ESO-MIDAS User Guide

Volume A: System

MIDAS Release **98NOV**

Reference Number: MID-MAN-ESO-11000-0002

Section	Title	Date
Chapter 1	Introduction	31-March-1999
Chapter 2	Cook-Book	31-March-1999
Chapter 3	Monitor & Syntax	31-March-1999
Chapter 4	Data Structures	NA
Chapter 5	Table File System	1-November-1992
Chapter 6	Graphics & Image Display	31-March-1999
Chapter 7	Data Exchange Format	31-March-1999
Chapter 8	Fitting of Data	15-January-1988
Appendix A	Command Summary	31-March-1999
Appendix B	Acknowledgements	1-November-1995
Appendix C	Site Specific Implementation	31-March-1999
Appendix D	Release Notes	31-March-1999

EUROPEAN SOUTHERN OBSERVATORY
Data Management Division
Karl–Schwarzschild–Straße 2, D–85748 Garching bei München
Federal Republic of Germany

Contents

List of Figures

List of Tables

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You can copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING

BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How To Apply These Terms To Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and an idea of what it does.

```
Copyright (C) 19yy name of author

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type 'show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type 'show c'
for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program 'Gnomovision'
(which makes passes at compilers) written
by James Hacker.
```

signature of Ty Coon, 1 April 1989

```
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Chapter 1

Introduction

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 MIDAS system provides general tools for image processing and data reduction with emphasis on astronomical applications including special reduction packages for ESO instruments at La Silla. The system is available for both VAX/VMS and UNIX systems.

A large number of contributions have been made to MIDAS by people inside and outside ESO. We greatly appreciate and acknowledge these efforts. A full list of acknowledgements can be found in Appendix B.

This manual gives the necessary information to do useful data reduction with the system, whereas a detailed technical description of the design and software interfaces can be found in other documents (see Section 1.6). These documents also describe how users can write and add their own application programs to the system.

1.1 How to use the MIDAS Manual

This document is intended to be a description of how to use the various facilities available in the MIDAS system. The manual consists of three volumes:

Volume A: describes the basic MIDAS system with all general purpose facilities such as MIDAS Control Language, data input/output (including graphics and image display), table system (MIDAS Data Base). A summary of all available commands as well as site specific features are given in appendices.

Volume B: describes how to use the MIDAS system for astronomical data reduction. Application packages for special types of data or reductions (*e.g.* long slit and echelle spectra, object search, or crowded field photometry) are discussed assuming intensity calibrated data. A set of appendices gives a detailed description of the reduction of raw data from ESO instruments.

[1] Trade-mark of the European Southern Observatory

Volume C: gives the detailed description for all commands available.

It is intended that users will mainly need Volume A for general reference. For specific reduction of raw data and usage of special astronomical packages, Volume B will be more informative. A printed version of the MIDAS help files is available in Volume C. Users are recommended to use the on-line help facility which always gives a full up to date description of the commands available.

1.1.1 New Users

To be able to use MIDAS, it is a great advantage to have some basic knowledge of computer systems such as how to login, use of the file editor and simple system commands. Such instructions can normally be found in local system documentation or in Appendix C of this volume. After having acquired this knowledge, new users should read Chapter 2 carefully. This will give a basic introduction to the MIDAS system with some examples.

1.1.2 Site Specific Features

MIDAS is used at many different sites on a large variety of configurations. The main part of this manual does not refer to special configurations or hardware devices. Site specific implementations and details of the local installation can be found in Appendix C.

1.2 General Concept of MIDAS

The MIDAS system is built along lines which should allow easy integration of complex analysis algorithms as well as allowing greater flexibility in interactive use and in the creation of user specific procedures from the basic building blocks. The first design proposal for MIDAS, made late 1980, used some ideas from the UK STARLINK project for the software interface definitions. The present version which became available in 1984 follows a similar philosophy in its application program interfaces, but has been expanded to the new Standard Interfaces which have a broader base than previously.

MIDAS has benefitted greatly from the experience gained at ESO using the Hewlett-Packard based image processing system IHAP (see F. Middelburg, IHAP Manual, ESO 1985). Not only have many of the internal design features such as “world coordinates” been incorporated, but also the command language has been designed in such a way that it is similar to the basic philosophy of IHAP.

The MIDAS system can be run in both an interactive and a batch mode. In addition, the interactive user will be able to create batch jobs which will run in parallel with the interactive work.

MIDAS is based on three sets of general interfaces for application programs to data structures, namely: a) the “Standard Interfaces” for general I/O and image access, b) the “Table Interfaces” for access to table structures, and the “Graphics Interfaces” for data representation in graphical format. These interfaces allow easy integration of application programs into MIDAS. To provide a portable system a layer of OS-routines have been used to shield MIDAS from the local operating system. These routines may only be used at lowest levels and are not available for normal applications.

To facilitate easy implementation of different graphic and display devices, MIDAS has adopted a set of device independent interfaces for plotting and image display. All plotting routines in MIDAS are based on the **A**STRONET **G**raphic **L**ibrary, developed and maintained by the italian ASTRONET. Further, the image display applications are using the **I**mage **D**isplay **I**nterface routines defined in collaboration with ST ScI, UK STARLINK, KPNO and Trieste Observatory.

1.3 Distribution Policy

As of the 95NOV release, patch level 2.0, ESO-MIDAS is available under the GNU General Public License. The short statement of the GNU license is included in every MIDAS file; the full text is printed in the beginning of this manual and is included as a separate file in the ESO-MIDAS Release. Terms and conditions for copying, distribution, and modification can be displayed by the command `license c`. The command `lecense w` displays details cocerning warranty.

All source code, documentation, as well as binary copies of ESO-MIDAS (already installed and without sources) for specific systems can be retrieved from the ESO-MIDAS anonymous ftp account. Manual pages, installation guides and other documentation are generally released together with the source code and can also be obtained from this ftp account. In some exceptional cases the release of the documentation can be delayed slightly. Only in cases where no possibility exist to retrieve ESO-MIDAS via the network ESO may send the release on magnetic or optical media. Requests for this service, which must include the format of the medium, can be sent to the Data Management Division at ESO/Munich.

As with the main release, patches, new applications, updates of the documentation, etc. will be made available on the above mentioned anonymous ftp account. They will announced via the various communication channels to the User's community and described in the ESO-MIDAS World Wide Web page on Support and Information Services. MIDAS sides are also recommended to check for the existence of new paches in the relevant README file on our anonymous ftp account on a regular basis.

1.4 Support

Most of our support services are available via the World Wide Web Homepage `http://www.eso.org/esom`

The MIDAS Users Support provides help for users that have problems or questions during installation or usage of the MIDAS system. It answers to questions cannot be obtained locally (e.g. through a manual) or after consulting our FAQ, please feel free to contact us for support. The most convinient and best way to contact us is using the **ESO-MIDAS Problem Report Form** that enable us to deal the reporting quickly using a set of processing tools. In addition this way of reporting offers us the possibility of maintaining a database of the questions and the update the FAQ listing.

The ESO-MIDAS Problem Form is available via the WWW and via the XHELP Graphical User Interface when running MIDAS. To start this interface enter: CREATE/GUI HELP. In case the ESO-MIDAS Problem Report System cannot be used the MIDAS User Support can be contacted via:

- `internet: midas@eso.org`
- `telefax: +49 89 32006480 (attn.: MIDAS SUPPORT)`

Requests and questions are acknowledged when received and processed as soon as possible, normally within a few days. Also, users are strongly encouraged to send suggestions and comments via the MIDAS Problem Report Form that enables quick processing of your request.

1.5 Requirements for Running MIDAS

MIDAS can run on computers with either UNIX[2] or VAX/VMS[3] operating systems. Details of hardware and software requirements for MIDAS are listed below.

1.5.1 Hardware

- Computer system: any system which can run either VAX/VMS or UNIX operating systems. MIDAS implementations have been made on a large number of systems from vendors e.g. VAX stations, DEC stations, SUN SPARCstations, HP 700 series, IBM RS/6000 systems, PCs. The availability for a specific system can be checked by asking the MIDAS Support.

- Memory: depending on the number of users of the systems but normally at least 8 Mbyte. A physical memory that is too small may significantly reduce the performance due to swapping of data to disk.

- Disk: the full MIDAS system requires of the order of 100 Mbyte of disk storage depending on the type of CPU. During installation an extra 10 Mbyte should be available for temporary files such as object code. The size of the system can be reduced in three ways: a) source files can be deleted after implementation, b) help–files can be removed if on–line help is not required, and c) parts of the system which are not used (*e.g.* crowded field photometry or echelle packages) need not be loaded. Implementations using shared libraries are available on some systems (*e.g.* SPARCstations and PCs). This reduces the needed disk space significantly.

  A typical disk size for a single user system is approximately 200 Mbyte assuming 60 Mbyte for the operation system, 80 Mbyte for MIDAS, and 50-100 Mbyte for a user with 2–dimensional data.

[2]Trademark of AT& T

[3]Trademarks of Digital Equipment Corporation

- Terminals: any alpha-numeric terminal can be used. MIDAS can either work in a simple line-by-line mode or provide special features such as line editing by using a terminal definition file for special terminal features.
- Graphic display: the graphics software of MIDAS uses the device independent plotting library AGL [4] made by the italian ASTRONET. Drivers for a significant number of different devices are available *e.g.* X Window System, PostScript, aand various types of graphics terminals (see a complete list in Chapter 6). It is possible to write drivers for devices for which none exist (see the AGL driver manual).
- Image displays: MIDAS uses the **I**mage **D**isplay **I**nterfaces which provides a general interface to image display devices. The X Window System running under different window managers is fully supported. Other devices can be used if an appropriate set of IDI-routines are written. A list of currently available IDI-routines can be obtained from the MIDAS Support Group.

1.5.2 Software

- System: UNIX or VAX/VMS operating systems.
- Compilers: C- and FORTRAN-77 compilers (a FORTRAN-to-C convertion package may be used instead of a FORTRAN-77 compiler).
- Libraries: AGL is used for all plotting in MIDAS. This library is normally available on the release media but can also be obtained from the Italian ASTRONET (free of charge for non-profit research institutes).

  NAG[5] is used in a few packages such as fitting. It is under license and can be purchased from the Numerical Algorithms Group. MIDAS can be installed without this library in which case some commands will be unavailable.

1.6 Other Relevant Documents

There are several other documents relevant to the MIDAS system. General descriptions of the system can be found in the following references:

- Banse, K., Crane, P. Ounnas, C., Ponz, D., 1983 : 'MIDAS' in *Proc. of DECUS*, Zurich, p.87
- Grosbøl, P., Ponz, D. , 1985 : 'The MIDAS Table File System', *Mem.S.A.It.* **56**, p.429
- Banse, K., Grosbøl, P., Ponz, D., Ounnas, C., Warmels, R., 'The MIDAS Image Processing System in Instrumentation for Ground Based Astronomy: Present and Future, L.B. Robinson, ed., New York: Springer Verlag, p.431.

[4]Astronet Graphica Library made by the Italian ASTRONET
[5]NAG made by Numerical Algorithms Group

- Grosbøl P., Banse, K., Guirao, C, Ponz, J.D., Warmels, R.H.: 1988, 'MIDAS Benchmarks of Workstations' in ESO Messenger, ESO, Garching, 54 , 59
- Grosbøl, P.J., Ponz, J.D.: 1990, 'The MIDAS System' in Acquisition, Processing and Archiving of Astronomical Images,, G. Longo and G. Sedmak (eds.), OAC and FORMEZ, 1990, p.109
- Warmels, R.H.: 1991, 'The ESO-MIDAS System', in Astronomical Data Analysis Software and Systems I , *PASP Conf. Series*, Vol. 25, p. 115.

For general bibliographic reference to the MIDAS system (VAX/VMS version), the first reference in the above list should be used.

Detailed technical information of software interfaces and designs used in MIDAS is given in the following documentation:

- MIDAS Environment Document
- MIDAS IDI–routines
- AGL Reference Manual

Users who want to write their own application programs for MIDAS should read the MIDAS Environment document which gives the relevant information and examples.

For users who have to work with both the IHAP and MIDAS systems a cross-reference document has been made for the most commonly used commands:

- MIDAS-IHAP/IHAP-MIDAS Cross-Reference

All documents, either in postscript or in ascii format, are available via the MIDAS anonymous ftp acccount.

Chapter 2

Cook-Book

This chapter outlines the necessary information to get started with the MIDAS system. Further details can be found in the appropriate sections of the following chapters.

The essential steps are:

- To login to the computer you want to use.
- Start up the MIDAS monitor.
- Load some data from tape or via `ftp` into your disk space.
- Execute the MIDAS commands you want.
- Save processed data on tape.
- Exit from the MIDAS monitor and logout.

These steps are outlined in the following sections.

2.1 Terminology

The following explain various terms used in this manual.

Keywords — global variables in the MIDAS monitor. They can be single numbers or characters or one dimensional arrays used to store input, output, or control information for MIDAS commands.

Frames — arrays of numbers representing data values with uniform sampling. They are used for storage of images or spectra.

Images — used interchangeably with frames.

Descriptors — variables associated to frames, tables or masks describing the contents in them. They are basically the same as keywords just associated to data files instead of the monitor. These descriptors have names like NAXIS (the dimension of the image array), CUNIT (the units of the axes), etc.

Tables — two dimensional arrays organised with rows and columns. They are used for storage of heterogeneous data contrary to frames and masks which store homogeneous data. They are typically used for saving lists of e.g. stellar positions and magnitudes. See Chapter 5 for a full description of the table facilities. Many commands output their results or take their input from tables. They constitute a simple data base system for MIDAS.

Catalogues — a list of frames, tables, or masks which can be used for input to various commands or merely for reference.

Procedures — These are lists of MIDAS commands stored in a file of type *filename*.prg and which can be executed by typing `@@` *filename*. See Chapter 3 for further details.

Fit file — a file that contains the function and parameter values for use in conjunction with the fitting commands described in Chapter 8.

2.2 Commands

MIDAS is a command driven system in which the user enters commands followed by parameters. This implies that the user must know a few commands and their structure in order to make effective use of the system. Since most users cannot keep all the commands and their parameters at their finger tips, an extensive on–line help facility has been created which can be used within the Midas terminal window as well as via a graphical user interface.
A MIDAS command has the following structure:

```
COMMAND/QUALIFIER par1 par2 ... par8
```

where `par1` is the first parameter and so on. The important points are:

- Command and qualifier are separated by a `/` (slash).
- The command/qualifier and the parameters are separated by a *space*.
- Most commands have qualifiers.
- A parameter may contain several sub-parameters which are separated by commas.
- In most cases if the parameters are not specified, the system makes sensible defaults, but the user should **not** always trust these default values to be those he might have chosen.
- Keep these rules in mind, otherwise you will confuse the command.

MIDAS commands divide themselves into three categories:

- MIDAS basic (system) commands

- MIDAS application commands
- procedure control commands

All MIDAS commands are listed in alphabetical order and explained in detail in help graphical user interface of MIDAS or can be found on the home page of ESO Midas (web address: http://www.eso.org/projects/esomidas/). The application commands are developed for special purposes such as CCD or CASPEC reductions. They are described and listed in the various sections related to particular applications. For reductions of data you should refer to Volume B of this manual. The commands for Midas procedures are described in Chapter 3.

2.3 Getting Started

To start the MIDAS system:

- Type `inmidas` on a UNIX (Linux) system (`INMIDAS` on a VMS system) This will setup the MIDAS environment and the terminal should respond with the Midas prompt:

  ```
  Midas 001>
  ```

- The available commands can be listed out by typing `HELP`. This only gives you the names of the commands presently available in the system. A summary and a subject grouped listing of the commands you will find in Appendix A. A command can have several qualifiers which will change the mode of execution of the command e.g `STATISTICS` has qualifiers `IMAGE`, `TABLE` and `POISSON`. It is possible, typing `HELP` *command*, to get a display of all the *command/qualifier* combinations available for the given *command*. Detailed information about a specific *command/qualifier* combination can be obtained by typing `HELP` *command/qualifier*. It is however much more convenient to start up the GUI for the Midas help facility with the command `CREATE/GUI HELP` and obtain help via pointing and clicking with the mouse.
- In some installations a number of tutorial commands are available. (see Table 2.1) They provide an illustration of different parts of the system.

Note

MIDAS is case insensitive, i.e. it does not matter if you enter MIDAS commands in upper or lower case characters. It's just for readability that we write all MIDAS commands in this chapter in upper case.

2.3.1 Simple MIDAS Session

This section gives two examples of simple MIDAS sessions. The first one reads some frames from a magnetic tape, displays them on a monitor and performs some simple operations. The second one creates MIDAS tables and performs some simple operations. In the following examples, user input is written in bold face type while comments, (after an exclamation mark) are written in normal roman font.

```
$ inmidas
Midas 001> HELP INTAPE                                  !get help for tape input
Midas 002> INTAPE * x TAPE1 FNN                         !lists headers of all files
                                                        !from tape mounted on TAPE1
Midas 003> INTAPE 2,16-17,31-33,52 CCD TAPE1            !read files from TAPE1

Image ccd0002          : FF D V  20S      , naxis:  2, pixels:      337,    520
Image ccd0016          : DK BIAS 1S       , naxis:  2, pixels:      337,    520
Image ccd0017          : DK BIAS 1S       , naxis:  2, pixels:      337,    520
Image ccd0031          : DK      60S      , naxis:  2, pixels:      337,    520
Image ccd0032          : DK      60S      , naxis:  2, pixels:      337,    520
Image ccd0033          : DK      60S      , naxis:  2, pixels:      337,    520
Image ccd0052          : A0532-527 V  300 , naxis:  2, pixels:      337,    520

Midas 004> INTAPE 78 ccd image.fits                     !read fits file from disk

Image ccd0078          : A1029-459 V  40S , naxis:  2, pixels:      337,    520

Midas 005> CREATE/ICAT                                  !create a catalogue of images
Image catalog icatalog.cat with 8 entries created...
Midas 006> SET/ICAT                                     !enable catalogue
Midas 007> READ/ICAT                                    !list the catalogue out
Image Catalog:  icatalog.cat

No    Name            Ident                                   Naxis   Npix(1,2)
#0001 ccd0002.bdf     FF D V  20S                                 2   337    520
#0002 ccd0016.bdf     DK BIAS 1S                                  2   337    520
#0003 ccd0017.bdf     DK BIAS 1S                                  2   337    520
#0004 ccd0031.bdf     DK      60S                                 2   337    520
#0005 ccd0032.bdf     DK      60S                                 2   337    520
#0006 ccd0033.bdf     DK      60S                                 2   337    520
#0007 ccd0052.bdf     A0532-527 V  300                            2   337    520
#0008 ccd0078.bdf     A1029-459 V  40S                            2   337    520

Midas 008> STAT/IMA ccd0016                    !compute statistics for this frame

frame: ccd0016   (data = R4)
complete area of frame
minimum, maximum:              184.0000       16383.00
at (   215,     2),(   337,   520)
mean, standard_deviation:      251.2245       880.0140
3rd + 4th moment:             0.1305419E+11  0.2137182E+15
total intensity:              0.4402457E+08
median, 1. mode:               16287.41       16287.71
total no. of bins, binsize:         256       63.52549
# of pixels used =   175240 or 100.00 % of all possible pixels (=   175240)
from     1     1 to   337   520 (in pixels)

Midas 009> STAT/IMA ccd0017

frame: ccd0017   (data = R4)
complete area of frame
```

```
minimum, maximum:                187.0000       16383.00
at (   113,     1),(   337,   520)
mean, standard_deviation:        251.2514       880.0129
3rd + 4th moment:               0.1305419E+11  0.2137182E+15
total intensity:                0.4402929E+08
median, 1. mode:                 16287.30       16287.73
total no. of bins, binsize:          256       63.51373
# of pixels used =   175240 or 100.00 % of all possible pixels (=   175240)
from     1     1 to    337   520 (in pixels)
```

Midas 010> **CREATE/DISPLAY** !create a display window
Midas 011> **LOAD/IMA ccd0017** !display image
Midas 012> **GET/CURS** !read some pixels values from the image

```
cursor #0
frame pixels          world coords       intensity
frame: ccd0017
   334.0   166.0     334.000     166.000      228.000
```

Midas 013> **EXTRACT/IMA ff = ccd0002**[<,<:@330,>] !remove
!irrelevant columns
Midas 014> **EXTRACT/IMA biai1 = ccd0016**[<,<:@330,>]
Midas 015> **EXTRACT/IMA biai2 = ccd0017**[<,<:@330,>]
Midas 016> **EXTRACT/IMA dk1 = ccd0031**[<,<:@330,>]
Midas 017> **EXTRACT/IMA dk2 = ccd0032**[<,<:@330,>]
Midas 018> **EXTRACT/IMA dk3 = ccd0033**[<,<:@330,>]
Midas 019> **EXTRACT/IMA ima1 = ccd0052**[<,<:@330,>]
Midas 020> **EXTRACT/IMA ima2 = ccd0078**[<,<:@330,>]
Midas 021> **STAT/IMA biai1**

```
frame: biai1    (data = R4)
complete area of frame
minimum, maximum:                184.0000       497.0000
at (   215,     2),(   147,   408)
mean, standard_deviation:        203.2299       3.752572
3rd + 4th moment:                8402620.     0.1709546E+10
total intensity:                0.3487425E+08
median, 1. mode:                 202.0980       204.2529
total no. of bins, binsize:          256       1.227451
# of pixels used =   171600 or 100.00 % of all possible pixels (=   171600)
from     1     1 to    330   520 (in pixels)
```

Midas 022> **STAT/IMA biai2**

```
frame: biai2    (data = R4)
complete area of frame
minimum, maximum:                187.0000       613.0000
at (   113,     1),(   286,   473)
mean, standard\_deviation:        203.2554       3.836732
3rd + 4th moment:                8406459.     0.1710918E+10
total intensity:                0.3487864E+08
```

```
median, 1. mode:                201.6387      202.8706
total no. of bins, binsize:          256      1.670588
# of pixels used =   171600 or 100.00 % of all possible pixels (=   171600)
from     1     1 to   330   520 (in pixels)
```

Midas 023> **READ/DESCR biai1 STATISTIC** !read descriptor statistic

```
frame: BIAI1   (data = R4)
STATISTIC      :
   184.0000      497.0000      203.2299      3.752572      8402620.
  0.1709546E+10  202.0980      204.2529      256.0000      1.227451
  0.3487425E+08
```

Midas 024> **COMPUTE/IMA ffb = ff-203.** !biais correction
Midas 025> **COMPUTE/IMA dk1b = dk1-203.**
Midas 026> **COMPUTE/IMA dk2b = dk2-203.**
Midas 027> **COMPUTE/IMA dk3b = dk3-203.**
Midas 028> **COMPUTE/IMA ima1b = ima1-203.**
Midas 029> **COMPUTE/IMA ima2b = ima2-203.**
Midas 030> **AVERAGE/IMA dk = dk1b,dk2b,dk3b** !compute an
!average of DARK frames

```
dk1b processed ...
dk2b processed ...
dk3b processed ...
```

Midas 031> **COMPUTE/IMA ffd = ff-dk** !dark subtraction
Midas 032> **COMPUTE/IMA ima1bd = ima1b-dk**
Midas 033> **COMPUTE/IMA ima2bd = ima2b-dk**
Midas 034> **COMPUTE/IMA ima1bdf = ima1bd/ffd** !flat-field the frame
Midas 035> **COMPUTE/IMA ima2bdf = ima2bd/ffd**
Midas 036> **LOAD/IMA ima1bdf**
Midas 037> **READ/DESCR ima1bdf LHCUTS**

```
frame: ima1bdf   (data = R4)
LHCUTS         :
  0.0000000E+00  0.0000000E+00 -0.7500000E-01   1.992324      0.0000000E+00
  0.0000000E+00
```

Midas 038> **CUTS ima1bdf 0.,1.992** !modify display cuts
Midas 039> **LOAD ima1bdf**
Midas 040> **READ/ICAT icatalog**

```
Image Catalog:  icatalog.cat

No    Name           Ident                                    Naxis   Npix(1,2)
#0001 ccd0002.bdf    FF D V  20S                                  2   337   520
#0002 ccd0016.bdf    DK BIAS 1S                                   2   337   520
#0003 ccd0017.bdf    DK BIAS 1S                                   2   337   520
#0004 ccd0031.bdf    DK      60S                                  2   337   520
#0005 ccd0032.bdf    DK      60S                                  2   337   520
#0006 ccd0033.bdf    DK      60S                                  2   337   520
#0007 ccd0052.bdf    A0532-527 V  300                             2   337   520
#0008 ccd0078.bdf    A1029-459 V  40S                             2   337   520
```

```
#0009 ff            FF D V  20S                                           2    330    520
#0010 biai1         DK BIAS 1S                                            2    330    520
#0011 biai2         DK BIAS 1S                                            2    330    520
#0012 dk1           DK      60S                                           2    330    520
#0013 dk2           DK      60S                                           2    330    520
#0014 dk3           DK      60S                                           2    330    520
#0015 ima1          A0532-527 V  300                                      2    330    520
#0016 ima2          A1029-459 V  40S                                      2    330    520
#0017 ffb           FF D V  20S                                           2    330    520
#0018 dk1b          DK      60S                                           2    330    520
#0019 dk2b          DK      60S                                           2    330    520
#0020 dk3b          DK      60S                                           2    330    520
#0021 ima1b         A0532-527 V  300                                      2    330    520
#0022 ima2b         A1029-459 V  40S                                      2    330    520
#0023 dk            average frame                                         2    330    520
#0024 ffd           FF D V  20S                                           2    330    520
#0025 ima1bd        A0532-527 V  300                                      2    330    520
#0026 ima2bd        A1029-459 V  40S                                      2    330    520
#0027 ima1bdf       A0532-527 V  300                                      2    330    520
#0028 ima2bdf       A1029-459 V  40S                                      2    330    520
```

Midas 041> **OUTTAPE icatalog,9-28 TAPE1** !save data on tape

```
File ff.bdf         written to tape with   241 blocks
File biai1.bdf      written to tape with   243 blocks
File biai2.bdf      written to tape with   243 blocks
File dk1.bdf        written to tape with   241 blocks
File dk2.bdf        written to tape with   241 blocks
File dk3.bdf        written to tape with   241 blocks
File ima1.bdf       written to tape with   241 blocks
File ima2.bdf       written to tape with   241 blocks
File ffb.bdf        written to tape with   241 blocks
File dk1b.bdf       written to tape with   241 blocks
File dk2b.bdf       written to tape with   241 blocks
File dk3b.bdf       written to tape with   241 blocks
File ima1b.bdf      written to tape with   241 blocks
File ima2b.bdf      written to tape with   241 blocks
File dk.bdf         written to tape with   241 blocks
File ffd.bdf        written to tape with   241 blocks
File ima1bd.bdf     written to tape with   241 blocks
File ima2bd.bdf     written to tape with   241 blocks
File ima1bdf.bdf    written to tape with   241 blocks
File ima2bdf.bdf    written to tape with   241 blocks
```

Midas 042> **OUTTAPE icatalog,7-8 ima** !save data on disk in FITS format

```
File ima1.bdf       written to disk> ima0001.mt
File ima2.bdf       written to disk> ima0002.mt
```

Midas 043> **PRINT/LOG** !print logfile on the default printer
Midas 042> **BYE**

Note

As you can see e.g. from Midas command #8, you can abbreviate commands and qualifiers (this is, however, not recommended when writing Midas procedures). Furthermore, you can omit qualifiers as in command #39, then the default qualifier of the command is taken. The command `SHOW/COMMAND Midas_command` *displays the default qualifier of command* `Midas_command`.

$ inmidas
`Midas 001>` **CREATE/TABLE flux1 2 30 flux1** ! create table from ASCII file
`Midas 002>` **SHOW/TABLE flux1** ! list table parameters

```
 Table : flux1                                  [Transposed format]
 No.Columns :        2    No.Rows    :        30
 All.Columns:        3    All.Rows   :        32
 Sorted by  Sequence    Reference  :  Sequence
 Col.#   1:LAB001          Unit:Unitless        Format:E15.6  R*4
 Col.#   2:LAB002          Unit:Unitless        Format:E15.6  R*4
 Selection : ALL
```

`Midas 003>` **READ/TABLE flux1 :LAB001 :LAB002 @1 @9** !read data from table

```
  Table : flux1

 Sequence LAB001          LAB002
 -------- --------------- ---------------
        1     4.71000e+03     1.12850e+04
        2     4.77000e+03     1.08300e+04
        3     5.05000e+03     8.77000e+03
        4     5.09000e+03     8.55000e+03
        5     5.12000e+03     8.34000e+03
        6     5.19000e+03     7.93000e+03
        7     5.23500e+03     7.73500e+03
        8     5.26500e+03     7.55000e+03
        9     5.39000e+03     7.00500e+03
 -------- --------------- ---------------
```

`Midas 004>` **NAME/COLUMN flux1 :LAB001 "(sec)" F6.0** !change ! format and define unit
`Midas 005>` **NAME/COLUMN flux1 :LAB002 "(erg/sec)" F6.0**
`Midas 006>` **READ/DESCR flux1.tbl HISTORY**

