WRITE(STRING,9002) IC
            END IF
            CALL STTPUT(STRING,ISTAT)
         END IF
C
         IROW = IROW + NCP + NHL            ! row index for next group
         IF (IROW.LE.NRINT) THEN     ! did we have all rows (= groups)
            GO TO 1001                                             ! NO
         ENDIF
C
C *** fill the  registration table
      DO 102 IK = 1,KONT
         CALL TBERWRI(TIDREG,IK,ICIDN,IDENT(IK),ISTAT)
         CALL TBRWRR(TIDREG,IK,NCPAR,ICPAR,DATR(1,IK),ISTAT)
```

```
  102 CONTINUE
      CALL TBSINI(TIDREG,ISTAT)              ! initialize the selecion
      CALL TBTCLO(TIDREG,ISTAT)                      ! close the table
      CALL STSEPI                                       ! over and out
      END



      SUBROUTINE INTDRD(IDEN,INTG1,INTG2,INTG3,
     2                 REAL1,REAL2,REAL3,REAL4,REAL5,REAL6,REAL7)
C+++
C.PURPOSE: Write the table info into the descriptor
C---
      IMPLICIT NONE
      INTEGER  IDEN
      INTEGER  INTG1,INTG2,INTG3
      REAL     REAL1,REAL2,REAL3,REAL4,REAL5,REAL6,REAL7
C
      INTEGER  ISTAT, IACT, KUN, KNUL
C
      INTEGER IOUT(3)
      REAL    ROUT(7)
C
      INCLUDE      'MID_INCLUDE:ST_DEF.INC/NOLIST'
      INCLUDE      'MID_INCLUDE:ST_DAT.INC/NOLIST'
C
      CALL STDRDR(IDEN,'INTPAR_R',1,7,IACT,ROUT,KUN,KNUL,STAT)
      REAL1  = ROUT(1)                         !
      REAL2  = ROUT(2)                         !
      REAL3  = ROUT(3)                         !
      REAL4  = ROUT(4)                         !
      REAL5  = ROUT(5)                         !
      REAL6  = ROUT(6)                         !
      REAL7  = ROUT(7)                         !
C
      CALL STDRDI(IDEN,'INTPAR_I',1,3,IACT,IOUT,KUN,KNUL,ISTAT)
      INTG1  = IOUT(1)                             ! number of groups
      INTG2  = IOUT(2)                         ! number of components
      INTG3  = IOUT(3)                         ! number of iterations
C
      RETURN
      END
```

```
      SUBROUTINE INTWRD(TID,IROW,NCOMPS,NHOLES)
C+++
C.Purpose: Reads the intermediate table starting from row = IROW.
C.         The data will be stores in a THREE common blocks to be read
C.         by calling program.
C---
      IMPLICIT       NONE
      INTEGER        TID                         ! table identification
      INTEGER        IROW                        ! row indication where to start
      INTEGER        NCOMPS                      ! number of components in window
      INTEGER        NHOLES                      ! number of holes in window
C
      INCLUDE        'MID_INCLUDE:RFOTDECL.INC/NOLIST'
C
      INTEGER        ISTAT
      INTEGER        IS, IR, IS, IH
      INTEGER        ICGRP
      INTEGER        ICIDN
      INTEGER        ICGEN(NINTP)
      INTEGER        ICFLG
      INTEGER        ICPAR(NINTC)
      INTEGER        TINULL
C
      DOUBLE PRECISION TDNULL,TDTRUE,TDFALS
C
      REAL           RNST, RNHL
      REAL           ROUT(NINTC)
      REAL           TRNULL,TBLSEL
C
      LOGICAL        NULL(NINTP)
C
      DATA           ICGRP/1/
      DATA           ICIDN/2/
      DATA           ICGEN/3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18/
      DATA           ICFLG/19/
      DATA           ICPAR/20,21,22,23,24,25/
C
C *** start the code; first define the columns
      CALL TBMNUL(TINULL,TRNULL,TDNULL)
      CALL TBMCON(TBLSEL,TDTRUE,TDFALS)
C
C *** read general parameters
      CALL TBERDI(TID,IROW,ICGRP,IDNGRP,NULL,ISTAT)          ! group nr.
```

```
      CALL TBRRDR(TID,IROW,NINTP,ICGEN,PARINT,NULL,ISTAT)    ! gen. par.
C
C *** read the star parameters
      CALL TBERDR(TID,IROW,ICGEN(15),RNST,NULL,ISTAT)        ! # stars
      CALL TBERDR(TID,IROW,ICGEN(16),RNHL,NULL,ISTAT)        ! # holes
      NCOMPS = INT(RNST)
      NHOLES = INT(RNHL)
C
C *** get the identification and fit parameters for the stars
      IF (NCOMPS.GT.0) THEN
         DO 100 IS = 1, NCOMPS
            IR = IROW + IS - 1
            CALL TBERDI(TID,IR,ICIDN,IDNCMP(IS),NULL,ISTAT)  ! ident cp.
            CALL TBERDI(TID,IR,ICFLG,FLGCMP(IS),NULL,ISTAT)  ! flag
            CALL TBRRDR(TID,IR,NINTC,ICPAR,ROUT,NULL,ISTAT)  ! comp. par.
            FITCMP((IS-1)*6+1) = ROUT(1)
            FITCMP((IS-1)*6+2) = ROUT(2)
            FITCMP((IS-1)*6+3) = ROUT(3)
            FITCMP((IS-1)*6+4) = ROUT(4)
            FITCMP((IS-1)*6+5) = ROUT(5)
            FITCMP((IS-1)*6+6) = ROUT(6)
 100     CONTINUE
      ENDIF
C
C*** get the "hole" parameters
      IF (NHOLES.GT.0) THEN
         DO 200 IH = 1, NHOLES
            IR = IROW + NCOMPS + IH - 1
            CALL TBERDI(TID,IR,ICIDN,IDNHOL(IH),NULL,ISTAT)  ! ident
            CALL TBERDI(TID,IR,ICFLG,FLGHOL(IH),NULL,ISTAT)  ! flag
            CALL TBRRDR(TID,IR,NINTH,ICPAR,ROUT,NULL,ISTAT)  ! hole par.
            FITHOL((IH-1)*3+1) = ROUT(1)
            FITHOL((IH-1)*3+2) = ROUT(2)
            FITHOL((IH-1)*3+3) = ROUT(3)
  200    CONTINUE
      ENDIF
C
      RETURN
      END
```

F.2.2 A Table Application in C

The following program `echregr` is written in C and fits a 2D polynomial to the echelle order positions. It is the standard reduction application in the Echelle context which implements the `REGRESSION/ECHELLE` command. This program is tied into MIDAS via the following procedure, `echregr.prg`:

```
! @(#)echregr.prg 3.3 (ESO-IPG) 4/8/92 18:10:06
!+++++++++++++++++++++++++++++++++++++++++++++++++++++++
!
!.COPYRIGHT   (C) 1991 European Southern Observatory
!.IDENT       echregr.prg
!.AUTHOR      Pascal Ballester,  ESO - Garching
!.KEYWORDS    Spectroscopy, Echelle,
!.PURPOSE     Fit mesured position of orders by a bivariate polynomial
!.VERSION     1.0    Creation    13-OCT-1983
!
!-------------------------------------------------------
!
IF P1 .NE. "?" WRITE/KEYWORD    DEFPOL/I/1/2  {P1}
DEFINE/PARAM   P2    3      N    "Number of outliers rejection:"
DEFINE/PARAM   P3    2.0    N    "Absolute tolerance (pixels):"
DEFINE/PARAM   P4    4.5    N    "Kappa-sigma factor :"
!
DEFINE/LOCAL  CNT/I/1/1      0
DEFINE/LOCAL  THRES/R/1/1    0.
DEFINE/LOCAL  ORDER/I/1/1    0
DEFINE/LOCAL  RMSMAX/R/1/1   0.
DEFINE/LOCAL  RMS/R/1/1      0.
DEFINE/LOCAL  REJECT/I/1/300 0  ALL
DEFINE/LOCAL  NBREJ/I/1/1    0
!
WRITE/OUT  "Warning:  Present display shows the positions"
WRITE/OUT  "          used to fit the orders"
LOAD/ECHELLE ECHELLE RAW
!
SELECT/TABLE  {ORDTAB}    (:X.GE.1.AND.:X.LE.{IMSIZE(1)})
SELECT/TABLE  {ORDTAB}    SELECT.AND.(:Y.GE.{SCAN(1)}.AND.:Y.LE.{SCAN(2)})

WRITE/KEYWORD IN_A    {ORDTAB}
WRITE/KEYWORD OUT_A   middummw.tbl
INPUTI(1) = ECHORD(1)
```

```
RUN echregr

COPY/TABLE  &w  &s
SORT/TABLE  &s  :RMS
ORDER = ECHORD(1)/2
RMSMAX = 3.5*{middumms.tbl,:RMS,@{ORDER}}

WRITE/OUT "*** Order by order linear rms ***"
READ/TABLE &w
WRITE/OUT "Maximum admissible rms : {RMSMAX}"

DO ORDER = 1  {ECHORD(1)}

 RMS = {middummw.tbl,:RMS,@{ORDER}}
 IF RMS .GT. RMSMAX THEN
   WRITE/OUT   "Warning :   Rejected order {ORDER}. Rms = {RMS}
   NBREJ = NBREJ + 1
   REJECT({NBREJ}) = ORDER
 ENDIF

ENDDO

DO ORDER = 1 {NBREJ}
    SELECT/TAB {ORDTAB} SELECT.AND.:ORDER.NE.{REJECT({ORDER})}
ENDDO

INPUTI(1) = DEFPOL(2) + 1      ! Degree Y + 1
INPUTI(2) = ECHORD(1) - NBREJ  ! Number of orders minus rejected orders
IF INPUTI(1) .GT. INPUTI(2) THEN
   WRITE/OUT "*****************************************************"
   WRITE/OUT "**** Warning : Number of selected orders {INPUTI(2)}
   WRITE/OUT "**** is too small for the current value of echelle
   WRITE/OUT "**** parameter DEFPOL(2)={DEFPOL(2)}
   WRITE/OUT "*****************************************************"\\
   HELP/ECHELLE DEFPOL
ENDIF

REGRESSION/POLY  {ORDTAB} :Y :X,:ORDER {DEFPOL(1)},{DEFPOL(2)}  KEYLONG
SAVE/REGR   {ORDTAB} COEFF  KEYLONG
COMPUTE/REGR   {ORDTAB} :YFIT     = COEFF
COMPUTE/TABLE    {ORDTAB} :RESIDUAL = :Y - :YFIT
THRES = {P4} * KEYLONGR(5)
IF THRES(1) .GT. {P3}   THRES = {P3}
```