```
HISTORY         :
CREA/TABL flux1 2 30 flux1 NULL TRAN
NAME/COLU flux1 :LAB001 "(sec)" F6.0
NAME/COLU flux1 :LAB002 "(erg/sec)" F6.0
```

`Midas 007>` **STAT/TABLE flux1 :LAB001** !compute statistics of a column

```
Table : flux1
Column #   1 Label :LAB001          Type :R*4
Total no. of entries :      30, selected no. of entries :      30
Miminum value :   0.47100E+04, Maximum value:   0.96800E+04
Mean value    :   0.66437E+04, Standard dev.:   0.15539E+04
```

`Midas 008>` **$more flux2.fmt**

```
!
! format file flux2.fmt
!
DEFINE/FIELD 1 6 R F6.0 :LAB001
DEFINE/FIELD 9 13 R F6.0 :LAB002
END
```

`Midas 009>` **CREATE/TABLE flux2 2 8 flux2 flux2**
`Midas 010>` **READ/TABLE flux2**

```
  Table : flux2

  Sequence LAB001 LAB002
  -------- ------ ------
         1   9705   2750
         2   9930   2700
         3  10130   2655
         4  10150   2650
         5  10520   2580
         6  10740   2540
         7  11040   2485
         8  11570   2410
  -------- ------ ------
```

`Midas 011>` **MERGE/TABLE flux1 flux2 flux** !merge of two tables
`Midas 012>` **SHOW/TABLE flux**

```
 Table : flux                           [Transposed format]
 No.Columns :        2    No.Rows    :        38
 All.Columns:        3    All.Rows   :        40
 Sorted by # Sequence     Reference  :  Sequence
 Col.#   1:LAB001            Unit:(sec)             Format:F6.0   R*4
 Col.#   2:LAB002            Unit:(erg/sec)         Format:F6.0   R*4
Selection : ALL
```

`Midas 013>` **SORT/TABLE flux :LAB002** !sort table according to !increasing values of a column
`Midas 014>` **REGRESSION/POLYNOMIAL flux :LAB001 :LAB002 5** !polynomial fit

```
flux
 POLYNOMIALS         Input Table :  UNION     Type : MUL L-S
 N.Cases     :      38 ; N.Ind.Vars   :   1
 Dependent variable  :   column #   1
```

```
 Independent variable:   column #   2       degree :   5

  degree
     0  4.4818E+04
     1 -2.6778E+01
     2  7.4671E-03
     3 -1.0405E-06
     4  7.1309E-11
     5 -1.9126E-15

          R.M.S error    :      72.04152
```

Midas 015> **SAVE/REGRESSION flux REGRE** !save the result of !regression in a descriptor
Midas 016> **CREATE/COLUMN flux :FIT** !create a new column !in the table
Warning: Column overflow mechanism activated
Midas 017> **COMPUTE/REGRESSION flux :FIT = REGRE** !compute !the results of the regression
Midas 018> **READ/TABLE flux :FIT @1..4,@6,@9**

```
  Table : flux

  Sequence       FIT
  --------   -------------
         1   1.1337231e+04
         2   1.0955642e+04
         3   1.0690593e+04
         4   1.0505399e+04
         6   1.0174703e+04
         9   9.7462227e+03
  --------   -------------
```

Midas 019> **CREATE/GRAPH**
!create graphic window
Midas 020> **PROJECT/TABLE flux newflux :LAB002** ! project one !column of a table in a new one
Midas 021> **INTERPOLATE/TT newflux :LAB002,:SPLINE flux :LAB002,:LAB001 0.001**
!spline interpolation
Midas 022> **CREATE/GRAPH**
!create graphic window
Midas 023> **PLOT/TABLE newflux :LAB002 :LAB001** !plot table columns
Midas 024> **SET/PLOT LTYPE=1 STYPE=0** !plot table columns
Midas 025> **OVERPLOT/TABLE flux :LAB002 :SPLINE**
Midas 026> **SELECT/TAB newflux sequence.gt.5** !select part of the table
Midas 027> **COPY/TAB newflux result** !copy selected table
Midas 028> **OUTTAPE result.tbl result.fits** !save file in FITS !format on disk
Midas 029> **BYE**

Command	Description
TUTORIAL/ALIGN	Explains the use of the ALIGN command
TUTORIAL/EXTRACT	Demonstrates the extraction of a subimage from a father image
TUTORIAL/FILTER	Displays some of the filtering options
TUTORIAL/FIT	Shows the fitting capabilities
TUTORIAL/HELP	Explains the usage of the HELP command
TUTORIAL/ITT	Shows the effect of various Image Transformation Tables on an image
TUTORIAL/PLOT	Demonstrates the plot package facilities
TUTORIAL/LUT	Shows the effect of various Look–Up Tables on an image
TUTORIAL/PLOT	Demonstrates the plot package facilities
TUTORIAL/SPLIT	Shows the split–screen capabilities of the display
TUTORIAL/TABLE	Demonstrates the table system

Table 2.1: List of Tutorials

2.3.2 Exit and Logout

To exit from the MIDAS monitor type `BYE`. You can reenter the MIDAS monitor at any time by typing `$gomidas` (`GOMIDAS` for VMS systems).

2.3.3 Executing System Commands

It is possible to execute commands of the operating system inside MIDAS, see e.g. command #8. This is done by typing a $ followed by the operating system command you want to have executed. After this command has been finished you can continue your work inside MIDAS. Since MIDAS executes its applications in a child process (subprocess for VMS) which leaves no traces after termination, you cannot simply use the host command `$ cd` (Unix) (`$ SET DEF` in VMS) to change the working directory once you are in a MIDAS session. Instead, use the MIDAS command `change/direc` for that purpose.
See Chapter 3 for a detailed discussion of the interaction of Midas with the host system.

Chapter 3

Monitor and Command Language

This chapter is organised as follows:

- In the first two sections we describe how to start MIDAS and how the host operating system and MIDAS coexist.
- Section 3 explains the different data structures used in MIDAS and how to access them in a MIDAS session.
- Section 4 describes the syntax of the MIDAS commands, as well as the editing and recalling of commands and also the on-line HELP facility in MIDAS.
- In section 5 you will find some details about how the MIDAS commands are executed.
- Then follows the largest and most detailed section (section 6), which gives in-depth information about the MIDAS command language, `MCL`.
  With `MCL` you can write high level MIDAS "programs" which are called MIDAS procedures.
  The topics include:
  - the `MCL` commands
  - passing parameters in MIDAS procedures
  - symbol substitution
  - loops and conditional branching
  - special functions
- Section 7 introduces the MIDAS contexts.
- Section 8 explains how to run application programs written in FORTRAN or C inside MIDAS. It also shows how to debug these programs as well as MIDAS procedures.
- All the commands related to MIDAS catalogs are listed in section 9, together with an example of how to use catalogs in MIDAS procedures.

- The MIDAS login procedure and MIDAS user levels are the topics of the last two sections.

The MIDAS directory tree structure is not covered in this chapter. For those interested, please refer to the *MIDAS Environment document.*

3.1 Starting the MIDAS Monitor

In order to get properly initialised, MIDAS needs the following information

- the MIDAS user mode to work in: *Parallel* or *Single User* mode, the default is *Single User* mode
- the MIDAS working directory for internal files and (optionally) private procedures: the default is a subdirectory `midwork` in your login directory, i.e. `SYS$LOGIN:[MIDWORK]` in VMS or `$HOME/midwork in Unix`. This directory does not have to be (or even should not be) the same directory where your data files are stored.

On a VMS system type `SETMIDAS` and follow the dialog if you want to change the defaults for the mode and the MIDAS work directory.
On a Unix/Linux system use the option *-m <mid_work>* to change the default working directory, and *-p* or *-P* to run MIDAS in parallel mode when you start MIDAS via the command `inmidas` (see below).

Two other variables are very important to MIDAS - `MIDVERS`, which holds the MIDAS version you use at your site, and `MIDASHOME`, the root directory for the MIDAS system code. These variables should have been set up correctly by your system manager when MIDAS was installed or use again `SETMIDAS` in VMS; for Unix these variables can again be chosen within the `inmidas` command. There are many more options for the `inmidas` command in Unix, which can be accessed interactively via the man page of `inmidas` (a complete Midas installation should also include the setting up of the man pages for `inmidas`, `gomidas` and `helpmidas`).
The detailed command description is as follows:

```
SYNOPSIS
   inmidas [ unit ] [ -h midashome ] [ -r midvers ] [ -d display ]
   [ -m mid_work ] [ -p/-P/-nop ] [ -noh ] [ -j midas-command-line ] [ -help ]

   Without arguments, inmidas initiates a  MIDAS  session  with default
   definitions. Some of these definitions can be modified with arguments in
   the command line of inmidas or by environment variables. The arguments
   in the command line override the corresponding environment variables.
OPTIONS
   inmidas has been configured by the Midas system manager at installation
   time to start a specific release of MIDAS. However, alternative releases
```

```
can be specified using the command line arguments:

-h midashome
   Home directory for MIDAS. Absolute pathname containing, at least, one
   release of MIDAS. It may also contain subdirectories for demo and
   calibration data.
-r midvers
   Release of MIDAS to be executed. It must be a subdirectory under
   midashome.
-d display
   Specifies another X server for the display and  graphical MIDAS windows
   NOTE: be aware of allowed access to a remote X server using the "xhost"
   command.
-p/-P/-nop
   Options -p and -P set the MIDAS environment variable MIDOPTION to
   PARALLEL while option  -nop sets it to NOPARALLEL (default: NOPARALLEL).
   In NOPARALLEL mode all intermediate MIDAS files in the MIDAS startup
   directory are deleted when starting MIDAS via inmidas.
   In PARALLEL mode no intermediate files are deleted, and this is
   necessary to run several MIDAS sessions with the same startup directory.
   With -P option and if unit is not given the system will select
   automatically one free unit for you. With -p option and no unit, the
   user will be requested to enter one.
unit
   Unit to be associated to the MIDAS session (default: 00 only if MIDAS
   is working in NOPARALLEL mode). Valid values for this option are in the
   range (00, 01, ..., 99, xa, ..., zz) where numerical values indicate
   that the user is working in an X11 environment (DISPLAY environment
   variable or argument -d should be given), and the others indicate an
   ASCII terminal.
-m mid_work
   Specifies the MIDAS startup directory (default: $HOME/midwork).
-noh
   Starts MIDAS without clearing the terminal and without welcome message.
-j midas-command-line
   midas-command-line will be executed in MIDAS as if it were the first
   command line typed in the MIDAS monitor.
   This option sets also the -noh  option.
   NOTE: midas-command-line should be typed between single quotes to be
   interpreted by inmidas as a single argument and to be passed to the
   MIDAS monitor as it is.
-help
   Display this help page.
```

So, to start MIDAS, type `INMIDAS` on a VMS system or `inmidas [arg1] ...` on a Unix system. This will initialize the MIDAS monitor as follows:

- In VMS the logical name `MID_WORK` is assigned to the MIDAS working directory; in Unix the environment variable `MID_WORK` is set accordingly. If the working directory does not yet exist, it is created.
  All internal files created by the MIDAS monitor will be stored in the MIDAS working directory. This is also the place to store your own `login.prg` as well as all your other MIDAS procedures which you want to execute from any other directory.

- In *Single User* mode, all MIDAS log- and keyfiles (FORGRxy.LOG, FORGRxy.KEY - where `xy` is the MIDAS unit described below) which exist in the MIDAS working directory as well as all MIDAS internal files are deleted.
  In *Parallel* mode no files are deleted.

- In VMS the user process is renamed to `MIDASxy`

- In VMS and in parallel mode in Unix you will be asked to enter the identification of a MIDAS unit as a two-character (case insensitive) string.
  Units are in the range (`00, 01, ..., 99, xa, ..., zz`) where numerical values indicate that the user is working in an X11 environment (DISPLAY environment variable should be set), and the others define a MIDAS session with no image display capabilities.
  So `23`, `xa`, `yf` or `Z3` will all be valid units. If you work in *Parallel* mode you have to use different MIDAS units for each session because the MIDAS unit is appended to the names of all MIDAS internal files.

On startup the current MIDAS version and patch level as well as the computer and operating system you are using are displayed together with a copyright notice. Then the prompt string

```
Midas 001>
```

appears on the terminal screen and you are ready to execute any of the available MIDAS commands.

The internal MIDAS files all reside in the MIDAS working directory `MID_WORK`, the data files are taken from the current working directory unless the complete file specification is given in the data file name.
Since MIDAS executes its applications in a child process (subprocess for VMS) which leaves no traces after termination, you cannot simply use the host command `SET DEF` (VMS) or `cd` (Unix) to change the working directory once you are in a MIDAS session. Instead, use the MIDAS command `change/direc` for that purpose.
Another possibility is to set the search path for your data files via the command

`set/midas_system DPATH=directory`. Use the MIDAS Help Utility for obtaining detailed information about these commands, e.g. `HELP change/direc`.
MIDAS is a case insensitive system. That means, you can type your input with upper or lower case characters. There are, however, some pitfalls with respect to the data files that reside in the local file system. In VMS, the system automatically translates all file names to upper case, so `LOLA.BDF` and `lola.bdf` specify exactly the same file. In Unix, file names may be specified using lower and upper case, so `LOLA.BDF` and `lola.bdf` are two different files. The convention in MIDAS is to always use lower case file names (e.g. in tutorial procedures) to guarantee portability between VMS and Unix. Also, all default file types are specified in lower case, e.g. `.bdf` and `.tbl` for images and tables.

Note

All MIDAS commands in the following sections are printed with capital letters. This is just for reasons of readability, i.e., to highlight them. The commands could all be typed in lower case as well.

3.2 MIDAS And the Host Operating System

Care has been taken that MIDAS and the Host Operating System (DCL for VMS and Bourne or C-shell for Unix) co–exist smoothly and complement each other. Migration from one environment to the other is simple:

If you are in the MIDAS environment, type `BYE` to switch back to the Host System.
If you have returned to the host environment from a MIDAS session, (indicated by the $–prompt in VMS, and by $ or % in Unix), type `GOMIDAS` (in VMS) or `gomidas` (in Unix) to revive MIDAS. The status of the keywords and the command buffer of the stopped MIDAS session are preserved - if you want to start afresh, use `INMIDAS` (VMS) or `inmidas` (Unix) again.
You may also use host commands directly inside MIDAS by preceding them with '`$`'. For instance,
`Midas 027> $DIR` (in VMS) or
`Midas 027> $ls` (in Unix)
will display the contents of the current directory.
Please, note, that currently this mode of operation will only invoke Bourne shell commands in Unix, not C-shell or Kornshell commands. To execute C-shell (or any other Shell) commands you have to insert them in a Bourne shell script which has as the first line: `#! /bin/csh`, or `#! /bin/ksh`, etc.

Note

If you work on a VMS system, beware of DCL command procedures:
DCL modifies command I/O streams when executing a procedure. This causes problems for the interprocess communication inside MIDAS. When executing a DCL procedure via `$ @` *'procedure' the correct settings will be maintained*

> *inside MIDAS.*
> *However assigning a symbol* `MIMI` *to the command above and then executing the DCL procedure by just typing* `$ MIMI` *will lead to disaster from which only a* `BYE` *and subsequent* `GOMIDAS` *will get you going again.*

Since images, tables, etc. are standard disk files, all host commands related to file operations can be employed. However, if a MIDAS catalog is used, care has to be taken that the information in the catalog is not invalidated, when e.g. renaming or deleting data files outside MIDAS (i.e. using commands of the host file system directly).

The output from MIDAS commands can be redirected to ASCII files enabling easy combination of MIDAS and host commands. E.g.
`Midas 123> READ/DESCR myimage * >dsc.dat`
will send all the output from the `READ/DESCR` command to the ASCII file *dsc.dat* (created in your current work directory) which can then be used by any host command. For example,
`Midas 124> $EDIT dsc.dat` (in VMS) or
`Midas 124> $vi dsc.dat` (in Unix)

Note

> *This mechanism is pretty much like the one used in Unix with the exception that there should be no space between the > and the output file name. Furthermore, this output redirection also works on VMS.*

`Midas 125> STATISTICS/IMAGE myimage >dsc.dat`
always creates a new file `dsc.dat`, if you want to append data to an existing ASCII file use
`Midas 126> STATISTICS/IMAGE myimage >>dsc.dat`
instead.
Finally,
`Midas 126> STATISTICS/IMAGE myimage >Null`
will suppress the output. If the keyword `LOG(12) = 0` (the default), the suppressed output is still written to the Midas logfile; if `LOG(12) = -1`, also the logging is suppressed.
As you may have guessed already, there is also input redirection. E.g.
`Midas 127> $ls a*.bdf >dscin.dat`
`Midas 128> READ/DESCR <dscin.dat`
will display the standard descriptors of all Midas images with names beginning with the letter 'a' in the current directory.
Again there should be no space between < and the file name.
See also the subsection 3.4.5 for more info about I/O redirection.

On a Unix system you can connect MIDAS and Unix command via the *pipe* symbol , e.g.
`Midas 129> READ/DESCR myimage * | $grep NGC425`
`Midas 130> $ls a*.bdf | read/descr`

3.3 MIDAS Data Structures

Here we describe and discuss the various data entities (structures) that MIDAS recognizes. They are stored in an internal binary format, accessible only through MIDAS and fall into the following categories:

Images are a set of data of same physical significance in one to three dimensions. The data must be sampled with constant step size along all 1, 2 or 3 axes and are stored in different formates, e.g., as bytes, 16 bit integers, or 32 bit reals on disk. However, most MIDAS applications work on real data, so the image pixels are converted on the fly to real format if necessary. The default file type is `.bdf` .
Besides the internal binary format also images in FITS format are supported by all Midas applications.

Tables are a structure for handling data which can be arranged in rows and columns. The data may be of numerical or character type. Numerical data may be sampled in any arbitrary fashion. The default file type is `.tbl` .

Fit-files are "degenerate" image files with just descriptors and no pixels and used to store the parameters needed for the fitting functions. The default file type is `.fit` .

Descriptors are variables attached to the structures mentioned above (i.e. stored in the same file) and describe the structure of the tables, images and fit files. They can also store any other auxiliary information connected to the data such as histograms, coordinates, comments and so on. For fit files they contain the fitting parameters.
The Midas descriptors are the equivalent of the FITS header keywords.

Catalogs contain lists of either images or tables or fit files for the purpose of grouping data together within MIDAS. They are exceptional in the sense that they are implemented as ASCII files so you can list and edit them (with care!) outside MIDAS. The default file type is `.cat` .

Keywords are variables which can be used to pass information from one MIDAS program to the next or to temporarily store intermediate results (there are also *reserved* or *system* keywords that keep MIDAS system parameters). They are referred to by a name and can be easily manipulated from the terminal or MIDAS procedures.

The individual data points in an image are referred to as "pixels" and in a table they are called "elements". The paragraphs below describe the structure of descriptors, and keywords, and the methods for specifying the individual data values in images and tables.

Note

There is no special syntax for file names in MIDAS. You can use any legal name of your host file system for images, tables and fit files. However, a name beginning with a digit or using any of the characters

*+, -, *, /, ‘, !, |, (and),   should be avoided, because these symbols will cause problems in e.g. the* `COMPUTE/IMAGE` *command. If you do want to use a file name with these special characters in a* `COMPUTE/IMAGE` *command you have to enclose the full name (including the file type) in quotes, i.e.*

`Midas 456> COMPUTE/IMAGE res = 12 + "quasar01+12.bdf"`

The length of these names is, in principle, limited to 60 characters for MIDAS applications (which used to be the size of keywords IN_A, OUT_A, employed in most procedures to store the image, table names...). However, via the `SET/MIDAS dpath=` *command you can specify other directories which are scanned for data files besides the current directory.*

Also file names like `abc.bdf.mine` *will not be appreciated by all MIDAS applications.*
As mentioned before, file names are case sensitive in MIDAS on Unix systems; names for descriptors and keywords are not. Thus, referring to a keyword with name `KEYA` *may be done e.g. via* `keyA` *or* `Keya`.

3.3.1 Specifying a Descriptor

Descriptors have been derived from the concepts used in a FITS file header and have many similarities with the FITS keywords. In particular the names of the MIDAS standard descriptors, e.g. `NAXIS`, `NPIX`, etc., (for details see Appendix “Standard Descriptors” of the MIDAS Environment doc.) correspond to those in the FITS header.

Descriptors come in four flavours: *integer, real, double precision* and *character*. Mixed types are not possible, i.e. you cannot have a real descriptor TEST and an integer descriptor TEST at the same time. Each descriptor also has a name (max. 48 chars.) and a length (no. of elements).

Note

NTT and VLT instruments will produce data frames with all ancillary information stored in the ESO hierarchical FITS keywords. In order to get ready for these long keywords, the max. size of the names of Midas descriptors has been extended from 15 to 48 chars. Therefore, images and tables created with a Midas version after 95NOV cannot be processed with an earlier Midas version! Files created with older Midas versions can, naturally, be used in the new (and future) Midas versions.
Since the 98NOV version you can set a switch (via `SET/MIDAS DSCFORMAT=OLD`*) to create new frames with the old descriptor format.*

Writing values into positions beyond the current length leads to an automatic extension of the descriptor (and update of its length) just as a text file is extended by the “editor” when you are editing it.
Beware, not to create uninitialized ‘holes’ by writing to descriptor elements which are not immediately following the current last element! MIDAS will not initialize the descriptor elements in between, so their values are unpredictable.

The command to write values into a descriptor requires the name of the data file (which could be an image, table or fit file), the descriptor name, the descriptor type, the first element to be accessed, and the total number of elements to be transferred (all separated by a '/' (slash)). Finally, the data values are given (separated by commas for numeric data, but no spaces). For example,

```
WRITE/DESCR imgfile Descname/C/1/7 Anyname
```

would write the ASCII string `Anyname` into the character descriptor `Descname` associated with the data file `imgfile.bdf`. Since spaces serve as parameter delimiters in MIDAS they have to be enclosed by double quotes (") if used as data. So

```
WRITE/DESCR imgfile Descname/C/1/7 "       "
```

would fill `Descname` with 7 blanks.

```
WRITE/DESCR imgfile Descname/R/4/3 17.3,8.8E2,-.3
```

would write the numbers `17.3, 880.0, -0.3` into elements 4,5 and 6 of real descriptor `Descname`. If the descriptor were created with fewer than 6 elements it would be expanded automatically.

```
WRITE/DESCR tblname.tbl Descname/R/4/3 17.3,8.8E2,-.3
```

would write the numbers `17.3, 880.0, -0.3` into elements 4,5 and 6 of real descriptor `Descname` of the table file `tblname.tbl`.
Note, that we had to add the file extension `.tbl` to the name `tblname`, since the command `WRITE/DESCR` defaults the first parameter to an image and appends the file type `.bdf` if no type is given by the user.

Single descriptor elements can also be written in a more direct way, via
`frame,descr = value`, e.g. to set `STEP(2)` of image `lola.bdf` to `1.234`, use
`lola,step(2) = 1.234`
The *value* can also be an *expression* made up of constants and elements of any MIDAS data structure, see the subsection 3.6.2.

This is how descriptors work at the most basic level. However, in many cases, higher level commands have been implemented to update specific descriptors. The MIDAS command `CUTS`, which sets the high and low cuts of an image (in descriptor `LHCUTS`) for displaying or plotting it, is an example of this.

Some of the commands dealing with descriptors are:
`READ/DESCR, WRITE/DESCR, SHOW/DESCR, DELETE/DESCR, INFO/DESCR, COPY/DD.`

An optional help text can be attached to each descriptor and is then displayed via the `READ/DESCR` and `SHOW/DESCR` commands. This text is copied from the original FITS file (if existing) when reading in the data file or can be explicitly set via `WRITE/DHELP`.

3.3.2 Specifying Keywords

As is the case for descriptors, keywords also have a name (max. 15 chars.), a type and a length (i.e. no. of elements). However, this length is fixed, and once the keywords are created with a certain size, they cannot be extended. The possible types for keywords are: *real, integer, character* and *double precision*. Like for descriptors there are no mixed types possible.
In order to write a value to a keyword, the same format as for descriptors is used.