```
SELECT/TABLE    {ORDTAB}  ABS(:RESIDUAL).LT.{THRES}

IF {P2} .LT. 1  GOTO END

DO CNT = 1   {P2}

   REGRESSION/POLY  {ORDTAB} :Y :X,:ORDER {DEFPOL(1)},{DEFPOL(2)}  KEYLONG
   SAVE/REGR   {ORDTAB} COEFF   KEYLONG
   COMPUTE/REGR   {ORDTAB} :YFIT = COEFF
   COMPUTE/TABLE    {ORDTAB} :RESIDUAL = :Y - :YFIT
   THRES = {P4} * KEYLONGR(5)
   IF THRES(1) .GT. {P3}   THRES = {P3}
   SELECT/TABLE    {ORDTAB}  ABS(:RESIDUAL).LT.{THRES}

ENDDO
!
END:
-PURGE      {ORDTAB}
```

To run the application execute the procedure via:
`Midas 001> @@ echregr defpol ninter absres kappa`

```
/* @(#)echregr.c 3.2 (ESO-IPG) 4/2/92 10:49:52 */
/* Program  : echregr.c                          */
/* Author   : P. Ballester  -  ESO Garching      */
/* Date     : 17.03.92                           */
/*                                               */
/* Purpose  :                                    */
/*   Performs a linear fit on independent orders */
/*   and writes downs the rms in a table         */
/*                                               */
/* Input    :                                    */
/*      IN _A     Name of order  table           */
/*      INPUTI(1) Number of orders               */
/*      OUT_A     Name of output table           */
/*                                               */

#include <math.h>
#include <tbldef.h>
#include <midas_def.h>

main()
{
      int     kunit;
      char    outtab[TEXT_LEN], ordtab[TEXT_LEN], text[TEXT_LEN];

      int     order, nb_order, present_order, row;
      int     rmscol, ordcol;
      int     order_col, xcol, ycol;
      int     ncol, nrow, nsort, allcol, allrow;

      int     select, null, status;
      int     iav, actvals, tid, tidord;

      double  x, y, cnt, sx, sy, sx2, sxy, sy2;
      double  det, a, b, rms;

      SCSPRO("echregr");

      SCKGETC ("IN_A", 1, 60,  &actvals, ordtab);
      SCKGETC ("OUT_A", 1, 60, &actvals, outtab);

      SCKRDI ("INPUTI", 1, 1, &actvals, &nb_order, &kunit, &null);

      TCTOPN (ordtab, F_I_MODE, &tidord);
      TCIGET (tidord, &ncol, &nrow, &nsort, &allcol, &allrow);
```

```
    TCCSER (tidord, "ORDER", &order_col);
    TCCSER (tidord, "X",     &xcol);
    TCCSER (tidord, "Y",     &ycol);

    TCTINI (outtab, F_TRANS, F_IO_MODE, 3, 100, &tid);

    TCCINI (tid, D_R4_FORMAT, 1, "I6",    "  ", "ORDER", &ordcol);
    TCCINI (tid, D_R4_FORMAT, 1, "F12.4", "  ", "RMS",   &rmscol);

    row = 1;

    for (order=1; order<=nb_order; order++) {

        cnt=0., sx=0., sy=0., sx2=0., sxy=0., sy2 = 0.;
        present_order = order;  /* Just to start the while () */

        while (present_order == order) {

          TCSGET(tidord, row, &select);

          if (select) {
          TCERDD(tidord, row,   xcol,  &x,  &null);
          TCERDD(tidord, row,   ycol,  &y,  &null);
          cnt += 1., sx += x, sy += y, sx2 += x*x, sy2 += y*y, sxy += x*y;
     }

          if (row >= nrow) break;
          TCERDI(tidord, ++row,   order_col, &present_order, &null);

  }

        if (cnt >= 3) {
          det = cnt*sx2 - sx*sx;
          a   = (sy*sx2 - sx*sxy)/det;
          b   = (cnt*sxy - sx*sy)/det;
          rms = sqrt((sy2 - a*a*cnt - 2.*b*a*sx - b*b*sx2)/cnt);
}
        else    rms = 99999.;


        TCEWRI(tid, order, ordcol, &order);
        TCEWRD(tid, order, rmscol, &rms);

   }
```

```
    /* Close everything and good bye. */
    TCTCLO(tidord), TCTCLO(tid), SCSEPI();

}
```

F.3 Examples Using the Graphics Interfaces

F.3.1 A Fortran Application

The following program `pltab` is written in FORTRAN 77 and simply plots two columns of a table against each other in the current active viewport. The example demonstrates the usage of the high level plot interfaces: `PTKWRx`, `PTOPEN`, `PTAXES`, `PTKRDI`, `PTDATA`, and `PTTEXT`. This program is tied into MIDAS through the following procedure, `pltab.prg`:

```
!+++++++++++++++++++++++++++++++++++++++++++++++++++++++
!.IDENTIFICATION: pltab.prg
!.PURPOSE: MIDAS proceduce to plot one or two columns of a table
!.USE:     execute as @@ pltab par1 [par2 [par3]] [par4] where:
!          par1 = input table
!          par2 = column 1
!          par3 = column 2
!          par4 = sc_x,sc_y,off_x,off_y (defaults device filling)
!.AUTHOR:  R.H. Warmels  ESO - Garching
!.VERSION: 931103 RHW Created for environment document
! ------------------------------------------------------------------------
DEFINE/PARAM P1 ?  TABLE "Enter table:"
!
IF P2(1:1) .EQ. "?" THEN
   DEFINE/PARAM P3 ? C "Enter input for the ordinate column:"
   IF P3(1:3) .EQ. "SEQ" THEN
      WRITE/OUT "*** FATAL: Illegal combination of column parameters"
      RETURN
   ENDIF
ENDIF
!
IF P2(1:3) .EQ. "SEQ" THEN
   DEFINE/PARAM P3 ? C "Enter input for the ordinate column:"
   IF P3(1:3) .EQ. "SEQ" THEN
      WRITE/OUT "*** FATAL: Illegal combination of column parameters"
      RETURN
   ENDIF
ENDIF
!
WRITE/KEYWORD IN_A {P1}
!
RUN pltab.exe
WRITE/KEYWORD PLCDATA/C/1/60  {P1}
WRITE/KEYWORD PLCDATA/C/61/20 "TABLE       "
```

```
!
@ purgeplt {P1}
IF MID$PLOT(26:30) .EQ. "SPOOL" THEN                  !make the plot if spool is on
   @ sendplot
ENDIF
```

To run the application execute the procedure via:

```
Midas 001> @@ pltab table column_1 column_2
```

```
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C.COPYRIGHT: Copyright (c) 1987 European Southern Observatory,
C                                        all rights reserved
C.IDENTIFICATION: PLTAB
C.LANGUAGE:  F77+ESOext
C.AUTHOR:    Rein H. Warmels
C.PURPOSE:   Plots two columns of a table
C.NOTE:      PLOTTBL uses the plotting routines available in the plot
C            library which again uses the low level AGL routines.
C.VERSION:   931103 RHW Created for the environment document
C ----------------------------------------------------------------------
      PROGRAM      PLTTBL                ! program PLTTBL   *** main body ***
      IMPLICIT     NONE
C
      INTEGER      COL(3)
      INTEGER      I, IST, IAC, IL1, IL2
      INTEGER      KUN, KNUL
      INTEGER      ISTAT, PLMODE, ACCESS
      INTEGER      MADRID(1)
      INTEGER      NCOL, NSC, NAC, NAR
      INTEGER      NCOLUM,NCOL1,NCOL2,NROW
      INTEGER      TID
      REAL         FRAME(8)
      REAL         RMIN,RMAX

      CHARACTER    LABEL1*80,LABEL2*80,LABEL3*80
      CHARACTER    TEXT*80
      CHARACTER*16 LABEL(3),UNIT(3),OLAB
      CHARACTER    TABLE*60,SEL*80
      CHARACTER*17 COLUMN(3)
C
      INCLUDE      'MID_INCLUDE:TABLES.INC/NOLIST'
      COMMON       /VMR/MADRID
      INCLUDE      'MID_INCLUDE:TABLED.INC/NOLIST'
      DATA         SEL/' '/
      DATA         ACCESS/0/
C
 9000 FORMAT (I4)
 9001 FORMAT('*** WARNING: zero dynamics range in x; '//
     2       'value = ',G13.6)
 9002 FORMAT('*** WARNING: zero dynamics range in y; '//
     2       'value = ',G13.6)
C
C *** start executable code
```

```
      CALL STSPRO('PLOTTAB')                                !start comm. with MIDAS
C
C *** read parameters
      CALL STKRDC('IN_A',1,1,60,IAC,TABLE,KUN,KNUL,ISTAT)      ! get table name

C
C *** read columns
      CALL STKRDC('P2',1,1,40,NCOL1,COLUMN(1),KUN,KNUL,ISTAT)    ! first column
      CALL STKRDC('P3',1,1,40,NCOL2,COLUMN(2),KUN,KNUL,ISTAT)   ! second column
C
C *** this procedure read the table
      CALL TBTOPN(TABLE,F_I_MODE,TID,ISTAT)
      CALL TBIGET(TID,NCOL,NROW,NSC,NAC,NAR,ISTAT)     ! read table information
C
      IF (NROW.LE.0) THEN
          CALL STTPUT('*** FATAL: No points in the table ... ',ISTAT)
          CALL STSEPI
      END IF
      CALL TDRSEL(TID,SEL,ISTAT)                                !  table selection
C
C *** get column adresses
      IST    = 1
      DO 10 I = 1,2
         CALL TBCSER(TID,COLUMN(I),COL(I),ISTAT)              ! find column no.
         CALL TBLGET(TID,COL(I),LABEL(I),ISTAT)
C
         CALL TBUGET(TID,COL(I),UNIT(I),ISTAT)                  !  read units

   10 CONTINUE
C
C *** get the labeling of the axes; first the x-axis
      OLAB   = 'COLUMN '
      IF (LABEL(1)(1:2).EQ.'  ') THEN
         WRITE (OLAB(7:10),9000) COL(1)
      ELSE
         OLAB   = LABEL(1)
      END IF
      LABEL1 = OLAB
      IL1= INDEX(LABEL1,' ')
      IL2= INDEX(UNIT(1),' ')-1
      LABEL1 = LABEL1(1:IL1)//'('//UNIT(1)(1:IL2)//')'
C
C *** the y axis
      OLAB   = 'COLUMN '
```

```
      IF (LABEL(2)(1:2).EQ.'  ') THEN
          WRITE (OLAB(7:10),9000) COL(2)
      ELSE
          OLAB   = LABEL(2)
      END IF
      LABEL2 = OLAB
      IL1= INDEX(LABEL2,' ')
      IL2= INDEX(UNIT(2),' ')-1
      LABEL2 = LABEL2(1:IL1)//'('//UNIT(2)(1:IL2)//')'
C
C *** calculate frame
      CALL TDUMNX(TID,COL(1),NROW,0,RMIN,RMAX)
      IF (RMIN.EQ.RMAX) THEN
          WRITE(TEXT,9001) RMIN
          CALL STTPUT(TEXT,ISTAT)
      ENDIF
      FRAME(1) = RMIN
      FRAME(2) = RMAX
      CALL PTKWRR('XWNDL',4,FRAME(1))