```
WRITE/KEYWORD Keyname/C/1/8 AKeyword
```

This command would write the ASCII string `AKeyword` into the character keyword `Keyname` and

```
WRITE/KEYWORD Keyname/I/1/2 17,-22
```

would write the values `17` and `-22` into the integer keyword `Keyname` (elements 1 and 2).

Single keyword elements can also be written in a more direct way, via
`key = value`, e.g. to set the 12th element of keyword `INPUTR` to `1.234`, use
`inputr(12) = 1.234`
The *value* can also be an *expression* made up of constants and elements of any MIDAS data structure, see the subsection 3.6.2.

Some of the commands dealing with keywords are:
`READ/KEYWORD, WRITE/KEYWORD, SHOW/KEYWORD, DELETE/KEYWORD, COPY/KEYWORD, COMPUTE/KEYWORD`.
Keywords and descriptors can be copied to each other via `COPY/KD` and `COPY/DK`.

3.3.3 Specifying Elements in a Table

The MIDAS table file system is described in detail in chapter 5 of this volume. Here we just explain briefly how to access the various elements in a table file. To do so, it is necessary to specify the *table name*, the *column* and the *row*. This is done in the following format (order):
`table column row`
where
table is the table name;
column is the desired column which can be referenced by label as `:col` or by sequence number as `#n`;
row is the desired row referenced by number as `@n` or by a value in a predefined reference column.
Like descriptor and keyword names, column labels are case insensitive. The command:

```
READ/TABLE tname #3 @10
```

would display the element in column 3 of row 10 in table file `tname.tbl` .
Similarly, the command:

```
READ/TABLE tname :MAGNITUDE 20.0
```

would access the element in the column labeled 'MAGNITUDE' and value 20.0 in the reference column (this reference column must have been defined before via the SET/REFCOLNUM command).
Note, that we need not specify the file extension .tbl as in the descriptor related commands. All table commands default the data files to tables with extension .tbl.

Single table elements can also be written in a more direct way, via
`table,column,row = value`,
e.g. to set the element in the 3rd row of the column labeled :XREF in the table lola.tbl to 1.234, use
`lola,:xref,@3 = 1.234`
The *value* can also be an *expression* made up of constants and elements of any MIDAS data structure, see the subsection 3.6.2.

Some of the commands dealing with tables are:
READ/TABLE, WRITE/TABLE, COMPUTE/TABLE, SHOW/TABLE, EDIT/TABLE, COPY/TABLE.

3.3.4 Specifying Pixels in an Image

In some commands it is necessary to specify the columns and rows of an image to which that command should refer.
This is done in the following way for e.g. a 2-dim frame: `frame[x1,y1:x2,y2]` where the column specification, x or the row specification, y can be any of

- world coordinates, indicated via real or integer numbers: 20.0,300
- pixel numbers, indicated via integers preceded by @: @35,@200
- or a special symbol to indicate *start* (<), or *end* (>) of a row or column; thus `[@20,<:@20,>]` specifies the complete 20th column of a 2-dimensional image

World coordinates are the physically meaningful coordinates with units such as wavelengths or arc seconds (which are defined in the descriptor CUNIT). Pixel numbers (starting with 1 for each dimension) are the indices of an image seen as an array.
For example, extracting the complete 12th plane from the 3-dim image stored in cube.bdf is done via
`EXTRACT/IMAGE plane12 = cube [<,<,@12:>,>,@12]`
Single pixels can also be written in a more direct way, via
`frame[x,y,z] = value`, e.g. to set the pixel in row 27 and column 1023 of the 2-dim image lola.bdf to 1.234, use `lola[@1023,@27] = 1.234`
The *value* can also be an *expression* made up of constants and elements of any MIDAS data structure, see the subsection 3.6.2.

Some of the commands dealing with images are:
`READ/IMAGE, WRITE/IMAGE, COMPUTE/IMAGE, STATIST/IMAGE, DELETE/IMAGE, COPY/II.`

3.3.5 Specifying Sub-Image

In all MIDAS commands which accept images as parameters you can also provide subimages in the syntax specified above.
For example, the command `LOAD/IMAGE` loads an image into the MIDAS display (window). Then,
`Midas 567> LOAD/IMAGE mygalaxy`
would try to display the whole 2-dim image *mygalaxy.bdf*, whereas
`Midas 568> LOAD/IMAGE mygalaxy[@100,@200:@199,@299]`
would only load a 100x100 subimage of *mygalaxy.bdf* beginning at the 100th x-pixel and 200th y-pixel.

3.4 Command Syntax

After start–up, the MIDAS monitor prompts you to interact with MIDAS by entering commands on the terminal which are terminated by `Return` (Enter key).
To enter a command-line on more than one terminal/window line, use the continuation character, a minus sign (`-`), as the last character of a line. The command line is limited to 400 characters. If you want to enter more than one command on a single line, separate the commands with a semicolon and space (`; `).
A MIDAS command line is structured as

```
command/qualifier par1 par2 ... par8 !comments
```

The command describes the general action you want to perform (a verb) and the qualifier usually specifies the object of that action, e.g. `WRITE/DESCRIPTOR`. The (at most) 8 command parameters have a max. length of 100 chars. and hold all other information needed to perform the required action. All parameters are separated by spaces.

Currently the following "objects" exist in MIDAS:

- keywords
- descriptors
- bulk data frames
  - images
  - tables
  - fit files
  - ASCII files
- catalogs

  - for images
  - for tables
  - for fit files

- auxiliary image–display data structures (where applicable)
  - LUTs (Colour Look–Up Tables)
  - ITTs (Intensity Transfer Tables)
    (internally these structures are stored in MIDAS table files)

All user input and output from MIDAS commands is recorded in an ASCII file, named FORGRxy.LOG (with `xy` the MIDAS unit) and stored in the `MID_WORK` directory. This MIDAS logfile serves as a hardcopy of a full MIDAS session. Terminating MIDAS with `BYE` and continuing later on via `gomidas` will not restart a new logfile but append to the existing one. The logging in MIDAS can be controlled via commands like `LOG/OFF, LOG/ON, LOG/TOF, DELETE/LOG`.
Comments may be appended to the command string and are separated by at least one white space and `'!'` (exclamation mark) from it. To give a complete line of comments, enter `'!'` as the first character of the input line (useful for structuring the contents of the MIDAS logfile).

Commands and qualifiers may be abbreviated to the number of significant characters needed to distinguish them from the rest. At most 6 characters are necessary for the command and 4 characters for the qualifier. Command and qualifier are separated by a `'/'` (slash). Nearly all commands need a qualifier, but there is only one qualifier per command (e.g. `comm/qual1/qual2` is unsupported in MIDAS). In case you omit the qualifier, the default qualifier of that command is used by MIDAS. The default qualifier of a MIDAS command *mycomnd* may be displayed via `SHOW/COMMAND mycomnd`. For example, the default qualifier for the `LOAD` command is `IMAGE`, so typing `LOAD/IMAGE` or `LOAD` will have the same effect.

The parameters depend on the actual command. A space (blank) is the delimiter for parameters in the command-line. Commas are used to subdivide parameters. If you need a space inside a parameter, this parameter has to be enclosed in double quotes (").
Normally, parameters are position dependent, i.e. `par1` is the first, `par2` the second, and so on. This may be overridden by using the following syntax:

```
command/qualifier P4=par4 P1=par1 P7=par7 ... !comments
```

If the command procedure which is activated by a MIDAS command uses the `CROSSREF` command, it is also possible to execute that command via:

```
command/qualifier label4=par4 label1=par1 label7=par7 ... !comments
```

The help text of each command specifies whether such a cross referencing of parameters is possible and if so, which labels to use. For details about the command `CROSSREF` see the description of it in the section below on MIDAS procedures.

Whenever possible parameters have defaults. If you do not want to override them use the symbol '?' (question mark) for a parameter if you use the position dependent format. Therefore,
`command/qualifier P4=22.345` is equivalent to
`command/qualifier ? ? ? 22.345`
The preset default values of MIDAS commands can be overridden with `CREATE/DEFAULTS`. For example, the default for the descriptor name(s) in the `READ/DESCR` command is `NAXIS,NPIX,START,STEP,IDENT,CUNIT,LHCUTS` for an image frame. Thus, typing
`Midas 234> READ/DESCR myimage`
will display the contents of the descriptors NAXIS, NPIX, START, STEP, IDENT, CUNIT and LHCUTS of image `myimage.bdf`. After changing this default via
`Midas 235> CREATE/DEFAULTS READ/DESCR ? HISTORY`
the same command
`Midas 236> READ/DESCR myimage`
will display the contents of the descriptor HISTORY of `myimage.bdf`.

To abort a MIDAS command, use `Ctrl/C` (sometimes you also have to hit `Return`), which will return control to you.

Note

Be careful when aborting commands which interact with a display/graphics window in the X11–Environment. For, you run the risk of losing the synchronisation with the MIDAS display server, which must then be re–initialized via `RESET/DISPLAY` *(see chapter 6 for details).*

3.4.1 Command Recalling

By default, the last 15 commands entered on the terminal are kept in an internal buffer (the no. of commands saved can be changed via `SET/BUFFER`). To recall (and execute) any of these commands, simply type the associated command number. This is the number "xyz" appearing in the prompt `Midas xyz>` when that command was entered. To display the command buffer, simply hit `Return`.
If you want to recall more than one command at once, enter all the relevant command numbers (separated by a semicolon and space), e.g. enter `14; 17; 22` if you want to repeat the commands numbered 14, 17 and 22 . Also `14; read/keyw in_a; 17` is possible.

To recall commands not by number but by pattern, use `:pattern` to repeat the last command matching the specified pattern. For example, if the last two commands in your command buffer are:

```
22 READ/IMAGE supernova
23 show/commands
```

Then, typing 22 as well as `:READ` or `:nova` will execute the command `READ/IMAGE supernova` again. Note that for the pattern matching MIDAS *does* make a distinction between upper and lower case.
You can also use the vertical arrow keys to navigate up and down through the command buffer.
Besides repeating complete input lines it is also possible to just use parts of the last command line. Each "token" of the last command line is saved internally until the next input. A "token" is the information separated by spaces in the command line. To repeat the tokens on a subsequent command line merely type a '.' For example, if you have in the command buffer:

```
READ/KEYWORD in_a
LOAD/IMAGE myframe 0 2,2
```

Then typing `' . yourframe . . '` as the next command is equivalent to typing `'LOAD/IMAGE yourframe 0 2,2'`.

All features described so far apply to genuine MIDAS commands as well as to host system commands (where the first character of the command line is the $ sign).

Some words of caution:
In VMS the version number of files may be specified using a semicolon, e.g.
`$ RENAME file.typ;7 lola.bdf`.
Typing such a command inside MIDAS will not work, since the monitor will interpret this input as two Midas commands. Instead, use a dot to separate the version number, e.g.
`Midas 234> $ RENAME file.typ.7 lola.bdf`.
In Unix the repetition of tokens may cause trouble. Consider the following:
`Midas 123> load/image vaca`
`Midas 124> $cp /elsewhere/toro.bdf .`
The intention was to simply copy the file `toro.bdf` from somewhere else to the current directory. But instead of `toro.bdf` you will find a strange file named `?` in your directory... In the line '123' only two tokens are entered, so all other 8 tokens are set to the default value `'?'`. In line '124' the third token will be set to the third token in the line above, so it changes to:
`Midas 124> $cp /elsewhere/toro.bdf ?`
Instead, specify also the result frame completely, e.g.
`$cp /elsewhere/toro.bdf toro.bdf`.
Preceding a host command by $$ disables the interpretation of specific symbols by MIDAS, thus
`Midas 124> $$cp /elsewhere/toro.bdf .`
will actually do the expected copy.

3.4.2 Command Line Editing

The commands in the internal command buffer may also be edited.

On Unix systems MIDAS comes (since the 94NOV release) with two different line-editors: *TermWindows* (the one developed at ESO) and *readline* (which is from the GNU project). The default line-editor is *readline* which provides a history stack of commands, `emacs` or `vi` editing functions, command and filename completion functions, and a communication channel to the MIDAS GUI XHelp for the On-Line Help utility. See the man page of `readline` for a complete list of options.
To use the old *TermWindows* line-editor just set the environment variable `TERMWIN` to `yes`. Please, note, that the *TermWindows* editor will be phased out in a future MIDAS release.

On VMS systems MIDAS comes only with the *TermWindows* line-editor.

To edit a MIDAS command, type the command number preceded by a dot (period) or followed by a dot. So '`.xyz`' or '`xyz.`' will both display the command '`xyz`' and put you into the edit mode where you can modify that command.
If you employ the pattern matching style, use '`.:pattern`' or '`:.pattern`' to edit a previous command containing that pattern.
You edit that line using the *arrow* keys and *delete* key of the keyboard and retyping the characters. On VMS systems to toggle between *Replace* and *Insert* mode use `CtrlA`.
Using '`.xyz`' ('`.:pattern`') will lead to the creation of a new MIDAS command with new command number, whereas '`xyz.`' ('`:.pattern`') modifies the specified command directly (and keeps this command number).
Also the commands recalled via the arrow keys can be edited.
As mentioned above only the 15 most recently used commands are kept in the command buffer on a first–in, first–out basis. So if you repeat or edit a certain command via its MIDAS command–number at least once in 15 command inputs, this command will always be kept in the buffer.
However, you may wish to make sure that a command remains always in the buffer. Entering '`xyz/LOCK`' will lock the command with number '`xyz`' in the buffer; to unlock the command, use '`xyz/UNLOCK`'.
The command `CLEAR/BUFFER` empties the command buffer and resets the command counter to 1. Since 3 digits are used for the command count, the counter is also reset to 1 after MIDAS command no. 999.

3.4.3 Command Line Suspension

If, while entering a command, you realize that you forgot the full command syntax or want to check something else, a mechanism has been introduced to let you interrupt the command line, execute another command or commands, and then resume with the interrupted line. To interrupt a command line enter '\' (back–slash) as the last character and hit `Return`. The command string is then saved internally. To resume entering the interrupted command line, type '\' (back–slash) again followed by `Return`. The saved command line will be displayed on the terminal and you may add more input.

Note

You cannot edit or change the saved portion of the command after reentering the interrupted string, since your new input is handled as if it were a continuation of the original command line.

3.4.4 On–Line Help

The help facility of MIDAS (command `HELP`) provides detailed descriptions of all supported commands and qualifiers. This applies also to the HELP command itself. If you work in an X-Window environment we suggest to use `XHelp`, the graphical user interface to the MIDAS Help facility, by executing the MIDAS command
`CREATE/GUI HELP`.
It is recommended to execute this command before any `LOAD/IMAGE` command, otherwise you could get problems with missing colors.
Besides providing a separate and convenient interface to the MIDAS Help utility this GUI also supports a *feedback* facility for reporting errors, problems and suggestions to the MIDAS group at ESO (this problem–report mechanism is based on the commercial *Action Remedy* System).

And if you use the *readline* line-editor you can also update the MIDAS Help facility with the current command-line by typing `Ctrl/X` or `F1`.

Also, the `TUTORIAL` commands will help you in exploring the MIDAS system. Use the MIDAS command `HELP TUTORIAL` to find out which tutorials exist and try them out.
There is also a tutorial about the `HELP` command itself. Use `TUTORIAL/HELP` to exercise many of the features described in this section.
Besides the `HELP` GUI you can enter `HELP` commands from the terminal; the syntax for using `HELP` from the command line are given in the table *Help Syntax*.

Normally, the HELP text is not written to the MIDAS logfile (to save space in the logfile) but if you wish to include this, you may do so by setting keyword LOG(3) to 1 via e.g. the MIDAS command `LOG(3) = 1`. To print out the help text, use `PRINT/HELP`.

3.4.5 Input/Output Redirection in Midas

Similar to Unix MIDAS commands also accept input from an ASCII file instead of an input line typed on the terminal. For example, you can use host commands to create a list of files and then execute a MIDAS command on all files in that list:

```
Midas 045> $ ls bb*.bdf >input.dat
Midas 046> STATISTICS/IMAGE <input.dat
```

This command sequence will execute the `STATISTICS/IMAGE` command for all frames in the file *input.dat*, i.e. all images beginning with 'bb'.
For more elaborate processing of groups of files you should use MIDAS catalogs and the `STORE/FRAME` command (see section 3.9 for detailed info about catalogs).

Command	**Description**
`HELP`	To display all currently existing MIDAS commands and topics
`HELP` *comnd*	To display all the *comnd/qualif* combinations available for the given *comnd*
`HELP` *comnd/qualif*	To get detailed information about the specified *comnd/qualif* combination
comnd/qualif ??	To display full command syntax (one line) of specified *comnd/qualif*
pattern?	To list all commands which begin with given *pattern*
`HELP/QUALIF` *qualif*	To list all commands which may use the given *qualif*
`HELP/SUBJECT` *subject*	For any info related to *subject*
`HELP/CL` *comnd*	To get detailed information on the MIDAS command language *comnd*
`HELP/KEYWORD` *keyword*	To get detailed information about the specified *keyword*
`HELP` *[Topic]*	To get information about a *topic*, e.g. the standard descriptors or available contexts
`HELP/APPLIC`	To get information about available application procedures
`HELP/CONTRIB`	To get information about available contributed procedures

Table 3.1: Help Syntax

Different ways exist to save the output of a MIDAS command in an ASCII file besides just using the logfile. There is a set of `PRINT/...` commands to print out the contents of the different MIDAS data objects, e.g. `PRINT/TABLE`. By default the output from these commands is sent to a printer but this can be changed to a file via the `ASSIGN/PRINT` command. So, if you want an ASCII copy of a MIDAS table, you do:

```
Midas 088> ASSIGN/PRINT file mytable.dat
Midas 089> PRINT/TABLE mytable
```

Another possibility which is applicable to all MIDAS commands, not just the `PRINT` commands, is to specify the output file directly in the command. Thus,

```
Midas 089> WRITE/TABLE mytable >mytable.dat
```

is equivalent to the two commands above. The file *mytable.dat* is created in the current directory. If you want to append the data to an existing file, use

```
Midas 090> WRITE/TABLE mytable >>mytable.dat
Midas 090> WRITE/TABLE mytable >terminal
```

only produces output on the terminal, i.e. it's the same as

```
Midas 090> WRITE/TABLE mytable
```

but is useful if you want to provide optional file output in a procedure (see section 3.6.2 for more details about that).

In case, the output should go to a file and also be displayed in the MIDAS command window, use

```
Midas 091> WRITE/TABLE mytable >mytable.dat+terminal
```

If you want to suppress the output completely use the special name `Null` for the output file, e.g.

```
Midas 092> WRITE/TABLE mytable >Null
```

will omit all output. No file `Null` is created.

Currently it is not possible to redirect the input as well as the output in the same command line, e.g.

```
Midas 046> STATISTICS/IMAGE <input.dat >output.dat
```

is not possible.

Note

No space should be between the <, > or >> and the in/output ASCII file names!
This output redirection scheme has been modeled after the way it is done in Unix but it also works on VMS systems.

3.5 Execution of Commands

MIDAS commands fall into two categories: the *basic* commands and all other *application* commands. The *basic* commands are executed inside the MIDAS monitor, which is the program you are interacting with. All other commands are implemented by executing a MIDAS procedure which runs one or more programs in a subprocess (child process). During the time a command is being processed in the subprocess, the MIDAS monitor is suspended until the corresponding program terminates in the subprocess. Only then control is returned to the user. To stop a command prematurely, type `Ctrl/C`.

Since process creation is much more expensive in VMS than in Unix these subprocesses are handled differently in VMS and Unix:
In VMS, the subprocess, named FORGRxy (with `xy` the MIDAS unit specified at start–up), is created at MIDAS initialisation time and kept alive until you exit from MIDAS via the command `BYE`.
In Unix, the child process is created each time the MIDAS command executes an application program. Upon termination of that program the child process dies. This also applies to commands of the host system – they are executed in a subshell.
Therefore, issuing `'$ cd /elsewhere'` inside MIDAS does not change your current directory permanently... You should use the MIDAS command `CHANGE/DIRECTORY` instead. See section 3.1 about the *change-directory* problem.

Some internal files are created when starting a MIDAS session in the directory specified via `MID_WORK:`. The most important ones are the *keyword file* and the *logfile*.
The *keyword file* is named `FORGRxy.KEY` (`xy` the MIDAS unit) and holds the keyword data base accessible by all programs running in the MIDAS environment.
The *logfile* is named `FORGRxy.LOG` and receives a log of all user input and all MIDAS output on the terminal (except HELP text, as explained before, and output from the host system). The *logfile* serves also as a "fall back" utility in case of system crash or other breakdown. In such a case the command `PLAYBACK/LOG 'logfile'` may be used to regenerate the complete MIDAS session.

Note

In order to use the playback facility, you have to rename the original logfile before restarting MIDAS via `INMIDAS` *or* `inmidas`. *Remember that INMIDAS deletes old MIDAS logfiles unless you run in* `parallel` *mode.*

3.6 MIDAS Command Language

The MIDAS command language (`MCL`) consists of all the commands which you enter interactively, and an additional set of commands to provide the necessary tools to write MIDAS "programs", called MIDAS *procedures*.
The `MCL` is a flexible and powerful tool to integrate application modules into MIDAS and to do rapid prototyping. But it is not intended to be a full blown programming language - for programming tasks MIDAS supports the standard interfaces in FORTRAN 77 and C (cf. the MIDAS Environment document). It is an interpreted language, so you do not need to compile MIDAS procedures. It is also a "Macro" language in the sense that you can build complex procedures, attach these procedures to a MIDAS command and qualifier combination and then put a single line with that command name into yet another procedure (up to 25 levels deep).

MIDAS procedures are handled in the following way:
The ASCII procedure file is read in by the MIDAS monitor and translated into an internal more compact format. This translated code is then executed inside the Monitor.
The individual lines of code are parsed and decoded in two passes: In the first pass, all symbol substitutions are done using the specified formats to convert from binary to ASCII. In the second pass, all control and conditional statements are processed directly by the Monitor (e.g. positioning the internal program pointer to the command line referred to by a `GOTO` statement) until an "executable" command line is found which is passed on to the usual command input pipeline of MIDAS as if it were typed in by the user.

For a detailed explanation of all the MIDAS Command Language commands see the appendix of this volume or use the MIDAS command `HELP/CL`.

The following *Command Language* commands provide the necessary programming constructs like looping and conditional branching for MIDAS procedures, they cannot be used interactively:

`BRANCH variable comparisons labels`
Compare *variable* with *comparison* values and branch to related *labels*

`CROSSREF label1 ... label8`
Define cross reference *labels* for parameters par1 ... par8

`DEFINE/LOCAL key data all_flag level_flag`
Define local keyword *key* and initialize it using *data*

`DEFINE/PARAMETER par def type prompt limits`
Declare *default* value, *type*, *prompt*string and *limits* for parameter *par*

```
DO loopvar = begin end step_size
  ...command body...
ENDDO
```
Execute a do–loop (as in FORTRAN)

`ENTRY procedure`
Define the beginning of a MIDAS *procedure* in procedure file with a different name

`GOTO label`
Jump to a *label* defined as *label:*, see below

`IF par1 op par2 command`
Execute conditional statement (as in FORTRAN)

`IF par1 op par2 THEN`

```
  ...if-sequence...
ELSEIF par1 op par2 THEN
  ...else if-sequence...
ELSE
  ...else-sequence...
ENDIF
```

Execute a conditional statement (as in FORTRAN)

`INQUIRE/KEYWORD key prompt-string`

Demand value for *key* from the user

`label:`

Declare a *label*, with *label* a string beginning with a letter, e.g. `Step_A:`

`RETURN par1 ... par3`

Return to calling procedure or terminal and pass up to 3 parameters

`PAUSE`

interrupt the current procedure and return to interactive level

`DEFINE/MAXPAR nopar`

Indicate that max. *nopar* parameters are expected

The following commands may also be used interactively, but are especially useful inside MIDAS procedures:

`@` (or: `@@`, or: `@a`, or: `@s`, or: `@c proc par1 ... par8`)

Execute the MIDAS procedure *proc* which is stored
in MID_PROC:, (or in one of the current procedure directories)
or in APP_PROC:, or in STD_PROC:, or in CON_PROC:, respectively

Note

Up to 4 directories for procedures can be specified via the command `SET/MIDAS PATH=`*. By default the current directory and the directory pointed to by the variable* `MID_WORK` *are used to localize Midas procedures when using the* `@@` *command.*

`ECHO/qualif levela,levelb`

Control the display of MIDAS commands (*qualif* = ON, OFF, FULL)
for procedures executing at a level in the interval [*levela,levelb*]

`COMPUTE/KEYWORD reskey = expression`

Evaluate an algebraic *expression* involving keywordss and constants, store result in *reskey*

`SET/FORMAT format_specs`
Define *formats* used for replacements of keyword and descriptor names in procedures with their actual values

`WRITE/OUT text`
Display *text* on terminal

`! comment`
Indicate beginning of a *comment* line

Note

It is good practice NOT to abbreviate the commands and qualifiers of a MIDAS command inside a procedure. Since new MIDAS commands can be created any time an abbreviated command may work at one time but become ambiguous at other times and cause the procedure to fail.
The command `TRANSLATE/SHOW myproc X` *will check that all commands and qualifiers are fully specified in procedure* `myproc.prg`.

Also avoid relying on the defaults of parameters of a given command. With the command `CREATE/DEFAULT` *these defaults can be changed dynamically. Instead write full command lines specifying all parameters in your procedures.*

3.6.1 Passing Parameters in MIDAS Procedures

A MIDAS command procedure may be created with an editor or via the command `WRITE/COMMANDS` which constructs a MIDAS procedure from the current command buffer. Default type for such a procedure file is `.prg` . This MIDAS procedure can then be executed with the commands:

```
@  file par1 par2 ... par8          ! if the procedure is in MID_PROC
@@  file par1 par2 ... par8         ! if in current directory or MID_WORK [1]
 @a  file par1 par2 ... par8        ! if in APP_PROC
@s  file par1 par2 ... par8         ! if in STD_PROC
@c  file par1 par2 ... par8         ! if in CON_PROC
```

where `par1 ... par8` are the actual parameters which may be accessed within the command procedure through the character keywords P1 ... P8.
`SET/MIDAS_SYSTEM PATH=directory`.

The maximum size of a single parameter is 100 characters, but all parameters together may not exceed 400 characters (which is the maximum size of a command line). The size

[1] These are the default directories, as explained in the previous section.

of the code of a procedure is not limited.
In the following, let us assume that all procedures are stored in the directory specified by MID_WORK so that we always use the MIDAS command @@ to execute them. A command procedure in turn can execute another command procedure (or itself) – up to 25 procedure levels deep. The end of a procedure file or the commands RETURN or ENTRY will bring you back up to the next higher level.
To pass parameters back to a higher level command, use the command
RETURN retpar1 ... retpar3 . These return values can then be accessed via the character keywords Q1, Q2, Q3. This technique is an alternative to using global keywords for that purpose.
To use the actual values of a parameter in the procedure, the formal parameters P1,...,P8 have to be enclosed in curly brackets ({, }):

```
!+
!  Example 1, MIDAS procedure exa1.prg
!+
READ/KEYWORD {P1}        ! read keyword the name of which is given as par1
@@ test {P2}             ! execute test.prg and pass par2 as first parameter
WRITE/KEYWORD INPUTC {P2} ! write contents of par2 into keyword INPUTC
```

Entering the MIDAS command @@ exa1 OUTPUTC ESO-Garching will lead to the execution of:

```
READ/KEYWORD OUTPUTC
@@ test ESO-Garching
WRITE/KEYWORD INPUTC ESO-Garching
```

The command @@ always passes 8 parameters to a command procedure. If fewer than 8 parameters are specified in the command line, dummy parameters (indicated by the special character '?' (question mark)) are internally appended.
Therefore, @@ exa1 OUTPUTC will put the character '?' into the first element of the character keyword INPUTC.

If we enter the command ECHO/ON before executing the procedure we would actually see the above commands displayed on the terminal (cf. subsection 3.8.1).

Note

Up to MIDAS release 88NOV apostrophes were used for symbol substitutions (e.g. 'P1'). Because of the backward compatibility of MIDAS you could still use apostrophes to indicate symbol substitutions, which is, however, discouraged. The main reason being that using { and } instead, makes nesting of substitutions possible.

The command DEFINE/PARAMETER should be used for each parameter that is referenced in

the procedure. This command will set the defaults, the type, and the prompt string for each parameter. For numeric values passed as parameter also lower and upper limits can be specified in the DEFINE/PARAMETER command.
The default values defined inside the procedure will be used in case the parameters are not explicitly provided (i.e. entered as '?'):

```
!+
!  Example 2, MIDAS procedure exa2.prg
!+
DEFINE/PARAM P1 999 NUMBER "Enter first input number:" 22,1024
DEFINE/MAXPAR 1                          ! only 1 parameter expected
WRITE/KEYWORD INPUTI/I/7/1 {P1}      ! store contents of P1 in INPUTI(7)
```

The MIDAS command: `@@ exa2 77` will set INPUTI(7) to 77, whereas `@@ exa2` will set INPUTI(7) to 999.
Entering `@@ exa2 17` will result in an error since the valid interval for the number passed as the first parameter is [22,1024].
If you do not want to give default values for a parameter (in other words, if specific input is required for this parameter), use the symbol '?' as default. In that case, and if the relevant parameter is not given, the user will be prompted for this parameter (using the prompt string specified in the DEFINE/PARAMETER command) .
The DEFINE/PARAMETER line above also demonstrates how to put a character string with embedded blanks into a single parameter (remember that blanks are parameter delimiters in MIDAS) by enclosing the prompt string with double quotes.

The DEFINE/PARAMETER command also checks the type of the parameter. The types which may be tested are: I(mage), T(able), F(itfile) , N(umber), C(haracter).
If for any reason you do not want type checking, use the character '?' instead of any of the types listed above.
For file–type parameters the following translations are executed:
catalog entry numbers, e.g. #27 are replaced by the corresponding file name in the catalog (if that catalog is active!) and the asterisk ('*') is substituted by the currently displayed image, if any.
For numerical parameters it is tested if the input is a number; for character strings it is only checked that the first character is a non–numeric character.

Using the plus sign ('+') as default value is another way to disable parameter type checking. This is the correct way to test inside a procedure whether a certain parameter has been entered or not, because it is impossible to distinguish between a parameter defaulted to '?' and an explicitly entered '?' parameter (see example 14a, 14b in subsection 3.6.5).
The system keyword PARSTAT holds 8 flags (for P1,...,P8) which are set to 1 or 0, if the type of the *ith* parameter conforms to the specified type or not. If PARSTAT(i) is 0 for any i the MIDAS procedure is aborted.
However, if /C(ONTINUE) is appended to any of the types listed above, the keyword

PARSTAT will only be set to 0 or 1 and the execution of the procedure continues, leaving it to the user to test PARSTAT(i) and decide how to go on.
So in our example above the command `@@ exa2 KB` will result in an error message and the procedure is aborted.
If we change the procedure to:

```
!+
!  Example 3, MIDAS procedure exa3.prg
!+
DEFINE/PARAM P1 999 N/CONT "Enter first input number:"
DEFINE/MAXPAR 1                          ! only 1 parameter expected
IF PARSTAT(1) .EQ. 1 -
  WRITE/KEYWORD INPUTI/I/7/1 {P1}    !store contents of P1 in INPUTI(7)
```

then `@@ exa3 KB` will not yield any error.
If we enter `@@ exa3 RW PG CG KB MP PB` the message
`Warning:  5 parameter(s) more entered than required...`
will be displayed but the execution of the procedure continues (the additional parameters are ignored). Note also the use of the continuation character (-) in the IF statement above.

The MIDAS command CROSSREF defines labels (of maximum 10 characters) for the parameters P1,...,P8 to enable cross-referencing of parameters if they are passed in arbitrary order.

Note

The command CROSSREF *has to be the first executable command (i.e. any command but a comment line) in a MIDAS procedure!*
The command DEFINE/MAXPAR *provides an additional consistency check and helps to detect erroneous usage of MIDAS procedures. Therefore, it's highly recommended to include it in all procedures.*

If we modify `exa3.prg` to:

```
!+
!  Example 4, MIDAS procedure exa4.prg
!+
CROSSREF IN_FILE OUT_FILE METHOD ALPHA
DEFINE/PARAM P1 ?   IMA "Enter name of input file:  "
DEFINE/PARAM P2 ?   IMA "Enter name of result file:  "
DEFINE/PARAM P3 ?   C  "Enter method:  "
DEFINE/PARAM P4 999 NUM "Enter alpha value:  " 22,1024
DEFINE/MAXPAR 4                         ! max. 4 parameters expected
WRITE/KEYWORD INPUTI/I/7/1 {P4}
```

then the following command lines will all be equivalent:

```
@@ exa4 in out FILTER 33
@@ exa4 P2=out P1=in P4=33 P3=FILTER
@@ exa4 OUT_FILE=out IN_FILE=in alpha=33 METHOD=FILTER
```

The labels may be truncated, so also

```
@@ exa4 OUT=out IN_F=in al=33 METH=FILTER
```

is o.k.

If you do not know a parameter value at the time you execute a MIDAS procedure, e.g. the value depends on the execution inside the procedure itself, use the command `INQUIRE/KEYWORD` in the procedure. The execution of the procedure is then interrupted and the user is prompted for a value before continuing. For example,

```
!+
!  Example 5, MIDAS procedure exa5.prg
!+
CROSSREF IN_FILE OUT_FILE
DEFINE/PARAM P1 ?   IMA "Enter name of input file:  "
DEFINE/PARAM P2 ?   IMA "Enter name of result file:  "
DEFINE/MAXPAR 2                          ! max. 2 parameters expected
WRITE/KEYWORD IN_B " " all              ! fill keyword IN_B with blanks
INQUIRE/KEYWORD IN_B "Which filter, enter LOW or HIGH: "
IF AUX_MODE(7) .EQ. 0 IN_B = "LOW"                 ! LOW is the default
```

The command `@@ exa5 old new` will stop with the message
`Which filter, enter LOW or HIGH:`
and wait for user input. The 7th element of keyword `AUX_MODE` will contain the number of characters typed in response to the `INQUIRE/KEYWORD` command. `AUX_MODE(7)` is set to 0 if the user just types `Return`.

3.6.2 Symbol Substitution in Command Procedures

As mentioned before, the Monitor performs symbol substitutions on MIDAS command lines in the first pass by replacing symbol names in the command line with their current value. For character symbols just the string is put in; for symbols of other types the binary data are converted to ASCII using the formats specified in the `SET/FORMAT` command. This substitution is iterated until no more symbol substitutions are possible. Keywords, descriptors, pixel values of an image or elements of a table are valid symbols in the MIDAS command language.

The following syntax for substitution is used to distinguish among keywords, descriptors, pixel values and table elements:

{star} is a string representing the value stored in the keyword *star*
If *star* is not a character keyword, the numerical value of *star* will be converted to an ASCII string according to the format defined via the SET/FORMAT command. Therefore, take care, you may lose precision if an inadequate format is chosen.

{galaxy,disk} is a string representing the value stored in descriptor *disk* of the image frame *galaxy.bdf*
If *disk* is not a character descriptor, the numerical value of *disk* will be converted to an ASCII string according to ...

{galaxy[x,y]} is a string representing the value of the image pixel at coordinate x,y of the 2-dimensional frame *galaxy.bdf.*
This value will always be converted to an ASCII string according to ...

{dust,:particles,7} is a string representing the element of the table *dust.tbl* in the column labeled *:particles* and row *#7*
If the table element is not of type character, the numerical value will be converted to an ASCII string according to ...

{dust,#2,77} is a string representing the element of the table *dust.tbl* in the second column and row *77*
Elements of numerical keywords with more than one element are specified like elements in a FORTRAN vector, e.g. `INPUTR(7)`, that also means that the first element has the index 1 (not 0). Also substrings of character keywords are indicated as in FORTRAN, e.g. `INPUTC(2:5)`. These features are also implemented for descriptors but not for table entries (yet).

Since the outcome of symbol substitution is a character string, it can be used in place of any operand of a Midas command, like the command, qualifier or parameters, e.g. `WRITE/OUT {{P1},STEP(1)}`.

It's with the the command `COMPUTE/KEYWORD` and its short form `variable = expression` (called *direct assignment*), with *variable* any of *keyword, descriptor, image-pixel* or *table-element* that symbol substitution gets a bit more complex. Therefore, we shall elaborate on that in some more detail.

First, the result (left hand side) keyword, descriptor, table element, etc. is specified WITHOUT the brackets, i.e

```
Midas > compute/keyw dust,#2,77 = inputr(6)*m$sin(inputr(12))
```

or

```
Midas > dust,#2,77 = inputr(6)*m$sin(inputr(12))
```

Any algebraic expression using the operators +, −, ∗, `/` and parentheses (,) and constants as well as any *variable* explained above is supported by the command `COMPUTE/KEYWORD` (and direct assignments). However, since keywords are the "objects" of the right-hand expression in this command they don't have to be enclosed in curly brackets unless we need a conversion from character keywords to numerical data. If a keyword name is found as operand in the `COMPUTE/KEYWORD` command, automatically its content is taken (charac-

ter strings have to be enclosed in double quotes). Using curly brackets always and thus substitution at the parsing stage of the Midas command line introduces dependency on the SET/FORMAT command and thus the possible danger of loss of precision for numerical keywords.
But this is only true for keywords - all other variables like descriptors, table elements, etc. have to to be either converted to ASCII strings via the curly brackets (or accessed via the M$VALUE function)!

```
Midas > inputd = image,step(2)+outputd(5)
```

is WRONG. Whereas,

```
Midas > outputr(12) = outputr(1)-inputi(12)+{galaxy,step(2)})
```

is O.K.

Also the commands DO, IF and BRANCH operate on keywords or constants only. So again, the name of a keyword indicates already to use it's contents, whereas character strings have to be enclosed in double quotes.
Inside a Midas procedure the line:

```
if {inputi} .eq. 22 then
```

is not wrong, but:

```
if inputi .eq. 22 then
```

is better, since *inputi* is an integer keyword and needs therefore no conversion from character string to numerical value.
However, if e.g. parameter P3 contains a number, then

```
if p3 .eq. 22 then
```

is WRONG, instead:

```
if {p3} .eq. 22 then
```

has to be used, since the parameters are implemented as character keywords.

Let us look at another examples:

```
!+
!  Example 6, MIDAS procedure exa6.prg
!+
DEFINE/PARAM P1 ?  N "Enter alpha value:  " -88.5,912.4
DEFINE/PARAM P2 ?  N "Enter loop_count:  " 1,999
DEFINE/MAXPAR 2                              ! max. 2 parameters expected
WRITE/KEYWORD VAR/R/1/1 0.                   ! init key VAR
VAR = {P1} * 3.3                             ! see remark a
WRITE/DESCR myframe rval/r/1/2 0.0,0.0 !
LOOP:                                        ! declare label LOOP
VAR = 1.+VAR                                 ! see remark b
myframe,rval(2) = var+12.99
WRITE/OUT {myframe,rval(2)}
myframe[@10,@20] = 20.0-{myframe,rval(2)}
WRITE/OUT {myframe[@10,@20]}
```

```
mytable,:DEC,@7 = {myframe[@10,@20]}*2.0
WRITE/OUT {mytable,:DEC,@7}
WRITE/OUT " "
IF VAR .LE. {P2} GOTO LOOP ! go to label LOOP, if VAR ≤ contents of P2
```

Then the command `@@ exa6 1.0 5.2` will yield:

```
1.72900E+01
2.71000E+00
5.42000E+00

1.82900E+01
1.71000E+00
3.42000E+00
```

Remark A:
Parameter `P1` is implemented as a character keyword. Therefore, we have to use `{P1} * 3.3` in the expression to convert the character data in `P1` to numerical data (both operands must be numeric).
If we had written `VAR = P1 * 3.3`, MIDAS would have protested because in the righthand expression we would have tried to multiply a char. string with a float.

Remark B:
Here we use keyword `VAR` directly (i.e. without curly brackets) since ist's a float keyword, so both operands of the righthand expression are of numeric type.

Note

For character keywords `COMPUTE/KEYWORD` *only supports character concatenation ('//'). If you want to write a character string into a character keyword, use* `WRITE/KEYWORD` *instead.*

To recapitulate it you must use curly brackets with keywords in `COMPUTE/KEYWORD` if you need to convert a character keyword to a numeric constant - but no conversion problems (like loss of precision) here. This is, for example, the case when you want to use the contents of the (character) parameters `P1, ..., P8` in a numeric expression, e.g.:
`outputr(3) = {p1}*outputr(12)`
This line would be translated to:
`outputr(3) = 0026*outputr(12)`
in the first pass over the command line, so the right hand side contains a numerical constant and a keyword - a valid numerical expression.
However, the Midas command line:
`outputr(3) = p1*outputr(12)`

would involve a char. and real keyword in the evaluation of the right hand side and is therefore illegal syntax.

Since symbols may be tested in conditional statements and thus change the control flow of a MIDAS procedure, they provide the link between application programs and the MIDAS command language.
The number of characters used in the ASCII representation of a numerical symbol is controlled via the command `SET/FORMAT format` for integer symbols and `SET/FORMAT rformat,dformat` for real (rformat) and double (dformat) precision symbols. Similar to FORTRAN the I–format (with leading zeroes not suppressed) is used for integer symbols and real or double precision symbols use the E–format, G–format or F–format:

```
!+
!  Example 7, MIDAS procedure exa7.prg
!+
WRITE/KEYWORD INPUTI 12                          ! set INPUTI(1) to 12
WRITE/KEYWORD INPUTR 12.345                  ! set INPUTR(1) to 12.345
                             ! and set INPUTD(1) to 123456.98765432
WRITE/KEYWORD INPUTD 123456.98765432
WRITE/OUT {inputi(1)} {inputr(1)} {inputd(1)}
SET/FORMAT I2                       ! use format I2.2 for integer symbols
            ! and use format E12.8 and G22.8 for real and double symbols
SET/FORMAT E12.8,G22.8
WRITE/OUT {inputi(1)} {inputr(1)} {inputd(1)}
SET/FORMAT i5                       ! use format I5.5 for integer symbols
            ! and use format F12.4 and E22.13 for real and double symbols
SET/FORMAT f12.4,e22.13
WRITE/OUT {inputi(1)} {inputr(1)} {inputd(1)}
```

The command `@@ exa7` will yield:

```
0012 1.23450E+01 1.23457E+05              default is I4 and E15.5,E15.5
12 1.23450003E+01 1.23456988E+05             uses I2 and E12.8,G22.8
00012 12.345 1.2345698765432E+05             uses I5 and F12.4,E22.13
```

If you want to omit any leading zeroes for integer symbols use `SET/FORMAT I1`, then only the necessary digits will be displayed.

Note

Use `SET/FORMAT f-format` *to only change the format for real symbols and* `SET/FORMAT ,f-format` *to only change the format for double symbols.*

The format which is specified via the SET/FORMAT command must be set individually in the body of each procedure. For backward compatibility the default is `SET/FORMAT I4 E15.5,E15.5` (format I4 for integer values, E15.5 for real and double precision values).

This represents a hidden danger with numeric (especially double precision) data because you might loose precision. Consider the following example:

```
!+
!  Example 7a, MIDAS procedure exa7a.prg
!+
inputd = 1234.55667701
write/keyword outputd/d/1/1 {inputd}
set/format ,f20.10
write/out {outputd(1)}
```

When executing `@@ exa7a` you will get 1234.5601 instead of 1234.55677701 even though you used the *large* format f20.10 for the representation of keyword OUTPUTD. What happened? Well, let's show the substitutions, so we do
`Midas > echo/full`
`Midas > @@ exa7a`
and we get:

```
1 > inputd = 1234.55667701
1 > write/keyword outputd/d/1/1 {inputd}
    write/keyword outputd/d/1/1 1.23456E+03
1 > SET/FORMAT ,f20.10
1 > WRITE/OUT {outputd}
    WRITE/OUT 1234.5601
1234.5601
```

So keyword OUTPUTD got the contents of INPUTD formatted according to the default format for double precision values (e15.5), which yields 1234.560 ...

If we change to:

```
!+
!  Example 7b, MIDAS procedure exa7b.prg
!+
set/format ,f20.10
inputd = 1234.55667701
write/keyword outputd/d/1/1 {inputd}
write/out {outputd}
```

(and turn the echo off) we get from `@@ exa7b` the output 1234.55677701 as expected.

So it's good practice to always put `SET/FORMAT` into the begin of your procedure if you use substitutions.

However, the best would be to use no substitution at all via:

```
!+
!  Example 7c, MIDAS procedure exa7c.prg
!+
inputd = 1234.55667701
outputd = inputd
SET/FORMAT ,f20.10
WRITE/OUT {outputd}
```

Now, we get from `@@ exa7c` the output 1234.55677701 as expected and we don't have to worry about the `SET/FORMAT` command.

Substitution begins inside the curly brackets, starting at the deepest nested level:

```
WRITE/OUT {IN_A}{INPUTC(1:3)}
```

will display `SPIRALABC` on the terminal, if key IN_A contains the string `SPIRAL` and key INPUTC(1:3) the string `ABC`.
It is sometimes necessary to substitute symbols in a nested order:

```
!+
!  Example 8, MIDAS procedure exa8.prg
!+
DEFINE/PARAM P1 myframe IMA "Enter name for input frame:  "
SET/FORMAT F5.1
WRITE/OUT {{P1},STEP(1)}
```

the command `@@ exa8` will force the Monitor to substitute the last command line in `exa8.prg` first to: `WRITE/OUT {myframe,STEP(1)}` and then yield: `20.5`
assuming that descriptor `STEP` of `myframe.bdf` contains `20.5` as first element. This example also illustrates the concept of recursive substitution.

You may want to use symbol substitutions for sending the output of a MIDAS command to an ASCII file or to the terminal depending upon the contents of e.g., the character keyword `mykey`.
Setting keyword `mykey` once to ">outfile" or ">terminal" if you want output to a file or not together with the command line `WRITE/TABLE mytab {mykey}` in your procedure will *not* work!
For, the check for the output redirection is done at the very first parsing of the command line before any symbols in that line are replaced...
Instead, setting `mykey` to "outfile" or "terminal" and changing the command line to: `WRITE/TABLE mytab >{mykey}`
will do the intended switching of output to a file or terminal.

3.6.3 DO Loops

Loops are supported in MIDAS procedures like the `DO` loops in FORTRAN (but note that loops are always executed at least once):

```
!+
!  Example 9, MIDAS procedure exa9.prg
!+
WRITE/KEYWORD N/I/1/1 0                ! keywords serve as loop variables
DO N = 1 6 2                      ! loop from N=1 until N≤6 in steps of 2
  WRITE/OUT N = {N}
ENDDO
```

A keyword of integer type (called N in our example) must be used to store the loop variable. The parameters follow the standard FORTRAN conventions with start (=1 in `exa9.prg`), end (=6) and in/decrement (=2) values given as shown above. DO loops may be nested up to 8 levels deep in a procedure.

The command `@@ exa9` will yield

```
N = 0001
N = 0003
N = 0005
```

Assume we have images `imag0001.bdf` to `imag0100.bdf` and want to add successive pairs and store the results into images `res0001.bdf` to `res0050.bdf`:

```
!+
!  Example 10, MIDAS procedure exa10.prg
!+
DEFINE/PARAM P1 ?  IMA "Enter root_name for input frames:  "
DEFINE/PARAM P2 ?  IMA "Enter root_name for output frames:  "
DEFINE/MAXPAR 2                          ! max. 2 parameters expected
SET/FORMAT I4                                       ! we need 4 digits
WRITE/KEYWORD N/I/1/1 0
WRITE/KEYWORD NN/I/1/2 0,0
!
DO N = 1 50                                   ! default increment is 1
  NN(1) = 2*N
  NN(2) = NN(1)-1
  COMPUTE/IMAGE {P2}{N} = {P1}{NN(2)}+{P1}{NN(1)}            ! sum up
  LOAD/IMAGE {P2}{N}                         ! display the result frame
ENDDO
```

Then, the MIDAS command `@@ exa10 imag res` will do the required task.

3.6.4 Local Keywords

Because keywords are implemented as a global data structure, different MIDAS procedures can access the same keyword. This useful feature can cause problems, however, if these keywords just serve as local, temporary variables like the DO variables. Consider the procedures below:

```
!+
!  Example 11, MIDAS procedure exa11.prg
!+
WRITE/KEYWORD N/I/1/1 0
DO N = 1 10
  @@ test
ENDDO

!+
!  MIDAS procedure test.prg
!+
WRITE/KEYWORD N/I/1/1 0
DO N = 1 12
  WRITE/KEYWORD INPUTI/I/12/1 {N}
ENDDO
```

Executing `@@ exa11` will give some unexpected results, since both procedures access the same integer keyword N as a common variable.

Therefore, procedures should use *local* keywords for DO loops and internal working storage. Local keywords are defined inside a MIDAS procedure via the command `DEFINE/LOCAL`. They are only known inside the procedure where they are defined (if the `lower_levels_flag` is set, they are also defined in all procedures called from this procedure). Local keywords may have the same name as an existing global keyword (except the system keyword names as stored in `MID_MONIT:syskeys.dat`) or local keyword of any other procedure, since local keywords are searched before the global ones. The above example will work, if modified as follows:

```
!+
!  Example 12, MIDAS procedure exa12.prg
!+
DEFINE/LOCAL N/I/1/1 0
DO N = 1 10
  @@ test
ENDDO


!+
!  MIDAS procedure test.prg
```

```
!+
DEFINE/LOCAL N/I/1/1 0
DO N = 1 12
  WRITE/KEYWORD INPUTI/I/12/1 {N}
ENDDO
```

Local keywords are deleted when returning to the next higher level at the end of a procedure.

Note

Always provide all initial values to a keyword. For example, `define/local lola/r/1/6 0` *usually works as intended, i.e. all elements of keyword* `lola` *are filled with 0, but depending upon the computer and operating system the contents of lola(2,...,6) may be unpredictable. So, use* `define/local lola/r/1/6 0 all` *in order to fill all elements of keyword* `lola` *with 0.*

3.6.5 Conditional Statements, Branching

As in FORTRAN 77 any of the following forms of the `IF` statement may be used:

```
IF log_exp command                    ! command = any MIDAS command
                                      ! with at most 4 params.

IF log_exp THEN                       ! xyz = any logical expression
  ...
ELSEIF log_exp THEN                   ! uvw = any logical expression
  ...
ELSE
  ...
ENDIF

IF log_exp THEN
  ...
ENDIF
```

`IF` blocks may be nested up to 8 levels deep in a procedure.
The logical expression `'log_exp'` is of the form:

```
arg1  op  arg2
```

where *arg1, arg2* are either names of keywords (this includes also the names P1, ..., P8) or constants, and *op* may be any of `.EQ.`, `.NE.`, `.GT.`, `.GE.`, `.LT.` or `.LE.` (with the same meaning as in FORTRAN 77). As with symbol substitution, specify single elements of an array and substrings via, e.g., `OUTPUTI(7)` and `IN_B(2:15)`.
If we do

```
WRITE/KEYWORD INPUTC beer
WRITE/KEYWORD OUTPUTC wine
WRITE/KEYWORD INPUTR/R/1/3 1.,2.,3.
```

Then,

`INPUTC .EQ. "beer"`	is TRUE
`INPUTC .EQ. "BEER"`	is also TRUE
`INPUTC .EQ. OUTPUTC`	is FALSE
`INPUTC(2:2) .EQ. OUTPUTC(4:4)`	is TRUE
`INPUTR(2) .GT. 5.4`	is FALSE

In string comparisons upper and lower case characters are not distinguished in order to guarantee case insensitivity.
Character keywords can only be compared to character keywords or character constants (which are enclosed by double quotes). This can become tricky in conjunction with symbol substitution:

```
!+
!  Example 13a, MIDAS procedure exa13a.prg
!+
DEFINE/PARAM P1 ?  N "Enter number:  "
DEFINE/MAXPAR 1                         ! only one parameter expected
IF {P1} .EQ. 1 THEN
 WRITE/OUT P1 = 1
ELSE
 WRITE/OUT P1 is not = 1
ENDIF
```

Entering `@@ exa13a 1` as well as `@@ exa13a 001` will give the expected output message `P1 = 1` since the line `IF {P1} .EQ. 1 THEN` has been converted in the first pass by the Monitor to
`IF 1 .EQ. 1 THEN` or `IF 001 .EQ. 1 THEN`
and the two integer constants are equal. Now, consider the almost identical procedure `exa13b.prg`:

```
!+
!  Example 13b, MIDAS procedure exa13b.prg
!+
DEFINE/PARAM P1 ?  N "Enter number:  "
DEFINE/MAXPAR 1                         ! only one parameter expected
IF P1 .EQ. 1 THEN
 WRITE/OUT P1 = 1
ELSE
 WRITE/OUT P1 is not = 1
ENDIF
```

Entering `@@ exa13b 1` will return the error message `invalid IF statement...` and abort. Why?
Well, in the `IF` statement above the contents of the character keyword P1, which is the character '1', is compared to the integer constant 1, an invalid comparison.
We modify the procedure once more:

```
!+
!  Example 13c, MIDAS procedure exa13c.prg
!+
DEFINE/PARAM P1 ?  N "Enter number:  "
DEFINE/MAXPAR 1                         ! only one parameter expected
IF P1 .EQ. "1" THEN
 WRITE/OUT P1 = 1
ELSE
 WRITE/OUT P1 is not = 1
ENDIF
```

Now, entering `@@ exa13c 1` will work and yield `P1 = 1` but `@@ exa13c 001` will output `P1 is not = 1` since the string "001" is not equal to "1".

As another example let us see, how we can check if a parameter has been entered at all:

```
!+
!  Example 14a, MIDAS procedure exa14a.prg
!+
DEFINE/PARAM P6 + NUMBER "Enter first input number:  "
IF P6(1:1) .EQ. "+" THEN
 WRITE/KEYWORD INPUTC NONE     ! no P6 entered, set INPUTC accordingly
ELSE
 WRITE/KEYWORD INPUTI/I/7/1 {P6}    ! store contents of P6 in INPUTI(7)
 WRITE/KEYWORD INPUTC YES   !indicate, that INPUTI holds a valid number
ENDIF
```

However, if we also want to check the limits of the given number we have to use the `DEFINE/PARAMETER` command again, because testing "+" against a numerical interval would lead to an error:

```
!+
!  Example 14b, MIDAS procedure exa14b.prg
!+
DEFINE/PARAM P6 + NUMBER "Enter first input number:  "
IF P6(1:1) .EQ. "+" THEN
 WRITE/KEYWORD INPUTC NONE     ! no P6 entered, set INPUTC accordingly
ELSE
 DEFINE/PARAM P6 + NUMBER "Enter first input number:  " 22,1024
```

```
 WRITE/KEYWORD INPUTI/I/7/1 {P6}    ! store contents of P6 in INPUTI(7)
 WRITE/KEYWORD INPUTC YES   !indicate, that INPUTI holds a valid number
ENDIF
```

Since, in the ELSE branch we know that parameter P6 is given, the default value "+" itself is never tested against the interval [22,1024].

For testing multiple alternatives use the BRANCH command. It has the syntax:
BRANCH variable casea,caseb,...,casez labela,labelb,...,labelz.

```
!+
!  Example 15, MIDAS procedure exa15.prg
!+
DEFINE/PARAMETER P1 ?  C "Enter method:  "
DEFINE/MAXPAR 1                         ! only one parameter expected
!
!  Use the first 2 characters of parameter P1 to distinguish the methods
BRANCH P1(1:2) AN,DI,HY ANALOG,DIGIT,HYBRID
!
!  fall through if no match ...
WRITE/OUT Invalid option - please try again
RETURN
!
ANALOG:
RUN ANALO
RETURN
!
DIGIT:
RUN DIGI
RETURN
!
HYBRID:
RUN HYBRI
```

Then, `@@ exa15 ANALOG` will execute the command `RUN ANALO` and `@@ exa15 digital` or `@@ exa15 di` will run the program `digi.exe`.

3.6.6 Special Functions

Special functions may be used with the command COMPUTE/KEYWORD. The currently supported functions are listed in the following tables (on the next pages). Note, that *arg1, arg2* may either be the name of a keyword, the contents of which are used, or a constant. Character constants have to be enclosed in double quotes to distinguish them from a keyword name. On–line help for these functions is available via HELP COMPUTE/KEYWORD.

Command	Description
`M$ABS(`*arg1*`)`	returns the absolute value of integer/real/double *arg1* as integer/real/double
`M$EXIST(`*arg1*`)`	returns 1 or 0, if file *arg1* exists or not
`M$EXISTD(`*arg1*`,`*arg2*`)`	returns 1 or 0, if descriptor *arg2* of frame *arg1* exists or not
`M$EXISTK(`*arg1*`)`	returns 1 or 0, if keyword *arg1* exists or not
`M$EXISTC(`*arg1*`,`*arg2*`)`	returns the number of the column specified in *arg2* of the table *arg1*; returns -2 or -1, if table *arg1* or column *arg2* doesn't exist
`M$FILTYP(`*arg1*`,`*arg2*`)`	returns a type_no for file in *arg1*, if the file name does not include a file type, the type definition in *arg2* is appended to the file name
`M$INDEX(`*arg1*`,`*arg2*`)`	returns index of string *arg2* in string *arg1* as integer value (same as function INDEX of FORTRAN 77, i.e. first position is 1 (not 0))
`M$INDEXB(`*arg1*`,`*arg2*`)`	same as `M$INDEX` but search is done backwards, starting at the end of the string
`M$LEN(`*arg1*`)`	returns length of string *arg1* until the first blank or end of string, e.g. for argument "abcd" result=4 and for "abc de" result=3
`M$STRLEN(`*arg1*`)`	returns 'true' string length of *arg1* e.g. for argument "abcd" result=4 for "abc de" result=6; if the string contains only blanks, 0 will be returned
`M$NINT(`*arg1*`)`	returns nearest integer of real/double *arg1*
`M$SYMBOL(`*arg1*`)`	returns the translation of Unix environment variable *arg1* (or DCL symbol in VMS) as a character string
`M$LOWER(`*arg1*`)`	returns character string *arg1* in lower case
`M$UPPER(`*arg1*`)`	returns character string *arg1* in upper case
`M$TSTNO(`*arg1*`)`	returns 1 or 0, if string *arg1* is a number or not
`M$TIME()`	returns current date and time as string of 30 characters (as given by the host system)
`M$TIME(1)`	returns current date and time as string in the form `yyyy-mm-dd`
`M$SECS()`	returns the current time as no. of seconds elapsed since 1st Jan. 1970 (as an integer)
`M$AGL(`*arg1*`)`	returns contents of AGL definition file `agldevices.dat` related to *arg1* as an ASCII string
`M$LN(arg1)`	returns natural logarithm of real/double *arg1*
`M$LOG(arg1)`	returns base–10 logarithm of real/double *arg1*
`M$EXP(arg1)`	returns exponential of real/double *arg1* (base e)

Table 3.2: Special Functions available for operations on keywords

Command	Description
`M$SIN(arg1)`	returns sine of real/double angle *arg1* (angle in degrees)
`M$COS(arg1)`	returns cosine of real/double angle *arg1* (angle in degrees)
`M$TAN(arg1)`	returns tangent of real/double angle *arg1* (angle in degrees)
`M$ASIN(arg1)`	returns arcsine of real/double *arg1* in degrees
`M$ACOS(arg1)`	returns arccosine of real/double *arg1* in degrees
`M$ATAN(arg1)`	returns arctangent of real/double *arg1* in degrees
`M$SQRT(arg1)`	returns square root of real/double *arg1*
`M$VALUE(imag[x,y,z])`	returns a pixel of image *imag* as real value
`M$VALUE(imag,dscr)`	returns an element of descr *dscr* of frame *imag* according to the descriptor type
`M$VALUE(tbl,col,row)`	returns a single element of table *tbl* in column *col* and row *row* according to the column type

Table 3.3: Special Functions available for operations on keywords - cont.

Note, that the `M$VALUE` function is different from all other functions:
Firstly it has a variable no. of arguments and secondly, because its operands are not keyword names (or constants) but the real names of images and tables.

As an example we want to to display the header of a FITS file stored on disk (without converting the data), the FITS file name may be entered with or without the file extension `.mt`; if not given we append the type inside the procedure:

```
!+
!  Example 16, MIDAS procedure exa16.prg
!+
DEFINE/PARAM P1 ?  ?  "Enter FITS file name:"
!
DEFINE/LOCAL INA/C/1/80 " " all     ! that fills all elements of INA with
blanks
DEFINE/LOCAL K/I/1/2 0,0
!
K = M$INDEX(P1,".mt")                 ! test, if type of FITS file entered
IF K .LT. 2 THEN
  WRITE/KEYW INA {P1}.mt                        ! if not, append type
ELSE
  WRITE/KEYW INA {P1}                     ! if yes, no need to append type
ENDIF
```

```
INTAPE/FITS 1 midd {ina} fnn | $more
!                       but if we want to save the no. of axes of the FITS file
!                       we do not use the keyword name INA but it's contents
K = M$VALUE({ina},naxis)
WRITE/OUT we have {K} axes
```

The MIDAS commands `@@ exa16 test` as well as `@@ exa16 test.mt` will both display the header of the FITS file `test.mt`. Note, that this procedure will display the header in a user friendly way, i.e. one screen at a time (and only work for Unix).
One of the MIDAS verification procedures, `verify3.prg` shows the usage of all currently available functions. Enter `@ vericopy` to copy this procedure into your current directory (also the usage of `verify3` will be shown then).

3.6.7 Interrupting Procedures

Sometimes, it may be necessary to interrupt the execution of procedures. One way to do this is via the command `INQUIRE/KEYWORD` which was already discussed before; depending upon the user input the procedure could continue or stop. But while the procedure is waiting for input, MIDAS is blocked, no other command can be executed.
With the command `PAUSE` a procedure is stopped and saved; MIDAS returns to the interactive level and you can execute any other command. To resume the stopped procedure at a later time, enter `CONTINUE`. Then, the procedure continues with the next command after the `PAUSE` line. Only one procedure can be in the `'PAUSEd'` state at a time, in other words it is not possible to stop and save several procedures together.
As an example, consider the case where after some tricky operations on an image you want to get a grayscale copy of the result on a Postscript Laser printer. Since the grayscale plot is quite a time consuming operation you want to make sure that the frame is really o.k. before sending that job to the printer queue.