      CALL TDUMNX(TID,COL(2),NROW,0,RMIN,RMAX)
      IF (RMIN.EQ.RMAX) THEN
         WRITE(TEXT,9002) RMIN
         CALL STTPUT(TEXT,ISTAT)
      ENDIF
      FRAME(5) = RMIN
      FRAME(6) = RMAX
      CALL PTKWRR('YWNDL',4,FRAME(5))
C
C *** get the plot setup
      PLMODE = -1
      CALL PTOPEN(' ',' ',ACCESS,PLMODE)                  ! access and plot mode
C
C *** make the frame, tickmarks, etc ...
      LABEL3 = 'Table: '//TABLE
      CALL PTAXES(FRAME(1),FRAME(5),LABEL1,LABEL2,'TITLE='//LABEL3)
C
C *** do the work
      CALL PLTBL(TID,NCOLUM,COL(1),COL(2),NROW)
C
C *** good bye and finish
      CALL TBTCLO(TID,ISTAT)
      CALL PTCLOS()
      CALL STSEPI                               ! stop communication with MIDAS
```

```
      END



      SUBROUTINE PLTBL(TID,NCOLUM,I1,I2,NROW)
C +++
C.PURPOSE:      Plots one or two columns of a table
C.AUTHOR:       Rein H. Warmels
C.COMMENTS:     none
C.VERSION:      931103 RHW Createsd for environment document
C ---
      IMPLICIT    NONE
      INTEGER     TID                ! Table identifier
      INTEGER     NCOLUM             ! # of columns to be plotted
      INTEGER     I1                 ! index to first column
      INTEGER     I2                 ! index to second column
      INTEGER     NROW               ! number of row
C
      INTEGER     NPMAX
      PARAMETER   (NPMAX=100000)
      INTEGER     IFIRST, I, ISTAT, IR, IAC
      INTEGER     STYPE, LTYPE
      REAL        VX, VY
      REAL        XPS(NPMAX), YPS(NPMAX)
      LOGICAL     IPLOT,ISEL,INULL
      CHARACTER   TEXT*80
C
 9001 FORMAT('*** FATAL: Maximum number of table entries is ',I8)
C
      IFIRST = 0
      IR     = 0
C
      DO 10 I = 1,NROW
         IPLOT  = .TRUE.
         CALL TBSGET(TID,I,ISEL,ISTAT)
         IF (ISEL) THEN
            CALL TBRRDR(TID,I,1,I1,VX,INULL,ISTAT)
            IF (INULL) THEN
               IPLOT  = .FALSE.
            ENDIF
C
            CALL TBRRDR(TID,I,1,I2,VY,INULL,ISTAT)
            IF (INULL) THEN
               IPLOT  = .FALSE.
```

```
            ENDIF
            IF (IPLOT) THEN
               IR      = IR + 1
               IF (IR.GT.NPMAX) THEN
                  WRITE (TEXT,9001) NPMAX
                  CALL STTPUT(TEXT,ISTAT)
                  CALL STSEPI
               ENDIF
               XPS(IR) = VX
               YPS(IR) = VY
            ENDIF
         ENDIF
   10 CONTINUE
C
      CALL PTKRDI('STYPE',1,IAC,STYPE)
      CALL PTKRDI('LTYPE',1,IAC,LTYPE)
      IF (LTYPE.EQ.0 .AND. STYPE.EQ.0) THEN
         CALL STTPUT('*** FATAL: LTYPE and STYPE '//
     2               'both equal 0: NO PLOT',ISTAT)
         CALL PTCLOS()
         CALL STSEPI
      ELSE
         CALL PTDATA(STYPE,LTYPE,0,XPS,YPS,0.0,IR)
      ENDIF
C
C *** end of the plotting
      RETURN
      END
```

F.3.2 A C Application Using Graphics Interfaces

The following program `plotexampl` is written in C and plots a line of a frame in the current active viewport, like the standard application `PLOT/ROW`. This example demonstrated the usage of the high level plot interfaces: `PCKRDx`, `PCKWRx`, `PCOPEN`, `PCAXES`, `PCDATA`, `PCGCUR` and `PCTEXT`. This program is tied into MIDAS through the following procedure,

```
!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
!.IDENTIFICATION: plotexampl.prg
!.PURPOSE:        MIDAS procedure to plot a line of an image
!                 execute as @@ plotexampl par1 [par2] [par3] [par4] [par5]
!                 where:
!                 par1 = input frame
!                 par2 = line number (defaulted to 1)
!                 par3 = 1st point,last point on the line (defaulted to 1,npix)
!                        this parameters is read in automatic scaling mode only
!                 par4 = sc_x,sc_y,off_x,off_y (defaults device filling)
!                 par5 = options for AG_AXES
!.AUTHOR:         R.M. van Hees
!.VERSION:        RvH 931025
! -------------------------------------------------------------------------
DEFINE/PARAM P1 ? IMA "Enter image:"               ! frame
WRITE/KEYWORD  IN_A   {P1}
DEFINE/PARAM P2 @1                                 ! line number
!
IF {PLRSTAT(1)} .NE. 0.0 .AND. {PLRSTAT(2)} .NE. 0.0 THEN
   DEFINE/PARAM P3 MANUAL C
ELSE
   DEFINE/PARAM P3 <,> C
   IF P3(1:1) .EQ. "M" THEN
      WRITE/OUT "*** FATAL: X axis not manually set; use SET/GRAPHICS"
      RETURN
   ENDIF
ENDIF
!
DEFINE/PARAM P4 0.,0.,-999,-999 NUM
@ plscoff {P4}                                   ! get the scales and offsets
WRITE/KEYWORD  INPUTC {P5}                           ! options for AGL
!
DATTIM = M$TIME()                      ! time and date of the plot
RUN plotexampl                                   ! draw the plot
!
WRITE/KEYWORD PLCDATA/C/1/60  {P1}                   ! name of data structure
WRITE/KEYWORD PLCDATA/C/61/20 "FRAME       "         ! type of data structure
```