```
!+
! Example 17, MIDAS procedure exa17.prg
!+
DEFINE/PARAM P1 ? IMA "Enter input frame: "
DEFINE/PARAM P2 ? IMA "Enter output frame: "
DEFINE/MAXPAR 2                        ! max. 2 parameters expected
WRITE/KEYWORD IN_A {P1}
DEFINE/LOCAL MYRESULT/C/1/80 {P2}
RUN tricky.exe
PAUSE
!
INQUIRE/KEYWORD INPUTC "Result frame o.k.? Enter YES or NO: "
IF INPUTC(1:1) .EQ. "Y" THEN
  ASSIGN/DISPLAY LASER
  LOAD/IMAGE {MYRESULT}
```

```
ENDIF
```

With `@@ exa17 venus jupiter` the procedure will start the program `tricky` to operate on `venus.bdf` and produce the frame `jupiter.bdf`, and then it will stop. Now, you can check the result by e.g. calculating the statistics of `jupiter.bdf` or simply displaying it. Then, resume the procedure via `CONTINUE` and type `YES` if you are satisfied with the result and want the hardcopy or `NO` if not.
Note also, that we used a local keyword to hold the name of the result frame and not the usual keyword `OUT_A`. Thus, we are sure that the result name is not accidentally overwritten by another command which also uses `OUT_A`.

3.6.8 Entry points

It is sometimes desirable to group several related procedures into a single file. In MIDAS, the `ENTRY` command defines entry points for different procedures in the same file. These individual procedures are executed by specifying also their entry point besides the file name in the '`@@`' command.

```
!+
!  Example 18, MIDAS procedure exa18.prg
!+
DEFINE/PARAM P1 11 NUMBER "Enter input number:  "
DEFINE/MAXPAR 1                        ! only one parameter expected
WRITE/OUT "Parameter 1 = {P1}"
!
ENTRY 2
DEFINE/PARAM P1 new C "Enter input:  "
DEFINE/MAXPAR 1                        ! only one parameter expected
WRITE/OUT "Parameter 1 = {P1}"
!
ENTRY third
DEFINE/PARAM P1 spiral IMA "Enter input image:  "
DEFINE/MAXPAR 1                        ! only one parameter expected
WRITE/OUT "Parameter 1 = {P1}"
```

The string following the `ENTRY` command (max. 8 characters) is used in the '`@@`' command to select the code segment in the file `exa18.prg`. Thus, `@@ exa18,2 old` will result in the display of the line: '`Parameter 1 = old`'; the following `ENTRY` statements indicates the end of this code segment and acts like a `RETURN` statement. Entering `@@ exa18,third` produces the output: '`Parameter 1 = spiral`'; and `@@ exa18 -12` will execute the lines with no preceding `ENTRY` statement, i.e. write: '`Parameter 1 = -12`'. This example also

shows that parameter P1 is not global, that means it has to be defined in each ENTRY segment of the procedure file.
Entries may also be used to structure the contents of a MIDAS procedure. In the following example, the procedure exa18.prg executes different code segments according to its first parameter.

```
!+
!  Example 19, MIDAS procedure exa19.prg
!+
DEFINE/PARAM P1 000 C "Enter control flags for entries:  "
DEFINE/PARAM P2 sombrero IMA "Enter image to work with:  "
DEFINE/MAXPAR 2                              ! max 2 parameters expected
!
DEFINE/LOCAL LOOP/I/1/1 0
DEFINE/LOCAL CCC/C/1/3 {P1(1:3)}
SET/FORMAT I1
DO LOOP = 1 3
 IF CCC({LOOP}:{LOOP}) .EQ. "1" @@ exa19,000{LOOP} {P2}
ENDDO
!
!  here the different sub-procedures
!
ENTRY 0001
!
CREATE/IMAGE {P1} 2,256,256 ?  gauss 128.5,128,128.5,128
!
ENTRY 0002
!
READ/DESCR {P1}
!
ENTRY 0003
!
STATIST/IMAGE {P1}
```

Then, to read the standard descriptors of image frame luna.bdf we would enter the command @@ exa19 010 luna; to create the frame sol.bdf we enter @@ exa19 100 sol. Finally, in order to create a gaussian image estrella.spc, and read its standard descriptors and do the statistics on the newly created image, we type the command
@@ exa19 111 estrella.spc.

3.7 Context Levels

Besides the fixed (general) MIDAS commands, the user may dynamically create new commands any time during a MIDAS session. Context files provide a way to group commands which relate to a specific reduction sequence or application package.

For example, the command `SET/CONTEXT applic1` will execute the MIDAS procedure `applic1.ctx`, which would contain all the new command definitions for the application package `applic1` as well as any new keyword definitions and default settings.
Each enabled context has a corresponding context no. which links new commands to the context in which they were created. The command `SHOW/COMMANDS` displays all additional MIDAS commands together with their context no. The context no. 0 is used for all commands which are created outside a given context.

Once the user has finished his/her data reduction with `applic1`, he/she may want to work with package `applic2` on some of his/her data as well. One could either add all commands of `applic2` on top of the ones from `applic1` or first remove all the commands from the currently enabled context, i.e. `applic1`, in one go via the command `CLEAR/CONTEXT`.
`SET/CONTEXT applic2` will then create all the new commands of the package `applic2`.

Use `SHOW/CONTEXTS` to display all the currently enabled contexts.
Up to 8 different contexts may be enabled at any time (assuming that all enabled commands fit in the MIDAS command table).
Some of the currently available contexts (application packages) are:

`astromet`	Astrometric Package
`ccdred`	Reduction of CCD data
`ccdtest`	Test of reduction of CCD data
`cloud`	Model for absorption lines
`daophot`	Object detection and classification, the DAOPHOT-2 package
`do`	Data organizer for astronomical observations
`echelle`	Reduction of echelle spectra
`echellec`	Reduction of echelle spectra - old (94NOV) context
`exsas (*)`	Analysis of X-ray data from the ROSAT satellite
`feros`	Reduction of the FEROS instrument on La Silla
`geotest`	Utilities to create geometric test frames and other artificial images
`imres`	Programs related to image restoration
`invent`	Object detection and classification, the INVENTORY package
`irac2`	Reduction programs for the IRAC2 camera
`irspec`	Reduction of IRSPEC spectra
`iue (*)`	IUE-tape reader
`long`	Reduction of long slit spectra
`lyman`	Package for multiple-components fitting of interstellar absorption lines
`mos`	Package for multiobject spectroscopy
`mva`	Package for multivariate data analysis

`optopus`	Package to prepare observations with the Optopus facility at La Silla
`pepsys`	Photometric planning and reductions (extinction correction + transformation to std. system)
`pisco`	Reduction package for data obtained with the PISCO instrument at La Silla
`romafot`	Photometric extraction package, the ROMAFOT software
`spec`	Package for 1-dim spectra
`statist`	Statistical tests on tables
`surfphot`	Deconvolution and rebinning
`tsa`	Package for analysis of astronomical time series
`wavelet`	Image processing tools using the wavelet transform

For example, the command `SET/CONTEXT invent` will activate the commands related to the `INVENTORY` photometric package.
Klicking on the button `Contexts..` in the Midas Help GUI or typing the command `HELP [CONTEXT]` will display all currently available contexts at your site.

Note

The contexts `exsas` *and* `iue` *have been developed at the Max Planck Institute for Extraterrestrial Physics in Garching, Germany and ESA Vilspa, Villafranca Satellite Tracking Station, Spain, respectively. They may be obtained on request from these institutions.*

3.8 Running a Program within MIDAS

To execute a user–written MIDAS application program (coded in FORTRAN or C), employ the command `RUN`. The command `RUN MYPROG` or `RUN myprog` will execute `myprog.exe` in a subprocess like any other MIDAS command.
It is better practice to embed the command `RUN MYPROG` in a MIDAS command procedure. Typical tasks of this procedure would be to provide default values for all parameters, to check the validity of parameter values, and to store the parameters into the keywords your program will use.
Let us assume you have written your special filter program and stored the executable module as `bestfilt.exe` on disk. Program `bestfilt` just needs the names of the input and output image which are obtained inside the program from the keywords IN_A and OUT_A .
The following MIDAS procedure:

```
!+
!  MIDAS procedure bestfilt.prg
!+
CROSSREF INPUT RESULT
```

```
DEFINE/PARAMETER P1 ?  IMA "Enter input frame:  "
DEFINE/PARAMETER P2 ?  IMA "Enter output frame:  "
DEFINE/MAXPAR 2                          ! max. 2 parameters expected
!
WRITE/KEYWORD IN_A {P1}
WRITE/KEYWORD OUT_A {P2}
RUN BESTFILT                                      ! .exe is the default type
```

will check, that the two parameters are valid MIDAS file names and prompt for input if any parameter is not given. Together with the MIDAS command

```
CREATE/COMMAND BESTFILT/IMAGE @@ bestfilt
```

your application will then be integrated smoothly into MIDAS.
Now, you can apply your own filtering algorithm to the image `lobo.bdf` by typing e.g. `BESTFILT/IMAGE res=perrito in=lobo`.
As with data files you can specify a search path for executables via the command `SET/MIDAS_SYSTEM EPATH=directory` which is then used by the `RUN` command. Furthermore, if your module is written in C you can pass parameters to the executable in the usual way (*argc, arv* stuff) via, e.g.
`RUN BESTFILT par1 par2 ...`

If you want to execute Midas commands from a shell script you can do so as shown in the following example, where we use the procedure `myproc.prg` and Bourne shell script `myscript` to obtain the median value of an image.

```
!
!  MIDAS procedure myproc.prg
!  to return the median of an image in an ASCII file
!  execute via '@@ myproc image_name'
!
define/param p1 ?  ima "Enter name of image:"
!
statistics/image {p1}
!
!  save median of statistics command (was stored in keyword OUTPUTR(8))
!
define/local value/r/1/1 {outputr(8)}
define/local fc/i/1/2 0,0
!
!  create new ASCII file (named 'info.asci') and store median in there
!
open/file info.asci write fc
write/file {fc(1)} median = {value}
close/file {fc(1)}
```

```
!
bye                                                     ! terminate Midas
```

```
#! /bin/sh
# Bourne shell script 'myscript' - execute via:  $ myscript parameter
#
# delete last file we got
rm -f info.asci
#
# start up Midas in another xterm window
# and let it execute the procedure 'myproc.prg'
# shell parameter $1 will be passed as parameter P1 to Midas
#
xterm -e inmidas -j "@@ myproc $1" 33 -p
#
# the info we want is now in file:  info.asci
#
more info.asci
```

There is also the possibility to run any Midas command (procedure) directly from the Unix command line via the *drs* command. To have it available you must do once

`$ source /midas/`*version*`/system/unix/drs_config`

or

`$ sh /midas/`*version*`/system/unix/drs_config.sh`

then, assuming you have a FITS file `fors1data.fits` in your directory, you type

`$drs read/descr fors1data.fits "eso.*"`

to get all the ESO hierarchical FITS keywords of the image `fors1data.fits`.
Note, that the descriptor pattern 'eso.*' has to be enclosed in double quotes to avoid expansion of this term by the shell.
With

`$ drs -h`

you get a short help about the command.

Note

The `drs` *command is currently only implemented for Unix (Linux) systems, furthermore, it cannot be applied to Midas commands which use a graphics or display window. Also, Midas keyword manipulations don't make much sense, since the keyword data base is NOT saved...*

3.8.1 Debugging of Procedures and Modules

Normally, the command lines of a MIDAS procedure are not displayed on the terminal. To control the display of the lines of a MIDAS procedure, use the command `ECHO`. With

ECHO/ON the lines of a MIDAS procedure are displayed on the terminal as they are read from the file and executed. This way, it is possible to get an impression of how much time various parts of a procedure need.
With ECHO/FULL the lines are displayed as they are read and if symbols have to be substituted, the lines are again displayed after substitution. To avoid echoing and return to a *silent* mode, enter ECHO/OFF.
The ECHO command has as parameter the procedure-level-interval where it should be applicable. Thus you can, e.g., display only the lines of a MIDAS procedure executing at level 2, etc. Echoing each command line of a MIDAS procedure will identify most of the syntax and other obvious errors. However, this may not be sufficient for long and complicated procedures.
For these cases use the Midas Command Language Debugger:

```
DEBUG/PROCEDURE levla,levlb ON/OFF          !en/disable procedure debugging
DEBUG/MODULE levla,levlb ON/OFF          !en/disable module (F 77, C) debugging
SHOW/CODE comnd/qualif                   !display the code of related procedure
```

Once procedure debugging is switched on, e.g., via DEBUG/PROC 1,3 ON, all MIDAS procedures executing at level 1, 2 or 3 start up in stepwise debugging mode. The prompt changes to Mdb and each command line is displayed on the terminal, and only executed when you hit Return. Furthermore, a set of basic debugging commands may be executed, e.g. listing the preprocessed procedure code, setting and clearing break points, and switching from stepwise to continuous mode. Also the keywords may be inspected at any moment. This is an important tool because local keywords cannot be checked otherwise; once the procedure terminates, all local keywords disappear.
If you want to execute any other command, enter PAUSE to interrupt the procedure you're debugging, execute any other command you want, and enter CONTINUE to continue debugging the procedure.
When you are in the debugger (indicated via the Mdb prompt), use the command 'h' (for HELP) to display all the available debug commands.
To switch the debugging mode for procedures off, use DEBUG/PROC 1,3 off.

If you must debug your application program, first compile and link that program with the debugger of your host system. Make sure, that this is the same debugger as the one stored in a system keyword of MIDAS (via the command SET/MIDAS_SYSTEM debug=...). Enter the command DEBUG/MODULE to switch on the debugging mode for applications. Subsequently, your application (as well as all other programs activated via the MIDAS RUN command) will be started with the debugger of your system and you can debug it in the usual way.

Note

Typing $dbx myprog.exe *(e.g. on a SUN) would also start up program* myprog.exe *in debug mode. But that would not tie the application into the MIDAS environment, i.e. the keywords would not be set correctly.*

If you just want to list the preprocessed code of a MIDAS procedure use the command `TRANSLATE/SHOW proc`. `TRANSLATE/SHOW proc X` will also check all commands in the procedure for completeness, so it's a good idea to execute that command for all your MIDAS procedures.
The command `SHOW/CODE comnd/qualif` will display the code of the procedure which is actually executed when you enter `comnd/qualif` as a MIDAS command.

Note

For a detailed description of the integration of user applications into MIDAS as well as complete examples (in FORTRAN and C) see the MIDAS Environment Document.

3.9 Catalogs in MIDAS

MIDAS catalogs are best described as a list/collection of one of the supported data structures, e.g. Images, Tables or Fit files. Catalogs are implemented as ASCII files with the file type `.cat`. A MIDAS catalog has entries either for images or tables or fit files currently existing in MID_WORK and is then referred to as an Image, Table or FitFile catalog, accordingly. If a catalog is enabled (via `SET/ICAT, SET/TCAT, SET/FCAT`), new entries are added automatically whenever new frames are created, otherwise entries in a catalog have to be explicitly created via `ADD/ICAT`, `ADD/TCAT`, etc.
Frames that are listed in a catalog may then be referenced by their name, as well as via `#n`, if `n` is the entry_no. of the frame in the currently enabled catalog, or `#n,cat_name` if the entry is in catalog `cat_name.cat` (only one catalog of each type can be enabled at any one time).

The following commands are related to the use of catalogs:

`CREATE/xCAT` *cat_name dir_specs* `x = I, T, F` for images, tables, fit files
create catalog *cat_name* for images/tables/fit files, using *dir_specs*,
which are the options of the host commands `DIRECTORY` (VMS) or `ls` (Unix).

`SET/xCAT` *cat_name*; `CLEAR/xCAT`
enable/disable automatic addition of entries for images/ tables/fit files in MID_WORK to catalog *cat_name*

`READ/xCAT` *cat_name low,hi*
display all entries of image/table/fit file catalog *cat_name* within given range *[low,hi]*

`ADD/xCAT` *cat_name frame_list*
add image/table/fit file entry(ies) specified in *frame_list* to catalog *cat_name*

`SUBTRACT/xCAT` *cat_name frame_list*

remove entry(ies) from image/table/fit file catalog *cat_name*

`EXECUTE/CATALOG` *proc P1 P2 ... P7*
execute the MIDAS procedure *proc* which was written for a single frame
for all frames in a catalog.

`WRITE/SETUP CATALOG`
setup the necessary keywords for an `EXECUTE/CATALOG` command

3.9.1 Using Catalogs in MIDAS Procedures

Assume we have written a specific application program, `pearl`, within the MIDAS environment, that processes an input image and produces some numbers as a result. We would like this program also to work on a sequence of images, not just on one input image:

```
!+
! Example 20, MIDAS procedure exa20.prg
!+
DEFINE/LOCAL CATAL/I/1/1 0                 ! define local key CATAL
!
LOOP:
STORE/FRAME IN_A {P1}                ! fill key IN_A with parameter 1
RUN pearl                                  ! run our application
GOTO LOOP
```

If P1 contains the name of an image, the command `STORE/FRAME` works exactly
like `WRITE/KEYWORD`. The keyword CATAL is not modified.
However, if P1 contains the name of a catalog of the form *file.cat*, this catalog is opened and the first entry in the catalog is stored into the keyword IN_A.
The number of the next entry is saved in the keyword CATAL (so this name is fixed!). In the loop, the entry number is taken from CATAL and the corresponding frame name put into keyword IN_A (in our example). If there are no more entries in the catalog, control is either transferred to a label which may be specified in the command line of `STORE/FRAME` or if not given, the procedure is terminated.
So `@@ exa20 myframe` will work on the single frame `myframe`,
whereas `@@ exa20 mycatal.cat` works on all frames with entries in the image catalog `mycatal.cat`.

If the program `pearl` produces also output frames, you should not have the catalog enabled (cf. `SET/ICAT` command). Because each new frame gets added to the enabled catalog and you would end up with an infinite loop!

Furthermore you may write your application and procedure just to work on single frames and then execute this procedure on all frames in a catalog via the command `EXECUTE/CATALOG`.

Note, that you have to set up some special keywords in advance for that via WRITE/SETUP CATALOG; for details see the HELP of EXECUTE/CATALOG.

3.10 Adapting MIDAS to your personal needs

There are commands in MIDAS which help you in tailoring MIDAS to your personal taste and needs.
Maybe the most important one is CREATE/COMMAND with which you can add abbreviated and alias command names. As a next step, try out CREATE/DEFAULTS in order to set up your own defaults for frequently used commands, e.g. size and location of display and graphics windows in an X11-environment. The command SET/MIDAS_SYSTEM has an extended set of options to let you change internal MIDAS features, ranging from selecting your preferred text editor (to be used e.g. in REPORT/PROBLEM) to choosing your own MIDAS prompt. With the command SET/BUFFER you modify the size of the internal command buffer.
If you have a MIDAS command procedure named 'login.prg' in the directory specified by MID_WORK, this procedure will be automatically executed whenever you get into the MIDAS environment, i.e. when you type INMIDAS (inmidas) or GOMIDAS (gomidas). Therefore, the procedure login.prg is the place where you should put all the commands needed to adapt MIDAS.

```
!+
!  MIDAS procedure login.prg
!  personal set up file for A. S. Tronomer       940815
!+
CREATE/COMMAND RK READ/KEYWORD                  !define abbreviations
CREATE/COMMAND WK WRITE/KEYWORD
CREATE/COMMAND RD READ/DESCR
CREATE/COMMAND WD WRITE/DESCR
CREATE/COMMAND XH CREATE/GUI HELP
CREATE/COMMAND SMOOTH/SPECIAL @@ mysmooth     !define a new command
!
CREATE/DEFAULT CREATE/GRAPH ? 400,800        !size for graphic window
CREATE/DEFAULT CREATE/DISP ? 600,600,400,400     !size+loc for display
window
!
SET/MIDAS_SYS edit=vi user=user env=midhost
SET/MIDAS_SYS prompt=Mid{mid$sess(11:12)}
```

Assuming you are working with MIDAS unit 22, this procedure will change the MIDAS prompt to Mid22, use 'vi' as editor when you run the REPORT/PROBLEM command, define the commands RK, WK, RD, WD, XH, SMOOTH/SPECIAL, and override the preset defaults for CREATE/GRAPHICS, CREATE/DISPLAY. Setting the environmment to *midhost* means, that all commands which MIDAS does not understand are passed on to the host system.

Thus, instead of `$ls` you can directly enter `ls` at the MIDAS prompt. Also, the user level is set to `USER`; cf. the following section.

3.11 MIDAS User Levels

Three different user-levels are maintained within the MIDAS system.

```
NOVICE (beginning MIDAS user)
USER (normal MIDAS user)
EXPERT (expert MIDAS user)
```

These user-levels are set via the command `SET/MIDAS_SYS user=level`,
e.g. `SET/MIDAS_SYS us=EXPERT`.
The level NOVICE is the default level assigned to you when getting into MIDAS.
The MIDAS user-level affects the terseness of the error messages.

Furthermore, EXPERT users may create commands which override already existing MIDAS commands (be careful ...).
All other users may only create *new* commands.

Note

Currently, there is no difference between the level NOVICE and USER, but this may change in a future release of MIDAS.

Chapter 4

Data Structures

This chapter will contain information on the data structures used in MIDAS such as `keys` (`keywords`), `descriptors`, `frames` (`bulk data frames`), `catalogs`, and `tables`.

At the present moment we refer the reader to the previous chapter 3 which gives a basic description of data structures used.

Chapter 5

Table File System

This Chapter describes the structure and use of tables in the MIDAS system. Section 5.1 is a general introduction. The table structure is outlined in section 5.2. Different functional aspects of the tables are described in sections 5.3 (Input/Output), 5.4 (Management) and 5.5 (Operations). A review of the commands is given in section 5.6, a more detailed explanation is included in appendix A. Section 5.7 describes format files to control input/output operations. Finally, section 5.8 contains an example that can be run at a terminal with graphic capabilities as `TUTORIAL/TABLE`.

5.1 Tables in Image Processing

The purpose of image processing systems is to extract information from image data. Systems which are designed to treat large numbers of images must also be able to analyse the data extracted from the images. The standard Database Management Systems (DBMS) provide many of the facilities needed; however, some of the desired interactive graphics, statistics, and mathematics are not always available. Therefore, a dedicated table system has been made to serve these purposes in MIDAS (Grosbøl and Ponz, 1985, *Mem.S.A.It.*, **56**, 429).

The use of tables can be divided into three main categories: internal, external, and user applications. One of the advantages of the MIDAS Table System (MTS) is that tables for these different purposes have the same structure and can be treated with the same set of routines. The three categories are discussed separately, although there is some overlap in the applications.

Internal tables are mainly used by MIDAS to compute transformations which later will be applied to images. Typical examples are dispersion relations, characteristic curves and coordinate transformations.

External tables: During a reduction procedure, data from external catalogs or data base may be needed. This information can be made available by transferring them to the MTS format. Examples are given: catalogs of photometric data which can be used to establish transformations from internal magnitudes to a standard photometric system, or astrometric catalogs for computations of accurate reference frames for images.

User tables are used for storage of values computed during the reduction (e.g. stellar magnitudes, line intensities, or isophotal diameters of galaxies). This provides an easy way to save such heterogeneous data in a computer readable format. Further, the user can investigate the properties of the data (e.g. distributions and correlations of different values, and so on).

5.2 Structure of Tables

Table data are arranged in columns and rows, and stored in MIDAS files with the extension `.tbl`. The entry at a given row and column may be either a single value or an array. The items in one row may describe different properties of the same object or feature. All elements in a given column must be of the *same* type and thus be associated with the *same* property. For instance, a table with stellar data could contain the following items in each row: identification, right ascension, declination, magnitude, and spectral type. The first column would then contain all the stellar identifications, the second the right ascensions, etc.

The supported column data types are numerical data (8/16/32-bit integers or 32/64-bit reals) and character strings.

Each column is tagged with a user-defined label, a display format and optional physical units and can be referred to either by its absolute number or its label.

An item in a table is accessed by giving its column and row in addition to the table name. The row number can either be given as an absolute value (i.e. the sequence number) or indicated by the value in a previously defined reference column.

In addition to the normal columns all tables contain a `SELECT` and a `SEQUENCE` column:

- The `SELECT` column enables the user to define and work with a subset of his table by flagging the rows that satisfy a selection criteria. The subtable will be used by commands that do not modify the table information whereas the selection flag will be reset by commands that modify table information and this before taking any action. The values of this column can be accessed in the `COMPUTE` command by using the name SELECT (short form SEL).

- The `SEQUENCE` column contains the sequence number of each row. The values of this column can be accessed in the `COMPUTE` command by using the name SEQUENCE (short form SEQ).

If an element is not defined it will be a `NULL` entry and will be listed as a "*" for all data types except for character strings. In that case it will be listed as an empty field.

The tables may be physically stored on disk in two formats: by records corresponding to the natural way of storing sequentially the rows and transposed , where all the values of a given column are stored together(default mode). A table can be always expanded in the sense that its number of columns and rows is automatically increased when the allocated space is exceeded.

5.3 Input/Output of Tables

The exchange of table data to and from the MTS is mainly done through standard ASCII files. This makes it easy for any program to get data from the MTS and to transfer data into it. Thus, output files from text editors and Database systems containing table data in a fixed format can directly be transferred into the MTS format.

Conversion between the ASCII data file and the table is defined by a format file (see section 5.7). If the format file does not exist, the conversion is done automatically via list-direct input in free format. In this case only REAL*4 and integer data, without `NULL` values are allowed.

Command	**Description**
`CREATE/TABLE`	Convert from ASCII files to MTS format
`PRINT/TABLE`	Transfer MTS information to the ASCII file assigned as output. The printer is used by default.

Table 5.1: Conversion between ASCII Files and MIDAS Tables

Normally, table files should be copied to magnetic tape in the FITS format for tables (Harten et *al.*, 1985, *Mem.S.A.It.*, **56**, 437) to make it easy to read them again on other computers. The conversion to FITS is done by the MIDAS command `OUTTAPE`; FITS tables are loaded onto disk by the command `INTAPE`.

5.4 Management of Tables

The management of tables is divided into four tasks: *defining, displaying, modifying* and *interactive editing* of tables. The commands that define or modify a table will update its descriptor HISTORY. As the length of this descriptor is limited, if you are doing a lot of operations on the same table, you may get a descriptor overflow. In that case you can turn off the automatic addition of history_lines by adding an integer descriptor named HISTORY_UPDA to the table and setting it to 0.

5.4.1 Definition of Tables

External tables are created as described in the previous section and the definition of their content is taken from the format file specified. To *create* a user table one can also use the `CREATE/TABLE` command by giving `NULL` as input. This will create a table of the specified size where all elements are `NULL`. Columns in a table can be created or deleted by using the commands `CREATE/COLUMN` and `DELETE/COLUMN`. The available commands are collected in Table 5.2. Some commands use internal tables to store results. In such cases the tables will be created and defined by the system according to defaults. Labels, display formats and units in an existing column are modified by the `NAME/COLUMN` command.

Command	Description
`CREATE/TABLE`	Create a table with specified size.
`CREATE/COLUMN`	Create a column.
`DELETE/COLUMN`	Delete column(s).

Table 5.2: Commands to Define Tables

5.4.2 Displaying Tables

Both the table parameters and the elements values can be *displayed.* The former are shown using the `SHOW` command Table values are listed out by the `PRINT` and `READ` commands, the output formatting being done using the display format associated with each column. Supported formats are Fortran-77 standard formats and special display formats to accommodate sexagesimal and time values. Finally table values can be plotted on a graphic device or display unit using the `PLOT` or `OVERPLOT` and `LOAD` command respectively. A list of these commands is given in Table 5.3.

Command	Description
`SHOW/TABLE`	Show table characteristics
`PRINT/TABLE`	Print elements in table
`READ/TABLE`	Read elements in table and display them on the terminal
`PLOT/TABLE`	Plot table elements on graphic device
`OVER/TABLE`	Plot table elements on top of a previous plot on the graphic device
`LOAD/TABLE`	Load table elements on the overlay plane of the display

Table 5.3: Commands to Display a Table

5.4.3 Modification of Tables

Elements in a table can be *inserted, changed*, and *deleted.* These functions are all performed by the `WRITE/TABLE` or `COPY` commands (See Table 5.4). The element to be modified must be defined by giving its column and row location. An element is deleted if the value is set to `NULL`. A whole row is considered deleted if the element in the reference column is `NULL`. The data type of a column cannot be changed once the column has been created. However, the command `COPY/TT` can be used to copy and convert the values of a column of a certain type into a column of an another type.

It is possible to *define a "subset"* of a table by the `SELECT` command. All commands that do not change a table element will only use the subset selected. By selecting `ALL` the whole table is selected.

Command	Description
WRITE/TABLE	Write value into a table element.
COPY/KT	Copy a keyword into a table element.
COPY/TK	Copy a table element into a keyword.
COPY/TT	Copy columns values into another column.
COPY/TI	Transform the format of the file from table into image.
COPY/IT	Transform the format of the file from image into table.

Table 5.4: Commands to Modify a Table

It is also possible to *transfer* data from one table to another. The four commands described in Table 5.5 can be used. *Interactive identification* of table entries is done with

Command	Description
COPY/TT	Copy all selected elements with identical reference values.
COPY/TABLE	Copy all selected elements from one table into another.
MERGE/TABLE	Merge common columns in several tables.
PROJECT/TABLE	Copy a set of columns from one table into another.

Table 5.5: Commands to Transfer Table Data

the command `IDENTIFY/xxx`, where `xxx` is `CURSOR` for the image display and GCURSOR for the graphic screen.

5.4.4 Interactive Editing of Tables

An interactive editing facility `EDIT/TABLE` exists in MIDAS to *create* and *modify* tables. The editor works in a "page–oriented" form, a "page" consisting of 20 rows and several columns to fill the screen format, using a Keypad mode or a command mode to perform the editing functions. The command mode is accessible by hitting CNTL-Z. Most of the editing functions are implemented on the right keypad of the keyboard (Table 5.2) as well as on the left keypad if it exists (e.g on Sun workstations). Some keys of the central keyboard are also recognized (Table 5.7). The functions only available in command mode are listed in Table 5.6

Command	Description
`EXIT`	to finish the editing session and produce the edited table
`QUIT`	to finish the editing session without producing the output table

Table 5.6: Table Editor COMMAND Functions

Command	Description
`Tab`	put the cursor in the next column field.
`Return`	next row.
`Delete`	delete previous character.
`Backspace`	move the cursor to beginning of line.
Cursor Arrows	move the cursor $\uparrow$, $\downarrow$, $\leftarrow$, or $\rightarrow$.

Table 5.7: Layout of the Table Editor Central Keypad

Page (L1) **Command**	Next (L2) **Find**
Advance (L3) **Bottom**	Backup (L4) **Top**
Right P (L5) **Left P**	Crea Col (L6) **Del Col**
Word (L7) **Show**	Change (L8) **Sort**
Line (L9) **Screen**	Row (L10) **Status**

Gold (F1)	H Functions (F2) **H Keypad**

Figure 5.1: Layout of the Table Editor Left Keypad

Page (KP 7) **Command**	Section (KP 8)	Right P (KP 9) **Left P**
Advance (KP 4) **Bottom**	Backup (KP 5) **Top**	Crea Col (KP 9) **Del Col**
Word (KP 1) **Show**	Eol (KP 2)	Change (KP 3) **sort**
Line (KP 0) **Screen**		Row (KP .) **Status**

Gold (F1)	H Left Key (F2) **H right Key**

Figure 5.2: Layout of the Table Editor Right Keypad

5.5 Operations on Tables

In this section we describe several of the mathematical operations that can be performed on table data. More specialised topics are described in Chapter 8, "Fitting of Data", and in Chapter 11 Vol B, "Multivariate Analysis Methods".

Arithmetic operations between columns in a table are done with the command `COMPUTE/TABLE`. The selection flag is reset by this command. Special functions, named according to the FORTRAN mathematical library, are also available.

Simple statistical descriptors are displayed with the command `STATISTICS/ TABLE`. These descriptors are stored in output keywords for further usage in a procedure.

A set of histogram-related commands, with qualifier `HISTOGRAM`, allow the graphic display of the histogram of a column (`PLOT` and `OVERPLOT` commands), the printout of histogram values (`READ` and `PRINT` commands), or the generation of a 1D image with the histogram of a column (command `COMPUTE/HISTOGRAM`).

Linear or polynomial fits in one or two dimensions can be performed on table columns with the command `REGRESSION`, qualifiers `LINEAR` and `POLY` respectively. The coefficients and error estimations are kept in output keywords that can be stored as table descriptors with the command `SAVE/REGRESSION`. Fitted values are calculated with `COMPUTE/REGRESSION`.

A topic of special interest is the generation of table data from an image and vice versa. A transformation was already described in section 5.4.3, and consists in copying the data from one format to the other, using the commands `COPY/IT` and `COPY/TI`. Tabular data can be converted into 1D or 2D image data with the command `CONVERT/TABLE`. This command works in several modes controlled by a parameter. In all cases the sampling domain of the result is defined by a reference image. The modes currently available are : POLY (polynomial fit to table data), SPLINE (spline approximation), PLOT (scattergram of the data in the table) and FREQ (2D histogram of the data in the table).

For more specific resampling and interpolating algorithms, the commands `REBIN` and `INTERPOLATE` will provide full conversion between image and table formats (qualifiers `TT, TI, IT` and `II`).

5.6 Command Overview

In this section we include a short description of the table commands in alphabetical order, (see the Detailed Command Description in Volume C of the MIDAS User Guide for a more detailed explanation).

Reference to tables is done by the filename. The extension `.tbl` must be appended to the filename in commands which can work both on images and tables (e.g.: `READ/DESCRIPTOR table.tbl`).

Reference to columns can be done either by "name" or by "number". Columns are referred to by name as `:label`, where `label` is a character string (Note the starting colon ':' in front of '`label`'.) The string (max. 16 characters, case insensitive) should start with a letter and may contain alpha-numeric characters and the underscore symbol.

Command	Description
COMPUTE/TABLE	Compute numeric expression of columns.
COMPUTE/HISTOGRAM	Compute column histogram, result in table or image format.
REBIN	Resampling data in table/image formats.
INTERPOLATE	Spline interpolation of data.
REGRESSION/LINEAR	Compute linear regression.
REGRESSION/POLY	Compute polynomial fit.
SAVE/REGRESSION	Store regression coefficients as table descriptors.
COMPUTE/REGRESSION	Compute fitted values using the regression coefficients.
STATISTICS/TABLE	Simple statistics on a table column.

Table 5.8: Operations on Table Data

Columns are referred by number as `#n`, where `n` is the integer defining the column position.

Access to rows can be done in two modes, "sequential" or "direct".

- Sequential access is defined by the row number as `@n`, where `n` is an integer constant.
- Direct access is done through the values in the reference column.

5.6.1 List of Commands

Table 5.9 contains a list of table commands.

Other table related commands are described in Chapter 8, "Fitting of Data", and in Chapter 11, Vol B,"Multivariate Analysis Methods".

5.7 Table Format Files

The conversion of ASCII data into table data can be done automatically (default option) for tables with `REAL*4` columns. In the case of more complex tables, a format file has to be provided to control this conversion.

Format files are `ASCII` files with an extension `.fmt`, used optionally by the commands `CREATE/TABLE, READ/TABLE` and `PRINT/TABLE` to control the input/output conversion. They may contain first a `FS` statement, they must contain then one `DEFINE/FIELD` statement for each column of the table and optional comment statements. `DEFINE/FIELD` statements follow the syntax:

```
DEFINE/FIELD pos1 pos2 type [format] label [unit]
```

where:

COMPUTE/TABLE table column = expression
COMPUTE/REGRESSION table column = name[(ind-vars)]
COMPUTE/HISTOGRAM image = table column
COMPUTE/HISTOGRAM table/TABLE = table column
CONVERT/TABLE image = table indv[,indv] depv refima method [par]
COPY/KT keyword table column row
COPY/TK table column row keyword
COPY/TI in-table out-image
COPY/IT in-image out-table
COPY/TT in-table column [out-table] column
COPY/TABLE in-table out-table
CREATE/COLUMN table column [unit] [format] [type]
CREATE/TABLE table ncol nrow filename [formatfile]
DELETE/COLUMN table column [...]
EDIT/TABLE table [ncol nrow]
IDENTIFY/CURSOR table identifier x [y] [tolerance]
IDENTIFY/GCURSOR table identifier x [y] [tolerance]
INTERPOLATE/IT out-table i,d in+image 5 [degree]
INTERPOLATE/TI out-image in-table i,d refima 5 [degree]
INTERPOLATE/TT out-table i,d in-table i,d s [degree]
JOIN/TABLE tab1 col1,col2 tab2 col1,col2 outtab tol1,tol2
LOAD/TABLE table1 column1 column2 [column3] [p1 [p1] [p3]]]
MERGE/TABLE table1 [table2 ...] out-table
NAME/COLUMN table column [column] [unit] [format]
OVERPLOT/HISTOGRAM table column [bin [min [max [LOG10]]]]
OVERPLOT/TABLE table column1 column2 [s-type]
PLOT/HISTOGRAM table column [sc-x,sc-y] [bin [min [max]]] [LOG10]
PLOT/TABLE table column1 column2 [sc-x,sc-y]
PRINT/HISTOGRAM table column [bin [min [max]]]
PRINT/TABLE table [column1 ...] [row1 [row2]] [file [format]]
PROJECT/TABLE in-table out-table column [column ...]
READ/HISTOGRAM table column [bin [min [max]]]
READ/TABLE table [column1 ...] [row1 [row2]] [format]
REBIN/IT out-table i,d[,b] in-image func parm intop
REBIN/TI out-image in-table i,d[,b] refima func parm intop
REBIN/TT out-tb i,d[,b] in-table i,d[,b] func parm intop
REGRES/LINEAR table dep-var ind-var1,ind-var2, ...
REGRES/POLYN table dep-var ind-var1[,ind-var2] degree1[,degree2]
RETRO/TAB table
SAVE/REGRESSION table name
SELECT/TABLE table logical-expression
SET/REFCOLUMN table column
SHOW/TABLE table
SORT/TABLE table column
STATISTICS/TABLE table column
WRITE/TABLE table column row value

Table 5.9: Table Commands

`pos1` — INTEGER, is the optional *starting position* of the field.

`pos2` — INTEGER, is the optional *last position* of the field.

`type` — defines the *type* of information as:

- `R` — REAL number, single precision,
- `D` — real number, DOUBLE PRECISION,
- `I` — INTEGER number,
- `C` — CHARACTER string.

`format` — defines the *format* associated with that field and used for listing out its values. Supported format are FORTRAN 77 standard format or special formats to accommodate sexagesimal values (*Sww.dd*) and time values (*Tww.dd*). These formats may be defaulted, the defaults being defined as:

- `Aw` — for CHARACTER string, where w = pos2-pos1+1
- `I11` — for INTEGER
- `E12.6` — for REAL in single precision
- `D24.17` — for REAL in double precision

`label` — defines the associated *label*, according to the rules in section 5.6.

`unit` — defines, optionally, the associated *units*.

The statement `FS` defines the list of field separators used in the ASCII data file. It is only used when pos1 and pos2 are not specified in the DEFINE/FIELD statement. This statement should be written as follows: FS = "f1f2f3". The number of field separators is not limited. If the blank is used as field separator and if the ascii data file contains character strings, the strings have to be enclosed by double quotes. Per default,
FS = "\t", i.e TABS and blanks are used as field separators.

The following format file

```
!+
!  Example format file test1.fmt
!+
DEFINE/FIELD 1 9 C :NAME "NGC"
DEFINE/FIELD 10 14 R F5.2 :RA "HOUR"
DEFINE/FIELD 16 20 R F5.2 :DEC "DEGREE"
DEFINE/FIELD 22 22 C :TYPE " "
DEFINE/FIELD 24 26 I :RV "KM.SEC-1"
END
```

corresponds to an `ASCII` file, `test1.dat` say, with the following record structure:

```
.........1.........2.........3
123456789012345678901234567890

NGC 3379 10.75 12.85 E 893
```

(The ruler, of course, is not part of the data file.)

The following format file, using FS statement,

```
!+
!  Example format file test2.fmt
!+
FS = "" DEFINE/FIELD C :NAME "NGC"
DEFINE/FIELD R F5.2 :RA "HOUR"
DEFINE/FIELD R F5.2 :DEC "DEGREE"
DEFINE/FIELD C :TYPE " "
DEFINE/FIELD I :RV "KM.SEC-1"
END
```

can be used to create a table from the ASCII file `test2.dat`

```
NGC 3379<TAB>10.75<TAB>12.85<TAB>E<TAB>893
```

5.8 Example

As an example of use of the table file system, we here describe the tutorial procedure executed via the `TUTORIAL/TABLE` command. This procedure uses a subset of the Uppsala General Catalogue. The format of the catalogue is defined in the file `ugc.fmt` as follows:

```
DEFINE/FIELD  9 20 R G11.6 :RA   "HOUR"
DEFINE/FIELD 21 32 R G11.6 :DEC  "DEGREE"
DEFINE/FIELD 33 44 R G11.6 :DB   "ARC.MIN."
DEFINE/FIELD 45 56 R G11.6 :DR   "ARC.MIN."
DEFINE/FIELD 57 68 R G11.6 :BT   "MAGNITUDE"
DEFINE/FIELD 69 80 R G11.6 :RV   "KM.SEC-1"
END
```

This file and the actual ASCII data in `ugc.dat` will be copied into your workspace. The first four lines of the data file have the following layout:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
          0.0117        15.87         6.500         6.300         12.00        1047.
          0.0233        20.47         4.000         3.800         12.70          *
          0.2933        59.03         8.000         10.00           *            *
          0.3583        16.20         5.800         5.100         14.60          *
```

(The ruler, of course, is not part of the data file.)

This tutorial shows the usage of some of the basic table file commands to analyse and display the data set.

```
CREATE/TABLE ugc 10 700 ugc     ! create the table file (UGC.tbl)
NAME/COL ugc :RA  F11.3         ! change format
NAME/COL ugc :DEC G12.6         ! change format
SHOW/TABLE   ugc                ! display structure
READ/TABLE   ugc  @1 @30        ! display a few entries
!
PLOT/TABLE  ugc :DR :DB         ! plot diameters in red and blue bands
!
REGR/LINEAR ugc :DB :DR         ! linear regression on these variables
READ/KEY    OUTPUTD             ! and display stored coefficients
!
READ/HIST   ugc :BT             ! display results on terminal
PLOT/HIST   ugc :BT             ! and plot device
!
SELECT/TAB ugc :BT.LT.13.5      ! select brightest objects
!
STAT/TAB   ugc :DR              ! do statistics on the subset,
READ/HIST  ugc :DR              ! display the result
!
PLOT/TAB   ugc :BT :RV          ! and plot the selected set
!
SELECT/TAB ugc :RV.GT.4000.0    ! select new subset with largest rad.vel.
!
PRINT/TAB  ugc                  ! print them
!
COMPUTE/TAB ugc :MBT = :BT-25.-5.*LOG10(:RV/50)   ! compute abs.magnitude
NAME/COL    ugc :MBT "ABS.B.MAG."                 ! include units
!
COMPUTE/TAB ugc :SIZE = :RV*SIN(0.000291*:DB)*20  ! diameter
NAME/COL    ugc :SIZE "KPC"                       ! include units
!
PLOT/TAB   ugc :MBT :SIZE                         ! display result
```

Chapter 6

Graphic and Image Display

6.1 Graphic Facilities

This section describes the facilities of the graphics package in MIDAS. The package makes use of the Astronet Graphic Library (AGL) , which has been accepted as the standard for maintenance and development of the MIDAS graphics software. An overview of the graphic commands currently available in MIDAS is presented.

6.1.1 Introduction

In order to provide a device-independent graphics package, the Italian Astronet Graphic Library has been adopted as the standard low level library for the plot package in MIDAS. One of the main reasons for using the AGL package rather than one of the more evolved and sophisticated packages (*e.g.* GKS) was the fact that the AGL package is simple; meanwhile AGL is still capable of doing the things one needs for reasonably advanced graphics, in particular with respect to interactive facilities. The AGL graphics library is fully integrated within the MIDAS directory structure and is generated like every other MIDAS subroutine library during installation. For an extensive description of the package we refer to the AGL User's and Installation Guides for Version 3.

The graphic facilities available in MIDAS can be divided into three main categories of commands, based on their functionality:

- general plot commands that deal with the setup of graphic packages: the assignment of the graphics device, routing a plot file to a graphic device, and general (over)plot commands for text, line, symbols, etc.;
- main plot commands which do the actual plotting and overplotting of data (*i.e.* images, tables, descriptors or keywords);
- cursor commands which use the graphics cursor.

Below, subsection 6.1.2 first describes how graphic display units (*e.g.* terminals, workstations) can be activated inside the MIDAS environment. Thereafter, in the subsections 6.1.3 to 6.1.5 the plot commands available in MIDAS will be discussed. In subsection 6.1.6 information can be found about how plot files can be manipulated. In section 6.1.8 a number of examples are presented to illustrate the available commands and their syntax. Finally, in subsection 6.1.9 an overview is given of all the available graphic commands in MIDAS.

6.1.2 Graphic Devices

The characteristics of graphic devices differ from device to device. Therefore, in order to obtain a useful plot, the characteristics of the graphics device in use have to be known in advance. MIDAS can run on various combinations of alpha–numerical terminals, image display systems and graphic devices. How to specify the graphic output device for these possibilities is described below.

If you are using the standard MIDAS configuration (alpha–numerical and graphic terminal together with an image display), the MIDAS start–up procedure MIDAS takes care of the proper assignment. If you want to obtain graphic output on a workstation running under X-Windows you have to issue the command `CREATE/GRAPHIC`. This command creates a window on the workstation where subsequent plot and overplot commands will write. Up to 4 graphic windows can be created this way. Removal of a window can be done with `DELETE/GRAPHIC`.

To get plot output on a graphics device (the graphic terminal included), a proper assignment for that device has to be done in advance. Obviously, the assignment depends on the type of device in use and hence may differ from system to system. If your institute mainly uses graphic terminals of brand "abc", life would be much simpler if this device were the default one and therefore the assignment to be included in the MIDAS startup procedure. You can ask your local MIDAS support to do so. The assignment to be made is:

- for VAX/VMS systems: `ASSIGN` *AGL_type* `AGL3DEV`;
- for UNIX systems (C-Shell): `setenv AGL3DEV` *AGL_type*;
- for UNIX systems (Bourne-Shell): `AGL3DEV=`*AGL_type*.

Changing the assignment from the default device to another one can always been done in the `login.prg` file (see Chapter 3). This of course would be useful if a particular device is not assigned by the MIDAS startup procedure but is used regularly. Finally, if you run MIDAS from a stand-alone (non-graphic) terminal, an assignment to the NULL device or to the postscript device can be made. In the first case only a graphic meta file will be created; in the later case a postscript file. In case of problems consult your local MIDAS support or your system manager.

Table 6.1 contains the most commonly used graphics devices (the *AGL_type*'s) currently supported by MIDAS.

Device Identification	AGL_type
DEC VT125 terminal	vt125
DEC VT240 terminal	vt125
DEC VT125 terminal with Retrogr.	tkg.vt640
DEC VT100 terminal with Retrogr.	tkg.vt640
CIT101 with CIG 201 card	tkg.cit101
GraphOn GO–250	tkg.vt640
HDS 2200	tkg.hds22
HPGL plotters	hpgl
LN03 plus laser printer	tkg.ln03
Tektronix 4010	tkg.t4010
Tektronix 4014	tkg.t4014
Tektronix 4100 series	tkg.t4100
QMS laser printer	tkg.qms
X-Windows	idi
Apple Laser Writer	pscript
Postscript printers	pscript
Postscript file	pscript
Versatec V-80	raster
Raster devices	raster
Null device	null

Table 6.1: Supported Devices

6.1.3 General Commands

The general commands in MIDAS mostly concern setting the plot characteristic, displaying the setup, assigning the graphic output device, and sending an existing plot to a device. These commands are:

`CREATE/GRAPHIC`	reate a graphic window on the workstation
`DELETE/GRAPHIC`	delete a graphic window from the workstation
`CLEAR/GRAPHIC`	clear the graphic screen or window
`SET/GRAPHIC`	set the graphic characteristics
`SHOW/GRAPHIC`	show the graphic characteristics
`ASSIGN/GRAPHIC`	assign the graphic device
`COPY/GRAPHIC`	route the plot file to a graphic device

The first two commands in this table are meant for users with workstations running the X-Window display software. With these commands one can create and delete graphics window(s). The create command allows control over the size of the graphics window as well as the position where it is to be put. `CLEAR/GRAPHIC` erases the graphics window or graphics terminal screen.

`SET/GRAPHIC` and `SHOW/GRAPHIC`

`SET/GRAPHIC` plays an important role in the plot package. This command gives the user control over the plot size, line types (`LTYPE`), line thickness (`LWIDTH`), symbol type (`STYPE`) and size (`SSIZE`), etc. In Table 6.2 the various options that can be set with the `SET/GRAPH` command are listed, together with the default settings. Below we briefly discuss some of the options. More extended documentation can be found in the help file of the `SET/GRAPHIC` command.

By default, all data points in frames, keywords and descriptors are connected with a line; data points in tables are plotted individually. This setting can be changed with the `LTYPE` and `STYPE` options in `SET/GRAPH`. Normally, when plotting data points in a frame, descriptor, or keyword, the plot package first looks for the line type. If the line type is set to 0 (`LTYPE=0`) it looks for the symbol (`STYPE`). If both `LTYPE` and `STYPE` are found to be 0 a fatal error occurs.

In case of table plotting the package first looks for the symbol type. When `STYPE=0` a line will be drawn corresponding to `LTYPE`. An error occurs if both `LTYPE` and `STYPE` are 0. For histogram plotting or when the bin mode @BINis on (`BIN=ON`), the package needs a line type greater than zero; an error occurs when `LTYPE=0`. Table data can not be plotted with `BIN=ON`.

By default, before a `PLOT` command is executed the graphics window is erased. To switch off the erase, you can use the option `CLEARGRA=OFF`. By doing so, subsequently issued plot commands will run in overplot mode: the screen content is kept. Hence, by issuing a number of `PLOT` commands you can easily produce several plots on one screen (page). To help you more in designing the layout of your plot scales (and sizes) as well as the position on the screen (paper) can be pre-defined by `XSCALE` and `YSCALE`, and `XOFFSET` and `YOFFSET`.

The default font used by MIDAS is a simple one but is plotted fast. More fonts are available to enable you to obtain publication quality graphics output. With the command `SET/GRAPHIC FONT=n`, where n is larger than 0, you can use a different (nicer) font than the default one. Currently, the following font sets are available:

- `0` Default built-in font
- `1` High quality roman font
- `2` Greek font
- `3` Script font
- `4` Old English font with astronomical symbols
- `5` Tiny roman font, simpler than 1

To display these fonts and the associated character and symbol sets you can run the tutorial TUTORIAL/GRAPH GENERAL.

If a colour postscript printer is at your disposal you may use the colour setting options COLOUR=n and BCOLOUR=n.

In the SET/GRAPH command defaults for single parameters can be (re)set by: SET/GRAPH par_name=value. The reinitialization of all plot parameters can be done by: SET/GRAPHIC or SET/GRAPHIC DEF.

ASSIGN/GRAPHIC and COPY/GRAPHIC

Table 6.1 contains the graphic output devices on which MIDAS plots can be produced. The user can specify one of these output devices in advance by the ASSIGN/GRAPHIC command. In addition to the hardcopy devices available, the command can also be used to reassign the graphics window. For example, the user can indicate that the plot has to be produced on a second graphics window, or on a display window of his/her workstation. After the plot command has finished and a plot file is produced, this plot file can be sent to a device by the COPY/GRAP command. This command accepts the same graphics devices as the ASSIGN/GRAPH command. For workstations this offers the possibility to copy a graph from one window to another. In section 6.1.6 more information can be found about how MIDAS takes care of your plot files.

Example:

```
assign/gra laser nospool
plot/tab example ?  #1 -50,-70,10,90
overplot/tab example ?  #2 -50,-70,70,90
copy/graphic laser
copy/graphic g,0
assign/gra g,0
```

In this example we first assign the graphics output to become the output device. However, the plot file is kept on disk and not spooled to the printer. After the plot is finished, it is sent to the printer. Hereafter, we also send a copy to the graphics window (provided one exists). Finally, we assign the graphics window as the output device.

6.1.4 Plot Commands

As described in Chapter 3, the MIDAS data structures include frames, masks, tables, catalogues, descriptors, and keywords. With the exception of the masks and catalogues, the plot package is able to plot the data stored in these structures. Data can be plotted with PLOT/ as well as with OVERPLOT/ commands. In the first case, MIDAS will start a complete new plot (*e.g.* a graphic terminal screen will be erased and old plotfiles will be deleted); in the latter one MIDAS will extend the existing plot information without erasing the results of previous plot commands.

In addition of the plot commands that enable the user to plot data command are available to produce coordinate box(es) within subsequently data can be overplotted, and com-

Option	Value and meaning and defaults
DEFAULT	no value; sets plot package in default mode
XAXIS=	AUTO or xstart,xend,xbig_tick,xsmall_tick in world coordinates;
	when xsmall_tick $<$ 0 a logarithmic axis is plotted; the default is AUTO
YAXIS=	AUTO or ystart,yend,ybig_tick,ysmall_tick in world coordinates;
	when ysmall_tick $<$ 0 a logarithmic axis is plotted; default is AUTO
FRAME=	RECT or SQUA; default is RECT
XSCALE=	AUTO, scale in world units/per mm or size of plot
YSCALE=	AUTO, scale in world units/per mm or size of plot
XOFFSET=	NONE or the offset of the left y axis to left device boundary
YOFFSET=	NONE or the offset of the lower x axis to lower device boundary
XFORMAT=	NONE, AUTO or format description (see below)
YFORMAT=	NONE, AUTO or format description (see below)
PMODE=	0 (plot without frame and legend), or
	1 (plot with frame and some information), or
	2 (plot with frame and full legenda) which is default
FONT=	font to be used to write text; default is 1; (see below)
LTYPE=	1 (solid line) to 6 (long dash - short dash); (see below)
	default is 1; 0 corresponds with no line at all
LWIDTH=	set line width; 0 or 1 for single width; 2, 3 and 4 for increasing thickness
STYPE=	1 (dot) to 21 (left arrow); default 4 (cross); 0 corresponds with no symbol at all
SSIZE=	value; set the scaling factor of symbols; default is 1
TSIZE=	value; set the scaling factor for text strings; default is 1
TWIDTH=	value; set the line width for text strings; default is 1
BINMODE=	OFF or ON; default is OFF
COLOUR=	number ranging from 0 to 7; the default is black (1)
	The setting has only effect on graphic display devices supporting colour
BCOLOUR=	number ranging from 0 to 7; set the background colour; the default is black (1)
	The setting has only effect on graphic display devices upporting colour
CLEARGRA=	ON or OFF. OFF will not clear graphic screen for a PLOT command;
	default ON

Table 6.2: SET/GRAPHIC Options

mands to add text, symbols, and (grid) lines. An overview of the commands is given below.