```
!
@ purgeplt {P1}
IF MID$PLOT(26:30) .EQ. "SPOOL" THEN              ! make plot if spool is on
   @ sendplot
ENDIF
```

To run the application execute the procedure via:

```
Midas 001> @@ plotexampl frame line_nr start,end sc_x,sc_y,off_x,off_y AGL_options
```

```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.COPYRIGHT   (c) 1993 European Southern Observatory
.IDENTifer   PLOTEXMPL
.AUTHOR      R.M. van Hees IPG-ESO Garching
.KEYWORDS    Graphics, bulk data frame, one-dimensional plotting
.LANGUAGE    C
.PURPOSE     Plot a line of a FRAME
  in/output: IN_A/C/1/60  = input frame
             P2/C/72      = line number (default 1)
             P3/C/72      = first and last pixel of the line
                            (default: the whole line).
             INPUTC/1/80  = options for PCAXES
.RETURNS
.COMMENTS    Test program for high level plot interfaces

.ENVIRONment MIDAS
             #include <midas_def.h>     Prototypes for MIDAS interfaces
             #include <plot_def.h>      General symbols for Plot routines

.VERSION     1.0     09-Sep-1993   Creation R.M. van Hees
------------------------------------------------------------*/
/*
 * Define _POSIX_SOURCE to indicate
 * that this is a POSIX program
 */
#define _POSIX_SOURCE 1

/*
 * definition of the used functions
 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

/*
 * define some macros and constants
 */
#include <midas_def.h>
#include <plot_def.h>

#define FOR_Y           4
#define PLDIM2          2                      /* only two dimensional plots */
#define KEY_EXIT        32            /* AGL key code for middle mouse buton */
```

```
/*
 * here starts the code of the function
 */
int main()
{
int     actvals, binmod, ii, imf, key, ltype, naxis, nrprow, stat,
        stype, npix[PLDIM2];

float   xpos, ypos, *xdata, *ydata, area[4], image[4], wcfram[8];

double  start[PLDIM2], step[PLDIM2];

char    *bin, *cmnd, *name, *cunit, *ident, *input, *buff, *text, *option,
        *label[PLDIM2], *pntr, *cpntr;

/*
 * initialised variables
 */
int   access =  0,                    /* parameter for PCOPEN: NEW plot mode */
      plmode = -1;                     /* plot mode taken from keyword PMODE */
float y_off  = 0.0;              /* no offset in the Y direction of the plot */

char  *err_1dim  = "*** WARNING: Image column contains only one point!",
      *err_flat  = "*** WARNING: zero dynamic range in data at %13.8g";
/*
 * allocate memory for different character pointers and initialise a few
 */
bin    = osmmget(4);
cmnd   = osmmget(21);
ident  = osmmget(33);
name   = osmmget(61);
input  = osmmget(73);
option = osmmget(81);
buff   = osmmget(81);
text   = osmmget(81);
cunit  = osmmget(PLDIM2 * 16 + 1);
for ( ii = 0; ii < PLDIM2; ii++ ) label[ii] = osmmget(81);

strcpy( label[0], "Position (" );
strcpy( label[1], "Pixel value (" );
strcpy( ident, "                                " );    /* SCIGET needs this */
*cunit = '\0';
for ( ii = 0; ii < PLDIM2; ii++ ) strncat( cunit, "                ", 16 );
```

```
/*
 * start of executable code
 */
stat = SCSPRO( "PLEXMPL" );                    /*contact with the MIDAS monitor*/

/*
 * find file name and read header information
 */
stat = SCKGETC( "IN_A", 1, 60, &actvals, name );
stat = SCIGET( name, D_R4_FORMAT, F_I_MODE,F_IMA_TYPE, PLDIM2,
               &naxis, npix, start, step, ident, cunit, &cpntr, &imf );

if ( npix[0] == 1 ) stat = SCTPUT( err_1dim );

/*
 * line number (Y-coordinate)
 */
image[2] = image[3] = 1;                         /* default line number = 1 */
if ( naxis > 1 )
   { stat = SCKGETC( "P2", 1, 72, &actvals, input );
     image[2] = image[3] = GETSIN( input, npix[1], start[1], step[1] );
     BOXPTW( image+2, npix[1], start[1], step[1], area+2 );
   }
/*
 * find first and last pixel along the row (X-coordinate)
 */
stat = SCKGETC( "P3", 1, 72, &actvals, input );
strcpy( buff, input );
image[0] = GETSIN( strtok( buff,"," ), npix[0], start[0], step[0] );
pntr = strchr( input, ',' );
image[1] = GETSIN( pntr+1, npix[0], start[0], step[0] );
BOXPTW( image, npix[0], start[0], step[0], area );
/*
 * Store the size of the extracted data in PLRGRAP
 */
PCKWRR( "PIXEL", 4, image );

/*
 * allocate virtual memory for a single line of the frame
 */
nrprow =(int) fabs(image[1] - image[0]) + 1;    /*number of points in the row*/
xdata = (float *) osmmget( nrprow * sizeof(float));
ydata = (float *) osmmget( nrprow * sizeof(float));
```

```
/*
 * copy the data of the row
 */
GETBDF( cpntr, image, npix, start, step, xdata, ydata );