PLOT/AXES	plot a box with tick marks, tick labels and axes labels
OVERPLOT/AXES	overplot a box with tick marks, tick labels and axes labels
PLOT/CONTOUR	plot contours of a two-dimensional image
OVERPLOT/CONTOUR	overplot contours of a two-dimensional image
PLOT/COLUMN	plot a column of an image
OVERPLOT/COLUMN	overplot a column of an image
PLOT/DESCRIPTOR	plot an entry in a descriptor
OVERPLOT/DESCRIPTOR	overplot an entry in a descriptor
PLOT/GRAY	plot gray scale map of a two-dimensional image
OVERPLOT/GRAY	overplot gray scale map of a two-dimensional image
PLOT/HISTOGRAM	plot a histogram of a table column or image
OVERPLOT/HISTOGRAM	overplot a histogram of a table column or image
PLOT/KEYWORD	plot the contents of a keyword
OVERPLOT/KEYWORD	overplot the contents of a keyword
PLOT/PERSPECTIVE	perspective plotting (3-dim.) of an image
PLOT/ROW	plot a row (line) of an image
OVERPLOT/ROW	overplot a row (line) of an image
PLOT/TABLE	plot table data
OVERPLOT/TABLE	overplot table data
OVERPLOT/ERROR	overplot table column containing errors
PLOT/VECTOR	plot vector map from two 2-dim. images with smoothing option
OVERPLOT/VECTOR	overplot vector map from two 2-dim. images with smoothing option
LABEL/GRAPHIC	plot text in an existing plot
OVERPLOT/LINE	overplot a line in an existing plot
OVERPLOT/SYMBOL	overplot a symbol in an existing plot
OVERPLOT/GRID	overplot a grid, by connecting tickmarks

Plotting and overplotting axes

The command `PLOT/AXES` offers the users the possibility to draw a frame with certain ranges in x and y. The command is very flexible, since it accepts both the ranges in x and y and the scaling factors as input parameters. Also, the user has the freedom to select the location where the frame is to be drawn. The actual data points can be plotted with subsequent overplot commands (see example below). More coordinate boxes can be plotted using the command `OVERPLOT/AXES` with the same parameter list as in the `PLOT/AXES` command. In the example below a series of plots is produced with plot and

overplot commands. First, we start with a PLOT/AXES and an OVERPLOT/TABLE command, and then continue with three OVERPLOT/AXES and OVERPLOT/TABLE commands. The result is four graphs in the graphics window.
Example:

```
assign/graph g,0                              ! assign graphic window 0
plot/axes 0,10 -1,1 -50,-70,10,80             ! plot the first axes
overplot/tab example ?  #1                    ! plot first coord. box
overplot/axes 0,10 -1,1 -50,-70,70,90         ! overplot second box
overplot/tab example ?  #2
overplot/axes 0,10 -1,1 -50,-70,130,90        ! overplot third box
overplot/tab example ?  #3
overplot/axes 0,10 -1,1 -50,-70,190,90        ! overplot fourth box
overplot/tab example ?  #4
```

Alternatively, the system also offers a more simplier way of doing the same thing: instead of the PLOT/AXES and OVERPLOT/AXES commands we switch off the clearing of the graphics window first and continue with simple PLOT/TABLE commands.
Example:

```
clear/graph
set/graph clear=off                           ! erase switched off
plot/tab example ?  #1 -50,-70,10,10          ! plot the fifth box
plot/tab example ?  #2 -50,-70,70,10
plot/tab example ?  #3 -50,-70,130,10
plot/tab example ?  #4 -50,-70,190,10         ! plot the last box
```

For both the plot and overplot commands one can use the command SET/GRAPH XFORMAT=none YFORMAT=none to switch off the tick mark labels along the axes. Of interest, especially for overlays, is another syntax of the PLOT/AXES and OVERPLOT/AXES command: it offers the possibility of drawing axes around (part of) an image displayed in the display window.

PLOT and OVERPLOT commands for plotting data

In general all commands for plotting data, have a well defined syntax:

```
PLOT/QUALIFIER P1 P2 P3 P4 P5 P6 P7 P8,
```

where:
P1 = table, image, descriptor or keyword name
P2 = columns, area, or indices of P1
P3 = scales in world coordinates/mm or size of the plot; only for PLOT commands
The meanings of the remaining parameters on the command lines vary from command to

command; in most cases they are used for options. Obviously, in case of overplotting , the parameter for the scales is absent.

Plotting text, symbols, and lines

For overplotting of text one can use the command `LABEL/GRAPHIC` . The text will be plotted in the font style set with the `FONT` keyword in the `SET/GRAPHIC` command. *E.g.* with `LABEL/GRAPHIC` and running in `FONT=1` text will be generated in the roman font type. `LABEL/GRAPHIC` make use of the built-in features of AGL. These features allow to change font, draw subscripts and superscripts, scale the text size, or draw various symbols, all within the text string. All these possibility become available by including metacharacters in the text string. Currently, AGL knows the metacharacter set listed in Table 6.3.

Metacharacter	**Meaning**
\{	begin grouping
\}	end grouping
\^	move the following part of the string up by half a character
\!u	move the following part of the string up by half a character
_	move the following part of the string down by half a character
\!d	move the following part of the string down by half a character
\<	backspace by a single character
\+	increase character size by 20%
\-	decrease character size by 20%
\!	force interpretation of the following part as a metasequence (this is needed to allow metasequences starting with 'n' not to be interpreted as newlines)
\0	select font 0 (Default font)
\1	select font 1 (Quality roman font)
\2	select font 2 (Greek font)
\3	select font 3 (Script font)
\4	select font 4 (Old English)
\5	select font 5 (Tiny roman font)
\[	increase line width (bolding; optional)
\]	decrease line width (bolding; optional)
\#<n>	draw marker number $< n >$ into the line
\n	perform a 'newline'
~~	draw a single '~' character (must be following by a blank)
\~	draw a single '\~' character (must be following by a blank)
~\	draw a single '\' character (must be following by a blank)
\\	draw a single '\' character (must be following by a blank)

Table 6.3: Meta Character in AGL and MIDAS

The character '~' can also be used instead of '\' as metacharacter flag. The '~' is more suited to C programs where '\' has a special meaning. All selections made by metacharacters are valid from the point in the string where they are defined up either to the end of current group (the part of the string enclosed in \{...\}) or to the end of the string. If the metacharacter sequence is more than one character long (escape not included, of course), it must be followed by a blank space.

Example:

```
LABEL/GRAP "e\{\!u(x\{\!u2\}+y\{\!u2\})\}= -
            (\alpha +\beta ) sin\{\!u2\}\theta "
```

This command will produce the label $e^{(x^2+y^2)} = (\alpha+\beta)\sin^2\theta$ at a position which the user should give via cursor input.

MIDAS/AGL also interprets a set of 'TEX-like' keywords as listed in table 6.4. Due to the fact that most of them represent special characters and symbols to be printed, only the names are listed; the symbols can only be seen by `LABEL/GRAPHIC` or the command `TUTORIAL/GRAPH`.

Besides overplotting of text strings, the user can also overplot lines (up to six different line types, depending on the device), and symbols (more than twenty). Depending on the device, up to four different line widths can be used. The selection of line properties and of symbol type can be done with `SET/GRAPH`, or, at least for line and symbol type, on the command line.

The command `OVERPLOT/LINE` offers the possibility to overplot a line. Similarly, `OVERPLOT/SYMBOL` overplots symbols. Both commands can be used interactively using the graphics cursor, or by giving the coordinates on the command line. Grid lines, connecting the major and/or minor tick marks can be drawn with the command `OVERPLOT/GRID`.

6.1.5 Graphic Cursor Commands

In some of the analysis programs in MIDAS the graphic cursor is a powerful tool. For example, using the cursor one can retrieve wavelengths and line intensities in a plotted spectrum, integration of emission or absorption lines over a wavelength range selected by the cursor, compute the line width and center, etc. With the general `GET/GCURSOR` command the user can retrieve information from plotted data and store this in a table. Listed below are some of the core and application commands which use cursor interaction. Many additional graphics commands, including those that use cursor interaction, are available in the various contexts, *e.g.* `SPEC` and `ECHELLE`.

`GET/GCURSOR`	read coordinates from graphics screen
`CENTER/GAUSS`	computes center of a 1-dim. or 2-dim. feature
`MODIFY/GCURSOR`	change data line of an image interactively
`INTEGRATE/APERTURE`	compute flux inside an aperture
`INTEGRATE/LINE`	integrate row of a frame using the cursor
`INTEGRATE/STAR`	compute flux, radius and background of stars

\AA	\Alpha	\Aquarius	\Aries
\Beta	\Cancer	\Capricorn	\Chi
\Delta	\Earth	\Epsilon	\Eta
\Gamma	\Gemini	\Iota	\Jupiter
\Kappa	\Lambda	\Leo	\Libra
\Mars	\Mercury	\Moon	\Mu
\Neptune	\Nu	\Omega	\Omicron
\PI	\Phi	\Pisces	\Pluto
\Psi	\Rho	\Sagittarius	\Saturn
\Scorpio	\Sigma	\Sqrt	\Tau
\Taurus	\Theta	\Upsilon	\Uranus
\Venus	\Virgo	\Xi	\Zeta
\aleph	\alpha	\asteroid	\beta
\bigcirc	\black	\blue	\cent
\chi	\circ	\cyan	\clover
\clubsuit	\comet	\dag	\ddag
\default	\delta	\diamond	\div
\downarro	\epsilon	\equinox	\equiv
\eta	\firtree	\gamma	\ge
\greek	\green	\hbar	\heart
\infty	\int	\iota	\italic
\kappa	\lambda	\larrow	\le
\magenta	\mp	\mu	\!nabla
\!ne	\!nu	\odot	\oint
\old	\omega	\omicron	\oplus
\otimes	\palmtree	\paragraph	\parallel
\partial	\perp	\phi	\pi
\pm	\propto	\psi	\red
\rho	\rightarrow	\roman	\script
\shield	\sigma	\snow	\spade
\sqrt	\sum	\tau	\theta
\times	\tiny	\uparrow	\upsilon
\varepsilon	\varphi	\vartheta	\white
\xi	\yellow	\zeta	

Table 6.4: TEX-like Characters for text strings in MIDAS Graphics

6.1.6 Handling of Plotfiles

Plotting in MIDAS will create a MIDAS plotfile which contains all essential plot information. By default this plotfile (metafile) is always created by the execution of the main plot commands. The plotfile will carry the name of the data structure that has been plotted: the name of a frame, table, descriptor or keyword. The plotfile has the extension ".plt". MIDAS keeps track of what the user has plotted. The `SHOW/GRAPH` command shows the user which is the last created plotfile. Subsequent overplot commands will append to this plotfile. Names of plotfiles are a composition of the device names that is currently assigned and the MIDAS session number. As an example, take the situation were the user runs MIDAS under the unit number 04 and the graphics window 1 (graph_wnd1) is assigned. In this case all plots created will be stored in the file `graph_wnd104.plt`. If the device "lp" was assigned, plot files will go into the file `lp04.plt`. Names of plot files are unique and do not have version numbers. Hence, MIDAS will delete an old plotfile if a new one with the same name is created. With the command `SHOW/GRAPH` you can display the name of the plot file.

There are several ways to obtain a hardcopy of a plot. Below you will find a few examples.

1. The user works with a graphic terminal or a workstation and has made this plot on the graphic terminal first. He/she can now send the plot to a hardcopy device using the `COPY/GRA` command.
   Example:
   ```
   MIDAS 001> PLOT/TABLE mytable :velocity :distance
   MIDAS 002> COPY/GRAPH LASER
   ```

2. The user does not have a graphics terminal (or does not want to use it), and wants to dump his plot directly onto a hardcopy device. In this case, the hardcopy device has to be assigned first as the output device by the `ASSIGN/GRAPH` command. Now all the plot(s) (including the overplot !!!) will be sent directly to the hardcopy device.
   Example:
   ```
   MIDAS 003> ASSIGN/GRAPH LASER        ! assign LASER as output device
   MIDAS 004> PLOT/TABLE mytable :velocity :distance   ! make plot
   ```

3. In a MIDAS plot command sequence (with many *e.g.* `OVERPLOT` and `LABEL` commands) intermediate output is not always wanted, in some cases even undesirable. In order to switch off the direct routing of plots to a device users can specify the extra switch `NOSPOOL` in the `ASSIGN/GRAPH` command. Using this switch the plotfile(s) will be stored on disk first. Once the user has finished his sequence of plot commands, he/she can create the complete plot on the hardcopy device using the command `COPY/GRAPH`. Intermediate results can be obtained using the same command.
   Example:
   ```
   MIDAS 005> ASSIGN/GRAP LASER NOSPOOL          ! plot file, don't send
   MIDAS 006> PLOT/ROW frame [@100,@150:@150,@250] 20.0,20.0
   ```

```
MIDAS 007> OVERPLOT/TABLE table #1                  ! overplot
MIDAS 008> LABEL/GRAPHIC "THIS IS AN EXAMPLE" 90 4 400,300
MIDAS 009> COPY/GRAPH LASER                         ! send the plot file
```

4. The user wants to send a previously created MIDAS plotfile (*e.g.* "midas.plt", and different from the last created plotfile) to a device.
   Example:

```
MIDAS 010> COPY/GRAPH LASER frame.plt     ! send plotfile to LASER
```

As can be seen in section 6.1.9, in most cases the user can produce a plot with certain scales of the x– and y–axis. In the current version routing the plot file (with `COPY/GRAPH`) to a device different from the one pre-specified (with the `ASSIGN/GRAP` command) may lead to incorrect scales. In case the pre-specified device is the same as the device to which the plot is sent the scales will be correct.
Example:

```
MIDAS 005> ASSIGN/GRAPH VERSA NOSPOOL
MIDAS 006> PLOT/ROW image [@100,@150:@150,@250] 20.0,20.0
MIDAS 007> OVERPLOT/TABLE table #1
MIDAS 008> LABEL/GRAPHIC "THIS IS A EXAMPLE" 90 4 400,300
MIDAS 009> COPY/GRAPH LASER            ! plot will have incorrect scales
MIDAS 010> COPY/GRAP VERSA                          ! with correct scales
```

6.1.7 Encapsulated PostScript Files

For any PostScript hardcopy printer that has been assigned by `ASSIGN/GRAPH`, MIDAS produces a so-called encapsulated PostScript file. Using a public domain macro package (*e.g.* Psfig/TEX) this PostScript plot file can, with a minimum of effort, be included in TEX or LATEX documents . To do so, in the TEX or LATEX document one should refer to the (possibly renamed) MIDAS PostScript file (normally `pscrplot.ps`). Below, follows a simple LATEXexample that shows how it works.
Example:

```
MIDAS 005> ASSIGN/GRAPH postscript
MIDAS 006> PLOT/ROW image [@100,@150:@150,@250] 20.0,20.0
MIDAS 007> OVERPLOT/TABLE table #1
```

In your directory we now have a PostScript file postscript.ps, containing the complete plot information written by the commands 006 and 007. Now, you can include this MIDAS PostScript file in our LATEX document, in this case using psfig, developed by Trevor Darrell (trevor@media.mit.edu). Here is how the LATEX text file with the included MIDAS plot then looks like.
Example:

```
\documentstyle[11pt,psfig]{article}
 \begin{document}
  \section*{Abstract}
   We show a simple example of how one can include a PostScript figure,
   generated by MIDAS, into a existing \LaTeX document.
   \nopagebreak

   \begin{figure}[h]
    \centering{
     \hspace*{-1.cm}
     \vbox{\psfig{figure=postscript.ps,width=10cm,height=5cm}}\par
    }
   \end{figure}
 \end{document}
```

6.1.8 Examples

To show the usage and the possibilities of the plot package, a number of examples are built are available below. These examples are available via the `TUTORIAL/PLOT P1` command, where `P1` can be:

- `GENERAL` to show an example for the graphics utilities, like drawing lines, symbols, text, changing fonts, etc ...
- `AXES` to show an example of plotting several axes in one graphics window and using different axes;
- `TABLE` to show an example of table plotting using different symbols and axes;
- `1DIM` to show an example of (spectral) line plotting;
- `2DIM` to show two–dimensional gray scale and contour plotting.

6.1.9 Command Summary

Table 6.5 on the next page shows an overview of the graphic commands in MIDAS, listed in alphabetical order.

ASSIGN/GRAPH [device_name] [spool_option]
CENTER/GAUSS in_specs out_specs [out_opt]
CLEAR/GRAPHIC
COPY/GRAPHIC [device_name] [plot_file]
CREATE/GRAPHIC [graph_id] [graph_spec]
DELETE/GRAPHIC [graph_id]
GET/GCURSOR [out_specs] [app_flag] [max]
INTEGRATE/LINE frame [y_coord] [x_start,x_end] [no_curs,degree] [batch_specs]
LABEL/GRAPHIC label [x_pos,y_pos[,mm]] [angle] [size] [centering]
MODIFY/GCURSOR frame [y_coord] [x_start,x_end] [no_curs,degree]
OVERPL/AXES [x_axis_spec] [y_axis_spec] [x_scale,y_scale] [x_label] [y_label] [x_off,y_off]
OVERPLOT/AXES [coord_str]
OVERPLOT/COLUMN frame [x_coord] [x_sta,x_end] [offset] [l_type]
OVERPLOT/CONTOUR frame coord_str [contours] [sm_par]
OVERPLOT/DESCRIPTOR frame [descriptor] [start,end] [offset]
OVERPLOT/ERROR table column1 [column2] column_err [orient]
OVERPLOT/GRAY frame [coord_str] [gray_lev] [sm_par] [gray_ness] [options]
OVERPLOT/GRID [tick_opt]
OVERPLOT/HISTOGRAM table column [bin [min [max]]] [offset] [log_flag]
OVERPLOT/HISTOGRAM frame [offset] [log_flag]
OVERPLOT/KEY [key_word] [start,end] [offset]
OVERPLOT/LINE [line_type] [x_sta,y_sta [x_end,y_end]]
OVERPLOT/ROW frame [y_coord] [x_sta,x_end] [offset] [l_type]
OVERPLOT/SYMBOL [x_coord,y_coord] [s_type] [s_size]
OVERPLOT/TABLE table [column1] [column2] [s_type]
OVERPLOT/VECTOR frame_a frame_b [coord_str] [sc_x,sc_y] [scale_r] [pos_range] [sm_par] [head]
PLOT/AXES [x_axis_spec] [y_axis_spec] [sc_x,sc_y] [x_label] [y_label] [x_off,y_off]
PLOT/AXES [coord_str]
PLOT/COLUMN frame [x_coord] [x_sta,x_end] [sc_x,sc_y]
PLOT/CONTOUR frame [coord_str] [x_scale,y_scale] [contours] [sm_par]
PLOT/DESCRIPTOR frame [descriptor] [start,end] [x_scale,y_scale]
PLOT/GRAY frame [coord_str] [x_scle,y_scale] [gray_lev] [sm_par] [gray_ness] [options]
PLOT/HISTOGRAM frame [x_scale,y_scale] [log_flag]
PLOT/HISTOGRAM table column [x_scale,y_scale] [bin [min [max]]] [log_flag]
PLOT/VECTOR frame_a frame_b [coord_str] [sc_x,sc_y] [scale_r] [pos_range] [sm_par] [head]
PLOT/KEY [keyword] [start,end] [x_scale,y_scale]
PLOT/PERSPECTIVE frame [coord_str] [azi_angle,alt_angle] [sm_par] [xy_flag]
PLOT/ROW frame [y_coord] [x_sta,x_end] [sc_x,sc_y]
PLOT/TABLE table [column1] [column2] [x_scale,y_scale]
SET/GRAPHIC option1[=value1] [option2[=value2] ...]
SHOW/GRAPHIC

Table 6.5: Graphic Commands

6.2 Image Displays

This section describes the setup of the image displays used by MIDAS and the functionality provided by MIDAS to interact with these displays. For a description of the conceptual model for an image display device see the definition document for the IDI–routines.
Via the IDI–interfaces MIDAS supports peripheral display devices like e.g. the venerable DeAnza or Ramtek devices and XWindow displays. Since the peripheral display devices are more or less extinct by now, we describe here only the way how MIDAS uses an XWindow display.

6.2.1 XWindow display

With the term `XWindow` display we refer to a bitmapped screen supporting the X–11 Window environment. These displays have less functionality provided in hardware than the "classical" peripheral image displays. On the other hand they offer much more flexibility via software. For example, display screens of different sizes may be created and different number of image channels may be connected to any one display.
MIDAS starts up an independent server process, the *IDI display server*, which owns all X11 related data structures. MIDAS applications do not interact directly with the windows but send messages to the server which then performs the actual task. Like this we can keep the windows alive while the different applications are executed and terminated, one by one.
Also, keep in mind that all interaction with the display will only work while the input focus is in the display window (either enforced by clicking the mouse in that window or just moving the cursor into it – that depends on how your window manager is set up).

Depending upon the graphics hardware on your system you can work in PseudoColor mode (8 bits per pixel) or RGB mode (24 bits per pixel). In both modes the values of image pixels are scaled to an integer number in the range [0,255]. That number then serves as an index into a Color Lookup Table (LUT) which determines the actual color of that pixel. In PseudoColor mode that index is the same for the red, green and blue color, whereas in RGB mode we use three image pixels per screen pixel and they serve as independent indices for the three colors. These three image pixels maybe in different planes of the same 3-dim image or in different 2-dim images. Accordingly, we use three image memories (channel 0, 1 and 2) for the red, green and blue component of an image.
You set MIDAS to PseudoColor or RGB mode via the command `INITIALIZE/DISPLAY` (by default PseudoColor mode is set up).
Assuming you have a 3-dim image *realima.bdf* with the red part in the 1st plane, green part in the 2nd and blue part in the 3rd plane, you can either load each plane individually into the corresponding image channel:

```
INITIALIZE/DISPLAY p5=RGB
LOAD/IMAGE realima,1 0
LOAD/IMAGE realima,2 1
LOAD/IMAGE realima,3 2                !note that channels begin with 0
```

or do it in one go:

```
INITIALIZE/DISPLAY p5=RGB
LOAD/IMAGE realima,1,3 0,...
```

The red plane is stored in channel 0, the green plane in channel 1 and the blue plane in channel 2. All three planes are combined for the display of the image.

With the command RESET/DISPLAY we delete all existing display (and graphics) windows and get back into initial mode, which is PseudoColor.
You can only work within one Color mode at the same time. However, if the hardware supports it, you can run two parallel Midas sessions with one using PseudoColor mode and the other one RGB mode.
Image displays are created on the screen via the CREATE/DISPLAY command. An "image display" is then represented by a window on the bitmapped screen, also called "display window" in the following. It may contain one or several image channels (max. 12 currently) in PseudoColor, whereas you always have 3 channels in RGB mode. An overlay channel for graphics and text and an alpha memory are associated with an image display.
The image channels may have the same size as the display window or could be larger. These channels are not realised in hardware (*e.g.* video memory) like for a peripheral image display, but exist as data structures in main memory.
Images may be displayed (*loaded* in MIDAS terminology) in image channels and it is also possible to draw graphs and write text into these channels. For the (single) overlay channel only drawing and writing of text is supported which will then be shown on top of the currently active image channel.
The alpha memory is used to display information about the displayed image.

Initially each display is provided with a grayscale LUT.
You may create several image displays at the same time on your bitmapped screen even though only one display can be the current active display window at any time. With the command ASSIGN/DISPLAY you switch from one display to the next.
Each image channel of a display window has independent scroll and zoom (emulated in software). You can also connect a *zoom window* to a display window. This window can then be used by commands such as GET/CURSOR and VIEW/IMAGE to provide a zoomed view of the subframe pointed at with the mouse.
The display windows can be moved and resized. After resizing you have to reload the previously loaded (displayed) image again.
Up to 9 different display windows of differing size can be created in a MIDAS session at a given moment. By running several MIDAS sessions in parallel you can create an unlimited amount of display windows which share the same LUT or not, depending upon the capabilities of your X-display system.
You can also create display windows on other X-displays within a MIDAS session. A special kind of display windows, so-called *shadow* displays can be connected to any display window created before. Then, all activity on that display window is replayed on the shadowing window (which could be on an X-display in another room). The most important

commands associated with display windows are:

`INITIALIZE/DISPLAY` - *setup Color mode, size of LUTs, etc.*
`CREATE/DISPLAY` - *create an image display with image channels(s)*
`ASSIGN/DISPLAY` - *switch between different display windows*
`DELETE/DISPLAY` - *delete a display window*
`SHOW/DISPLAY` - *show status of active display*
`MODIFY/DISPLAY` - *change display window from full view to icon status*
`RESET/DISPLAY` - *delete all windows and get back into inital state*

Using Image Channels

Images can be loaded into any of the different image channels (memories) of a display window, except the overlay channel. For data cubes the different planes can be loaded into different channels and then combined in their display (RGB mode) or viewed individually by blinking between the different channels.
The image channels are referenced by numbers 0, 1, ... A typical command sequence would be:

```
LOAD/IMAGE galaxy 0        ! load an image in channel 0
LOAD/LUT heat              ! load a colour look–up table
DISPLAY/LUT                ! display a LUT bar at the bottom of the window
```

The most important commands associated with the image channels are:

`LOAD/IMAGE` - *load an image into image channel*
`GET/IMAGE` - *read an image from the image channel*
`GET/CURSOR P5=w` - *read pixel values of displayed image using a zoom window*
`VIEW/IMAGE` - *explore an image already loaded*
`DISPLAY/CHANNEL` - *display image in selected channel*
`CLEAR/CHANNEL` - *erase data in an image channel*
`SCROLL/CHANNEL` - *scroll image loaded in channel*
`SHOW/CHANNEL` - *show status of image channel*
`BLINK/CHANNEL` - *blink between different channels*

Using the Cursors

Each display window has two independent cursors available. The first cursor (cursor 0) is moved via the mouse, the second cursor (cursor 1) is moved via the *arrow* keys on the keyboard.

Using both cursors a region of interest (`ROI`) is supported. The `ROI` can have rectangular or circular shape, its center is moved via the mouse and its size is adjusted via the *arrow* keys. The resizing of the `ROI` may be done in small or large increments. This is controlled via the number keys 0, 1, 2, ..., 9 on the keyboard. Pressing the 0–key corresponds to an increment of a single screen pixel, whereas 1, 2, ..., 9 lead to larger increments.

The cursor position can be read by pressing the `ENTER` button which is the leftmost button on the mouse. To exit from a command which uses interactive cursor input, press the `EXIT` button which is the rightmost button on the mouse. The middle button is currently not used in MIDAS (it behaves like the `EXIT` button), but may be employed in the future. The `RETURN` key on the keyboard serves as an `EXECUTE` button. The `EXECUTE` button works usually like the `ENTER` button, only in some very special cases its functionality is different from the `ENTER` button. If so, it is explained in the relevant help info, an example are the `MAGNITUDE/...` commands.

Pressing the `ENTER` button on the mouse requires a stable hand. If you press the `ENTER` button and only slightly move the mouse by doing so, this will be interpreted by the display server as a *Cursor Move* instead. Therefore, it may be safer to move the cursor via the mouse but get the cursor input with the `EXECUTE` button.
You can also create a special *cursor window* for each display. Then, the area in the display window in cursor focus is also shown in that cursor window with a fixed magnification. Commands associated with cursors are:

`GET/CURSOR` - *read cursor positions*
`SET/CURSOR` - *define a specific cursor shape*
`CREATE/CURSOR_WINDOW` - *create cursor w. for existing display*
`DELETE/CURSOR_WINDOW` - *delete a cursor w.*

Look–Up Tables

Look–up tables or LUTs are the tables that map the data in the image memory into colours on the display when the system is used in PseudoColor mode.
In contrast to hardware display systems, the size of the LUT is not constant but depends upon how many colors are already used by other X applications running already. If less colors than a given *LUTmin* value are available, a so called *private* LUT is generated. This LUT is then only applicable when the focus is in that display window, changing the colors of all other windows in a nice psychedelic way (the *LUTmin* value is specified in the `INITIALIZE/DISPLAY` command).

Commands exist to load LUTs into the image display, to modify the LUTs interactively and to read back LUTs from the image display.

In RGB mode the unique LUT used is a ramp in the three colors.

Some of the existing LUTs for PseudoColor mode are:

```
backgr
color
heat
idl4, idl5, ...4, idl5, ...            !inspired by the IDL system
light
```

```
pastel
pseudo1, pseudo2
rainbow, rainbow1 ... rainbow4
random, random1 ... random4
smooth
staircase
stairs8
```

Use the command `TUTORIAL/LUT` to see how some of the available LUTs actually look like and how to modify a LUT interactively.

The main commands associated with LUTs are:

`LOAD/LUT` - *load a look–up table*
`GET/LUT` - *read back a look–up table*
`MODIFY/LUT` - *interactively modify a look–up table*
`CLEAR/LUT` - *bypass the look–up table*
`SET/LUT` - *pass through a look–up table*
`DISPLAY/LUT` - *show the look–up table as a colour bar*
`CREATE/LUT` - *create a look–up table using the HSI colour model*

Via the MIDAS application procedure (i.e. a procedure executed via the MIDAS command `@a`) `vertlut.prg` you can display the active LUT in a vertical bar at the left/right of the window with tick marks and intensities.

Commands like `MODIFY/LUT` need a displacement. Use the *arrow* keys to move left, right and down (to thin) or up (to thicken) a colour bar. Again the speed of the movements is controlled via the keys 0, 1, ..., 9.