/*
 * Get the manual setting for the axes
 */
PCKRDR( "XAXIS", 4, &actvals, wcfram );
PCKRDR( "YAXIS", 4, &actvals, wcfram+FOR_Y );

/*
 * If the X axis is not defined by the user we calculate frame along X axis
 */
if ( fabs( *wcfram ) < PLT_EPS && fabs( *(wcfram+1) ) < PLT_EPS )
   { wcfram[0] = area[0];
     wcfram[1] = area[1];
   }
/*
 * If the Y axis is not defined by the user we calculate frame along Y axis
 */
if ( fabs( *(wcfram+FOR_Y)) < PLT_EPS && fabs( *(wcfram+FOR_Y+1)) < PLT_EPS )
   { MINMAX( ydata, nrprow, wcfram + FOR_Y, wcfram + FOR_Y + 1 );
     if ( wcfram[FOR_Y] == wcfram[FOR_Y+1] )
        { sprintf( buff, err_flat, wcfram[FOR_Y] );
          SCTPUT( buff );
        }
   }
/*
 * the tick marks will be set by PCAXES
 */
wcfram[2] = wcfram[3] = wcfram[FOR_Y+2] = wcfram[FOR_Y+3] = 0.0;

/*
 * still we have to save the coordinate frame in PLRGRAP
 */
PCKWRR( "XWNDL", 4, wcfram );
PCKWRR( "YWNDL", 4, wcfram+FOR_Y );

/*
 * setup graphics device according to MIDAS settings
 * default graphics display & meta-file name
 */
```

```
PCOPEN( " ", " ", access, &plmode );

/*
 * plot the axes
 */
if ( strlen( cunit ) > 0 )
   { strncat( label[0], cunit+17, 16 );
     strncat( label[1], cunit, 16 );
   }
for ( ii = 0; ii < PLDIM2; ii++ )
    { strcat( label[ii], ")" );
      LABSTR( label[ii] );
    }
/*
 * draw the axes and define the clipping area
 */
stat = SCKGETC( "INPUTC", 1, 80, &actvals, option );
PCAXES( wcfram, wcfram+FOR_Y, label[0], label[1], option );

/*
 * get the symbol type, line type and binmode
 */
PCKRDI( "STYPE", 1, &actvals, &stype );
PCKRDI( "LTYPE", 1, &actvals, &ltype );
PCKRDC( "BINMOD", 4, &actvals, bin );
binmod = (strncmp( bin, "ON", 2 ) == 0) ? 1 : 0;

/*
 * plot the data
 */
PCDATA( stype, ltype, binmod, xdata, ydata, y_off, nrprow );

/*
 * draw some additional information
 */
SCTPUT( "Go with the cursor to plot info about the frame (left mouse button)");
stat = PCGCUR( &xpos, &ypos, &key );
if ( key != KEY_EXIT )
   { sprintf( text, "Frame: %s", name );
     PCTEXT( text, xpos, ypos, 0.0, 0.75, 0 );
   }
SCTPUT( "Go with the cursor to plot info about the drawn row" );
stat = PCGCUR( &xpos, &ypos, &key );
if ( key != KEY_EXIT )
```

```
   { sprintf( text, "Row: %.0f", image[2] );
     PCTEXT( text, xpos, ypos, 0.0, 0.75, 0 );
   }
/*
 * close plot file and terminate graphic operations
 */
PCCLOS();

return SCSEPI();
}
```

F.4 Examples Using the Communication Interfaces

F.4.1 An interactive Midas client (written in C)

The following module `myclient` is an interactive Midas client which connects/disconnects to/from a Midas server and sends user provided commands to the Midas server.

```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.COPYRIGHT    (c)  1994   European Southern Observatory
.IDENT        myclient.c
.LANGUAGE     C
.AUTHOR       K. Banse   ESO - Garching
.KEYWORDS     Midas client, communication
.PURPOSE      Communicate with a Midas server (background Midas) which is
              running on the same or different host
.VERSION      1.00  941020:   First version
--------------------------------------------------------------------*/


main()

{
int         id, stat, n;
int         noelm, retstat[2], kk, jj, iwa[3], ibuf[80];
int         felem, maxvals, actvals, unit, nullo, kstat;
register int nr;

float       rwa[3], rbuf[80];

double      dwa[3], dbuf[80];

char        metho, munit[4], kline[120], cbuf[80], remo[64];



id = -1;

input_loop:
if (id == -1)
   {
   printf("Options are:\n");
   printf("os XY = open socket communication with Midas unit XY\n");
```

```
   printf("of XY = open ASCII file based comm. with Midas unit XY\n");
   printf("is XY = start Midas unit XY + open socket communication\n");
   printf("if XY = start Midas unit XY + open ASCII file based comm.\n");
   printf("h = display help text, q = quit the test program\n");
   }

printf("option > ");
gets(kline);


while (kline[0] == ' ')
   strcpy(kline,&kline[1]);

if (kline[0] == '\0') /* empty line */
   goto input_loop;

else if (kline[0] == 'q')
   return(0);

else if (kline[0] == 'h')
   {
   printf("Options are:\n");
   printf("h = display help text, q = quit the test program\n");
   printf("c `secs' = send Midas command + wait for `secs'\n");
   printf("w = wait for last command, s = get status\n");
   printf
("x = close client connection, xx = as `x' but also terminate Midas server\n");
   }

else if (kline[0] == 'i')
   {
   munit[0] = kline[3];
   munit[1] = kline[4];

   if (kline[1] == 's')                  /* sockets */
      {
      metho = 'X';
      stat = XCXINI(munit,"",10,&id);   /* wait max. 10 seconds */
      }
   else                                  /* files */
      {
      metho = 'Z';
      stat = XCZINI(munit,"QX","",10,&id);
      }
```

```
   printf("XC%cINI, status = %d\n",metho,stat);
   }

else if (kline[0] == 'o') /* test XCXOPN or XCZOPN */
   {
   munit[0] = kline[3];
   munit[1] = kline[4];
   if (kline[1] == 's') /* sockets */
      {
      metho = 'X';
      stat = XCXOPN(munit,"",&id);
      }
   else /* files */
      {
      metho = 'Z';
      stat = XCZOPN(munit,"QX","",&id);
      }

   printf("XC%cOPN, status = %d\n",metho,stat);
   }

else if (kline[0] == 'x') /* test XCXCLO or XCXSTP */
   {
   if (kline[1] == 'x')                 /* also shut down server */
      {
      stat = XCXSTP(id);
      printf("XCXSTP, status = %d\n",stat);
      }
   else
      {
      stat = XCXCLO(id);
      printf("XCXCLO, status = %d\n",stat);
      }

   id = -1;
   }

else if (kline[0] == 'c') /* test XCCSND */
   {
   n = CGN_CNVT(&kline[2],1,1,iwa,rwa,dwa); /* Midas conv. routine */
   if (n<1) iwa[0] = -1; /* that means No Wait */

   printf("Enter Midas command: ");
```

```
   gets(kline);
   stat = XCCSND(id,kline,iwa[0],retstat);

   if (stat != 0)
      printf("XCCSND, status = %d\n",stat);
   else
      printf("background Midas returned status = %d\n",retstat[0]);
   }

else if (kline[0] == 'w') /* test XCCWAI */
   {
   stat = XCCWAI(id,retstat);

   if (stat != 0)
      printf("XCCWAI, status = %d\n",stat);
   else
      printf("background Midas returned status = %d\n",retstat[0]);
   }

else if (kline[0] == 's') /* test XCXSTA */
   {
   stat = XCXSTA(id);

   if (stat == 0)
      printf("background Midas %c%c idle\n",munit[0],munit[1]);
   else if (stat == 1)
      printf("background Midas %c%c busy\n",munit[0],munit[1]);
   else
      printf("XCXSTA, status = %d\n",stat);
   }

else
   printf("unknown command...\n");

goto input_loop; /* get next input */

}
```

Appendix G

MIDAS Help files

Each command in MIDAS must have it own help-file which can be used by the on-line help facility. The name of the file is constructed from the command name and qualifier. The name is in lower case and consists of the first six charachers of the command name with the first four characters of the qualifier appended. If the command name is shorter than six characters an underscore '_' is inserted to separate the two fields. The help files have the extension '.hlq'. It contains a number of sections as shown in Table G.1.

This files will be processed by TEX using a special macro. Thus, care should be taken that the text is compatible with TEX. An example of a help file for the command `READ/DESC` is given in Figure G.1.

```
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%.COPYRIGHT  (c)  1990 European Southern Observatory
%.IDENT      read_desc.hlq
%.AUTHOR     KB, IPG/ESO
%.KEYWORDS   MIDAS, help files, READ/DESCR
%.PURPOSE    On-line help file for the command: READ/DESCR
%.VERSION    1.0  13-APR-1984 : Creation, KB
%---------------------------------------------------------------
\se
SECTION./DESC
\es\co
READ/DESCR 13-APR-1984  KB
\oc\su
READ/DESCR frame [descr_list] [disp_flag]
                                                    display descriptor values
\us\pu
Purpose:
            Display descriptor values.
\up\sb
Subject:
            Descriptors.
\bs\sy
Syntax:
            READ/DESCR frame [descr_list] [disp_flag]
\ys\pa
            frame = name of data frame holding descriptor;
                    default file type is .bdf; so if you want to see the
                    descriptors of a table, you have to append also the
                    type to the filename
\ap\pa
            descr_list = single descriptor name or several descriptors
                    separated by a comma; defaulted to the standard
                    descriptors
\ap\pa
            disp_flag = B(rief) for short, F(ull) for extensive display of
                    descriptor; defaulted to B
\ap\sa
See also:
            WRITE/DESCR, SHOW/DESCR, PRINT/DESCR, DELETE/DESCR, COPY/DD
\as\no
Note:
            If descr_list = '*', all descriptors of the frame are displayed.
\on\xs
Examples:
\ex
            READ/DESCR durazno STEP
            Display contents of descriptor STEP of frame durazno.bdf .
\xe\ex
            READ/DESCR durazno start,step full
            Display descriptors STEP and START of frame durazno.bdf in
            extended format.
\xe\ex
            READ/DESCR naranja.tbl *
            Display all descriptors of table file naranja.tbl .
\xe \sx
```

Figure G.1: example of helpfile for READ/DESC command

Help section	Start label	End label	Contents
Section	`\se`	`\es`	Command qualifier section
Command	`\co`	`\oc`	Command name and date
Purpose	`\pu`	`\up`	General purpose of command
Subject	`\sb`	`\bs`	Subject keywords
Short help	`\su`	`\us`	Short command description
Syntex	`\sy`	`\ys`	Command syntax
Parameter	`\pa`	`\ap`	Parameter item
Note	`\no`	`\on`	General notes
See also	`\sa`	`\as`	Command cross references
Outputs	`\op`	`\po`	Command outputs
Restrictions	`\rs`	`\sr`	Command restrictions
Examples section	`\xs`	`\sx`	Example section
Examples item	`\ex`	`\xe`	Example item

Table G.1: Sections in help-files.

Index