Intensity Transformation Tables

In addition to a LUT you can also apply an intensity transformation table which maps pixels through a table before using them as LUT index. Using ITTs enables you to apply modifications to the displayed image without actually changing the data in the image channels. The ITTs are emulated by convolving the ITT values with the current LUT. Like for LUTs, interactive modification of an ITT is done via the *arrow* keys.
The main commands which control the ITT functions are:

`LOAD/ITT` - *load an ITT*
`GET/ITT` - *read back an ITT*
`MODIFY/ITT` - *modify the ITT interactively*
`CLEAR/ITT` - *bypass the ITT*
`SET/ITT` - *pass through an ITT*

To display the ITT, use `DISPLAY/LUT` which shows the combined effect of LUT and ITT. Some of the currently available ITT tables are:

```
ramp
neg
expo
log
neglog
jigsaw
staircase
```

With the command `TUTORIAL/ITT` you can see the effect of ITTs and modify the ITTs interactively.

Graphics

Graphics can be done either in an image channel or in the *overlay* channel which is associated with the display window. The overlay channel is always the highest numbered channel of a display and put on top of any displayed image channel. However, it is not implemented like an image channel. Therefore, only drawing and labeling functions are supported (i.e. you cannot load an image into the overlay channel).

All plotting functionality is provided via the family of `DRAW` commands for different shapes. For writing text you choose a specific character font via a font no.
This number refers to a font name as stored in the ASCII file
MIDASHOME/`MIDVERS/systab/bin/x11fonts.dat` (Unix syntax).
Three fixed font sizes as well as one dynamic font are supported for writing text. The fonts for the fixed sizes are chosen once via the `INITIALIZE/DISPLAY` command, the dynamic font can be set with each individual labeling command. [1]

The application procedure `showfonts.prg` lets you view the available fonts, as well as create the ASCII fonts file. The currently available commands associated with the use of the overlay channel are:

`SET/OVERLAY` - *enable the overlay of graphics on top of the image*
`CLEAR/OVERLAY` - *disable the overlay*
`CLEAR/CHANNEL overlay` - *clear contents of overlay channel*
`DRAW/...` - *draw a geometric shape like* `CIRCLE`, `RECTANGLE`, *etc. in the overlay plane*
`LABEL/DISPLAY` - *write a string into the overlay channel*

Alpha Memory

Each display window may or may not have an associated alpha memory depending on the options in the `CREATE/DISPLAY` command.

[1] Please, take care not to refresh the display window when using different dynamic fonts. Only the last chosen dynamic font is active and will be used for all text written with any dynamic font before.

The alpha memory is attached to the bottom of the display window and consists of 3 lines, the number of characters per line depends on the width of the image display.
Commands associated with the alpha memory are:

`LABEL/DISPLAY` - *write a string into the alpha memory*
`CLEAR/ALPHA` - *clear the alpha memory*

Graphical User Interface – XDisplay

There exists a graphical user interface (GUI), `XDisplay` for interaction with a MIDAS display window which is activated via the command `CREATE/GUI display`. This GUI is aimed at the casual MIDAS user who is not very familiar with the commands which provide specific display–related functions. It supports most of the display commands of MIDAS and also lists all available LUTs and ITTs.

6.2.2 Image Hardcopy

A hardcopy of the image shown on the image display merged with the contents of the overlay channel can be made via `COPY/DISPLAY`.
Grayscale and color hardcopies can be produced only on Laser printers which support `PostScript`. Typical command sequences for getting an image hardcopy are:

```
LOAD/IMAGE frame                 ! load image into image display
COPY/DISPLAY                     ! make hardcopy of screen
```

or

```
ASSIGN/DISPLAY device            ! assign hardcopy device as display
LOAD/IMAGE frame                 ! load image directly to hardcopy
```

where available devices and special parameters are described in Appendix C. In addition, a set of device specific commands are normally defined to set it up and to get status information (see Appendix C).

The command `COPY/DISPLAY` first creates a Postscript file of the contents of the display and then prints this file on the specified hardcopy device. Thus, you can also produce just a Postscript file for inclusion in another Postscript document.

Chapter 7

Data Exchange Format

This chapter describes how to exchange data between MIDAS and other systems. MIDAS supports the **F**lexible **I**mage **T**ransport **S**ystem (FITS) as the official format for data interchange and long term storage. The FITS format is recommended by the IAU for exchange of digital information between astronomical institutes and provides a computer independent description of the data.

It is important to recognise that MIDAS officially only supports the FITS format for data exchange. For backwards compatibility, MIDAS can also read the IHAP format which was used by the old ESO image processing system and some data acquisition systems at La Silla. Other formats are not supported officially and no help will be provided to access such data.

Note

The data representation of internal MIDAS files is machine dependent in order to optimise performance. Further, their layout may change with time. The use of the FITS format for storage and exchange will always ensure that proper conversions are made. This is not the case with other formats. It is safer for you to save your data in FITS format, especially if you intend keeping them for a longer time.

For standard text files such as programs, procedures and ASCII data files, operating systems utilities can be used (*e.g.* `tar`). Text files can also be saved as FITS headers using ASCII-catalogues in MIDAS, however, such files can only be decoded by the MIDAS FITS reader.

The MIDAS commands for data exchange (`INTAPE`, `OUTTAPE`) can convert the FITS files directly to/from either sequence tape devices or disks. It is therefore also possible to convert to/from FITS on disk and afterward use general utilities (*e.g.* `tar` or `ftp`) to do the actual transfer. In that case, proper options must be used to ensure that the files are transferred in binary mode and with correct blocking factors.

7.1 FITS Format

The FITS format provides a general way to encode both a definition of data and the data themselves in a machine independent form. The original definition is published in a number of articles, see Wells et al. [1], Greisen et al. [2], Grosbøl et al. [3] and Harten et al. [4]. A good general introduction to FITS can be found in [5] while a concise description is given in [6] both available through the NASA FITS Support Office and its anonymous ftp archive at Internet node `nssdca.gsfc.nasa.gov`.

7.1.1 Structure of FITS files

A FITS file contains a sequence of logical header/data units (HDU) which all start with a set of header records describing the following data records. The logical record length of a FITS file is always 2880 bytes of 8 bits. Both header and data sections start in a new logical record. FITS headers are encoded in ASCII as 80 character card images each starting with an 8 character keyword defining the type of information contained on the card. The card images follow each other directly without any end-of-line character which means that many standard text processing tools may have problems. Values of parameters are decoded using standard FORTRAN–77 rules. They describe in detail the data following the header records. Since a single FITS file may have many HDU's each corresponding to a data set (*e.g.* an image or a table), the translation of it may produce several result frames. After the last HDU in the file additional records may exist.

The basic FITS paper [1] specified both a logical and physical record length of 2880 bytes. The increasing volume of data and higher recording densities made this physical record size inefficient. To increase storage efficiency and make use of new recording media such as optical disks and helical scan devices, the FITS standard was extended to allow physical blocking factors different from one [3]. The allowed range of blocking factors is explicitly defined for a given media. For most tape media, factors between 1 and 10 are allowed giving a maximum physical block length of 28800 bytes. Each file terminates with a tape-mark, and the last file on tape terminates with a double tape-mark *i.e.* end of information.

7.1.2 FITS data-types and extensions

The FITS header specifies both the format and size of the data records following. The data representation is defined by the value of the `BITPIX` keyword and can have the values given in Table 7.1.2. The MIDAS data format closest matching it is chosen by default. Single precision real values are used when the explicit scaling is given in the FITS file (*i.e.* by the keywords `BSCALE` and `BZERO`).

When writing FITS files, their data type will also be the one closest to the internal representation except if *basic* FITS formats are explicitly requested by an option. In that case, `BITPIX` will be one of the originally allowed values (*i.e.* 8, 16 or 32) or for tables the `ASCII` format.

Each HDU in a FITS file will normally correspond to a single MIDAS file. HDU's with no associated data are not stored. Besides the prime HDU which either is a simple

BITPIX value	Data representation	Scaling	MIDAS data type
8	8-bit unsigned integer (ASCII)	No	`D_I1_FORMAT`
		Yes	`D_R4_FORMAT`
16	16-bit twos-complement integer	No	`D_I2_FORMAT`
		Yes	`D_R4_FORMAT`
32	32-bit twos-complement integer	No	`D_I4_FORMAT`
		Yes	`D_R4_FORMAT`
-32	32-bit IEEE floating point	-	`D_R4_FORMAT`
-64	64-bit IEEE floating point	-	`D_R8_FORMAT`

Table 7.1: Relation between FITS and MIDAS data types

data matrix or a random group structure a number of extension are defined. Currently, the extensions listed in Table 7.1.2 are translated by MIDAS whereas other 'unknown' extensions are skipped. Text and MIDAS-`fit` files can also be stored in FITS. The content of these file types is stored as FITS headers using a special MIDAS conversion. Thus, other FITS readers may not be able to retrieve the full information.

7.1.3 FITS keywords

Keywords in the FITS headers are stored in MIDAS files as descriptors. The naming conventions are not identical. It is therefore necessary to perform a mapping between them. In general, the MIDAS descriptors will get the same name and data type as the FITS keyword. Illegal characters in FITS keywords will be substituted by underscore ('_'). Keywords which describe the data structures are handled in a special way. Special name mapping is given in the following sections.

It is simple to map FITS keywords into MIDAS descriptors because the latter can have longer names and store arrays of values. The other way a direct translation is not always possible. If the MIDAS descriptor is compatible with a prime FITS keyword (*i.e.* its name is 8 or less characters and its value consists of only one number or string), it

FITS Extension type →	MIDAS frame
`prime`	image (`.bdf`)
`random groups`	image (`.bdf`) + table (`.tbl`)
`TABLE`	table (`.tbl`)
`BINTABLE`	table (`.tbl`)
`IMAGE`	image (`.bdf`)

Table 7.2: Relation between FITS Extensions and MIDAS frame types

FITS keyword	MIDAS descriptor	FITS keyword	MIDAS descriptor
NAXIS	NAXIS	OBJECT	IDENT
NAXISn	NPIX(n)	DATE-OBS	O_TIME(1)
CRPIXn	REFPIX(n)	DATAMIN	LHCUTS(3)
CRVALn	START(n)*	DATAMAX	LHCUTS(4)
CDELTn	STEP(n)	EPOCH	O_POS(3)
CROTAn	ROTA(n)	EQUINOX	O_POS(3)
BUNIT	CUNIT(1)		
CTYPEn	CUNIT(n+1)		

Table 7.3: Name translation between standard FITS keywords and MIDAS descriptors. Asterisk indicates that data value is transformed (see text).

will be written as a prime FITS keyword. Care must be taken to avoid that MIDAS descriptors with names overlapping standard FITS keywords names are used. This may lead to unpredictable results when such frames are saved in FITS format and restored. Descriptors which contain a full stop ('.') or start with underscore ('_') are assumed to be hierarchical and will be store as `HIERARCH` cards (see Section 7.1.3). Also in this case, the descriptor must be single values.

All other MIDAS descriptors are encoded in a special section of the header as *HISTORY* cards. The format used preserves all the descriptor information but is only known to MIDAS and may not be decoded by other readers.

Standard keywords

The FITS format specifies a number of "standard" keywords with well defined meaning. They can be classified in two groups namely: a) data structure definition, and b) general description. The first group is translated into special MIDAS descriptors for the data structure in question. The conversion for image files is given in Table 7.1.3. The `START` descriptor of a MIDAS image refers to the first pixel and is computed from the values `CRVAL` and `CRPIX` for the reference pixel as given in the FITS header. The actual reference pixel is saved in the descriptor `REFPIX` which is necessary for non-linear coordinate systems. Each element of the descriptor `CUNIT` has 16 character. Thus, the string corresponding to the FITS keyword `CTYPE1` is stored in `CUNIT(17:31)` as it is viewed as a single string of type `C*1`.

Tables have more complicated structures which make the translation less direct. With respect to other keywords, images and tables are treated in the same way. The data types of these descriptors are checked and must correspond to the FITS standard.

FITS keyword	Data type	MIDAS descriptor	Remarks
MJD-OBJ	Real	O_TIME(4)	Modified Julian Date of start of exposure
TM-START	Real	O_TIME(5)	UT of start of exposure, in sec. after midnight
TM-END	Real	-	UT of end of exposure
EXPTIME	Real	O_TIME(7)	Exposure time in sec.
RA	Real	O_POS(1)	Right Ascension in degrees
DEC	Real	O_POS(2)	Declination in degrees
POSTN-RA	Real	O_POS(1)	Right Ascension in degrees
POSTN-DE	Real	O_POS(2)	Declination in degrees
AIRMASS	Real	O_AIRM	Mean airmass of exposure
FILENAME	String	FILENAME	Original file name
ESO-LOG	String	ESO-LOG	Observing log

Table 7.4: Name translation between special non-standard FITS keywords and MIDAS descriptors.

Non-standard keywords

Besides the standard FITS keywords defined in the FITS documents, it is possible to use other, non-standard keywords. Such non-standard keywords have no global definition and may be interpreted differently by different FITS readers.

A number of non-standard keywords has been used by ESO *e.g.* for definition of frames acquired at the La Silla observatory. They are given in Table 7.1.3 and should not be used with other definitions. An extended set of FITS keywords are defined for describing data from the VLT (see [7]).

Due to the freedom to use non-standard keywords it is important to know their exact definitions when transferring data between different systems.

Hierarchical keywords

To avoid possible misinterpretations and naming conflicts for keywords describing acquisition parameters, ESO has adopted a hierarchical keyword convention for this purpose using the keyword `HIERARCH`. Such keywords have the following syntax:

```
HIERARCH domain level-1 ...  level-n keyword = value / comment
```

where the *domain* always is `ESO`. Several hierarchical levels may exist with a keyword and associated value at the lowest level. Naming and parameter follow the general FITS rules when applicable (*e.g.* max. 8 characters). Such keywords are mapped to MIDAS descriptors with names starting with the *domain*, then all *levels* and ending with the *keyword* where the components are separated with the full stop character ('.'). The hierarchical keyword:

```
HIERARCH ESO TEL FOCU SCALE = 1.489 / Focus length (deg/m) = 5.36"/mm
```

will yield the MIDAS descriptor name `ESO.TEL.FOCU.SCALE` while

```
HIERARCH ESO INS OPTI-3 ID = 'ESO#427 ' / Optical element identifier
```

will become the descriptor `ESO.INS.OPTI-3.ID`. When writing such descriptors out again, the reverse translation is applied.

In MIDAS releases prior to `96NOV`, a special encoding scheme was used to translate hierarchical keywords name to MIDAS descriptor name shorter than 16 characters. This is not needed anymore since long descriptor name are supported from the `96NOV` release. For backwards compatibility, a `C`-flag is avaliable in the `INTAPE/FITS` command providing the pre-96NOV name mapping.

The hierarchical structure provides a convenient and clear way to separate information concerning different subsystems. The definition of FITS keywords used by ESO for data acquisition can be found in the *Data Interface Control Document* [7]. This document also gives a full definition of all hierarchical keywords in the ESO-domain. FITS readers which do not know this convention should save the `HIERARCH`-keywords as comments following standard FITS rules.

7.1.4 Restrictions

The MIDAS reader of FITS tapes accepts most of the FITS formats including standard extensions. The MIDAS implementation has the following restrictions:

- the maximum number of axes in images is 16,
- maximally 512 columns are allowed for tables
- matrix and variable size array conventions for `BINTABLE` are not available,
- information in headers without associated data is not stored to avoid creation of empty data files (*e.g.* for simple table files).

7.2 IHAP Format

The IHAP format is defined in the IHAP manual (see version April 1990, Appendix A). It is the internal format of the IHAP system and may also be used by some old data acquisition systems at the La Silla observatory.

7.2.1 Translation of IHAP header

The IHAP header contains the description of the data in a fixed, binary format. With reference to the IHAP manual, the translation of this information into MIDAS descriptors are given in Table 7.2.1. IHAP defines the end of exposure which is converted to exposure

IHAP header	MIDAS descriptor	IHAP header	MIDAS descriptor
DRASC	O_POS(1)	JUSR1	IHAPUSER(1)
DECLN	O_POS(2)	JUSR2	IHAPUSER(2)
JYEAR/JDAT	O_TIME(1)	DUSR1	IHAPUSER(3)
DCRAT	O_TIME(5)	DUSR2	IHAPUSER(4)
DSTTM	O_TIME(6)	AHGC	LHCUTS(2)
DENTM	O_TIME(7)*	ALWC	LHCUTS(1)
JDENT	IDENT	DAIRM	O_AIRM
DXSTR/DYSTR	START		
DXSTP/DYSTP	STEP		
DXEND/DYEND	NPIX*		

Table 7.5: Name translation between IHAP header and MIDAS descriptors. Asterisk indicates that the value is transformed (see text).

time. Also, start and end are given for image axes instead of start and number of pixels In addition to the main header, keywords and comments may exist. They are saved in the descriptor O_COM in ASCII format.

7.2.2 Restrictions

The main limitations of the MIDAS reader of IHAP formats are the following:

- only standard image formats are decoded,
- only tapes written with code 1 specifications are decoded.

Other formats such as tables and special data acquisition, can only be read by an IHAP system after which they can be converted to FITS by IHAP itself.

7.3 Conversion between FITS and internal format

The FITS conversion commands INTAPE and OUTTAPE can both access FITS files on disk or directly on tape devices. Thus, it is in principle possible to use general system utilities to transfer the FITS files between disk and tape. When using operating system utilities special care must be taken to ensure that the files on the tape conform to FITS blocking specification - it is by no means obvious!

The syntax of FITS conversion commands is as follows:

```
INTAPE file_list id device [flag]
OUTTAPE cat[,list] device [flag] [dens,block] [type]
```

These are described in more detail in the following sections.

In order to save disk space, it may be desirable to compress the files. For this purpose, operating system utilities may be used (*e.g.* `compress` for UNIX). The steps to achieve this are as follows:

- Use `OUTTAPE` to write file to disk in FITS format.
- Compress FITS file on disk *e.g.* `compress`.
- Use system utility to save compressed file on tape *e.g.* `tar`.

To retrieve files execute the above process in reverse order.

7.3.1 Devices

The *device* in the `INTAPE/OUTTAPE` commands may either refer to a tape device, a disk file or a disk file prefix (possibly including the full path name). If the first 4 characters of the device name are the string '`tape`' (case independent), it will be assumed to be a logical tape name. The environment variable of the same name will then be used for the logical to physical name translation. Otherwise, the name will be assumed to be a physical name.

The name of a tape device may contain the host name in which case a remote device will be accessed. This will work only if the MIDAS tape-server has been installed. The full name is then given as '`host:device`' *e.g.* for a device on a UNIX system `ws1` it could be '`ws1:/dev/nrst8`'. For access to local devices the physical device name can be given directly. When writing on a tape device, it must be set to *write enable* either by inserting a write ring or changing a switch on the cartridge.

A single file can be specified by including its extension otherwise a name is assumed to be a file prefix. When a file prefix is used the full file is made up of the prefix, the four digit sequence number and the file extension '`.mt`'.

7.3.2 File naming

The file name is generated from the prefix and the sequence number. A FITS file may contain several extensions each of which may result in a MIDAS file. If more than one file of the same type (*e.g.* image or table) is associated to the same FITS file, an additional letter is appended to the name. The fourth table file in the FITS file '`obs0023.mt`' would get the name '`obs0023c.tbl`'.

The original file name is normally available in the descriptor `FILENAME` which can be used for renaming it (see `RENAME/FILE` or `INDISK/FITS`). This is not done by default because it could overwrite existing files.

7.3.3 Reading FITS

The `INTAPE`-command is used to convert FITS (or IHAP) files to the internal MIDAS format:

```
INTAPE file_list id device [flag]
```

where `file_list` is a list of absolute position numbers of files (if input is from tape) to be read with 1 being the first file (*e.g.* 1, 3, 5, 50-60 would read from the 1st, 3rd, 5th file plus files 50 through 60 included), and `id` is an identification prefix of the names of the files created. Tape devices will be rewound if the driver does not maintain the absolute position of the tape. This may take a significant amount of time on some devices. It is easier to read many files from disk (with `INTAPE`) if their names are constructed with the given prefix, a four digit number and the extension '`.mt`'. It is also possible to read a single file from disk by specifying its full name including extension in which case any extension may be given.

The `flag` is a list of three one character flags which specifies the amount of information listed on the senior terminal and in the log file and the storage format of the data (see help file for full description).

The `INTAPE` command can be used to read FITS files directly from disk. However, it is not convenient due to the naming scheme used. For this purpose, it is better to use:

```
INDISK/FITS in_files [out_spec] [option]
```

where `in_files` specifies the FITS files to be converted (note: wildcard characters are supported). The `out_spec` can be used to control the names of the resulting FITS files. If the original file name is stored in the `FILENAME` keyword in the FITS header, it is possible to use this name for the output file by specifying the option `ON`. For a more detailed description of the command check with the help-text.

7.3.4 Writing FITS

Translation of MIDAS files to FITS format is done by the `OUTTAPE` command:

```
OUTTAPE cat[,list] device [flag] [dens,block] [type]
```

where the `cat[,list]` is a catalogue of files to be written with an optional list of numbers (see `CREATE/xCAT` for creation of catalogues). It can be defaulted by giving either '`*`' or '`?`' in which case all files in catalogues set by the `SET/xCAT` command are written out. A single file can be written by specifying it with its full name including extension. The `device` may specify an actual tape device, a prefix for disk files or an explicit file name for a single file.

The `flag` is an optional list of three one-character flags specifying the append mode, the amount of information listed on the senior terminal and in the log file and if the `LHCUTS` descriptor in the file should be used for scaling (see help file for full description).

Although the `OUTTAPE` command in principle can be used to write FITS files directly to disk, it is not convenient due to the naming scheme used. A better command for this purpose is:

```
OUTDISK/FITS in_files [out_spec] [option] [out_type]
```

where `in_files` specifies the files to be converted to FITS (note: wildcard characters are supported). The `out_spec` can be used to control the names of the resulting FITS files where the basename of the input frame can be specified by given the string `NAME=INPUT`. For a more detailed description of the command check with the help-text.

Note

The default options of the `OUTTAPE` *command will start at the current tape position. This may over-write previous data on the tape. Be sure to use the* `append` *flag if files have to be added to the tape. Or position the tape at end-of-information using operating system utilities before writing new files. Also use a* **non-rewind** *device name (e.g.* `/dev/nst1`*) when specifying a physical device - otherwise the operating system may rewind the tape by itself!*

The `dens,block` parameter can specify the tape density (*e.g.* 1600 or 6250bpi) and a physical blocking factor in the range 1-10. By default, a blocking factor of 10 is used. Note that some old FITS readers may not be able to read blocked FITS files (*e.g.* IHAP) in which case a blocking on 1 must be given explicitly. The tape density is used only for 9-track 1/2-inch tapes. Some 1/2-inch tape devices require the density also to be set manually on the drive.

The `type` flag is used to specify the type of FITS format to write where '`B`' indicates basic FITS *i.e.* with integer format only. The default is '`O`' for original including floating point representation.

Bibliography

[1] D.C. Wells, E.W. Greisen and R.H. Harten, 1981: *Astron. Astrophys. Suppl. Ser.*, **44**, p. 363

[2] E.W. Greisen and R.H. Harten, 1981: *Astron. Astrophys. Suppl. Ser.*, **44**, p. 371

[3] P. Grosbøl, R.H. Harten, E.W. Greisen and D.C. Wells, 1988: *Astron. Astrophys. Suppl. Ser.*, **73**, p. 359

[4] R.H. Harten, P. Grosbøl, E.W. Greisen and D.C. Wells, 1988: *Astron. Astrophys. Suppl. Ser.*, **73**, p. 365

[5] NOST 1992: 'A User's Guide for the Flexible Image Transport System (FITS)', NASA/Science Office of Standards and Technology, Code 633.2, NASA Goddard Space Flight Center.

[6] NOST 100-1.0, 1993: 'Definition of the Flexible Image Transport System (FITS)', NASA/Science Office of Standards and Technology, Code 633.2, NASA Goddard Space Flight Center.

[7] Data Interface Control Document, 1997: `GEN-SPE-ESO-19400-794/1.1`, ESO Garching.

Chapter 8

Fitting of Data

This chapter deals with the modelling and the analysis of image and table data by fitting non–linear functions, using least squares approximation. The different non–linear least squares methods implemented in MIDAS are first shortly described and discussed. The MIDAS commands dealing with functions or linear combination of functions and with the modelling process are then presented.

The basic scheme under these commands is to provide the necessary tools to define the functions entering in the fit, to give initial guesses for the parameters and, in iterations controlled by the user, find the optimal parameters of the functions. These parameters can be used to generate fitted data either as images or as columns in tabular form.

Due to the nature of the methods, it is recommended to use these commands in fitting problems involving small amounts of data. For analysis involving large amounts of data, like full CCD images, there are algorithms, in the context of 2D–photometry, optimized for special purpose analyses. A tutorial command (`TUTORIAL/FIT`) has been introduced in order to show the capabilities of the package.

A brief description of the implemented methods is included in section 8.1. Section 8.2 describes how to specify functions in the fit. Section 8.3 describes how to include external functions. The usage of the commands is illustrated in section 8.4. The output of the programs and their possible interpretation are discussed in section 8.5. An example is presented in section 8.6, it may be convenient for first time users to run the command `TUTORIAL/FIT` while reading this section. Section 8.7 contains a summary of the commands. Finally, the functions supported in the current version are listed in section 8.8. References can be found in section 8.9.

8.1 Outline of the Available Methods

Let $y(x, a)$ be a function where $x = (x_1, \ldots, x_n) \in \mathrm{I\!R}^{\mathrm{n}}$ are the independent variables and $a \in A \subset \mathrm{I\!R}^{\mathrm{p}}$ are the p parameters lying in the domain A. If A is not the whole space $\mathrm{I\!R}^{\mathrm{p}}$, the problem is said to be constrained.

If a situation can be observed by a set of events $(y^{(i)}, x^{(i)})i = 1, \ldots, m$, i.e. a set of couples representing the measured dependant and variables, it is possible to deduce the value of the parameters of your model $y(x, a)$ corresponding to that situation. As the measurements are generally given with some error, it is impossible to get the exact value of the parameters but only an estimation of them. Estimating is in some sense finding the most likely value of the parameters. Much more events than parameters are in general necessary.

In a linear problem, if the errors on the observations have a gaussian distribution, the "Maximum Likelihood Principle" gives you the "best estimate" of the parameters as the solution of the so-called "Least Squares Minimization" that follows:

$$\min_{a \in A} \; \chi^2 \; (a)$$

with

$$\chi^2 \; (a) \; = \; \sum_i \; w^{(i)} \; [\; y^{(i)} \; - \; y(x^{(i)}, a) \;]^2$$

The expected variance of the so–computed estimator is minimum among all approximation methods and is therefore called in statistics an "efficient estimator".

The quantities

$$r^{(i)}(a) \; = \; \sqrt{w^{(i)}} \; [\; y^{(i)} \; - \; y(x^{(i)}, a) \;]$$

are named the residuals and $w^{(i)}$ the weight of the i^{th} observation that can be, for instance, the inverse of the computed variance of the observation.

If $y(x, a)$ depends linearly on each parameter a_j, the problem is also known as a Linear Regression and is solved in MIDAS by the command `REGRESSION`. This chapter deals with $y(x, a)$ which have a non-linear dependance in a.

Let us now introduce some mathematical notations. Let $g(a)$ and $H(a)$ be respectively the gradient and the Hessian matrix of the function $\chi^2(a)$. They can be expressed by

$$g(a) \; = \; 2 \; J(a)^T \; r(a) \qquad and$$

$$H(a) \; = \; 2 \; (J(a)^T \; J(a) \; + \; B(a))$$

where $r(a)$ is the residuals vector

$$r(a) \; = \; (r^{(1)}(a), \ldots, r^{(m)}(a)) \quad ,$$

$J(a)$ the Jacobian matrix of $r(a)$ i.e.

$$J(a)_{ij} \; = \frac{\partial r^{(i)}}{\partial a_j}$$

and $B(a)$ is

$$B(a) \; = \; \sum_i \; r^{(i)}(a) \; H_i(a)$$

with $H_i(a)$, the Hessian matrix of $r^{(i)}(a)$.

In the rest of the chapter, all the functions are supposed to be differentiable if they are applied the derivation operator even when this condition is not necessary for the convergence of the algorithm.

A certain number of numerical methods have been developed to solve the non–linear least squares problem, four have so far been implemented in MIDAS. A complete description of these algorithms can be found in [1] and [3], the present document will only give a basic introduction.

8.1.1 The Newton–Raphson Method.

This is the simplest one. The necessary condition for the function $\chi^2(a)$ to have an extremum is that the partial derivatives vanish i.e.

$$\sum_i r^{(i)} \frac{\partial r^{(i)}}{\partial a_j} = 0 \quad (j = 1, \ldots, p)$$

or, equivalently,

$$J(a)^T \; r(a) \; = 0 \quad .$$

This is usually a system of non–linear equations that, numerically, can be solved using the Newton–Raphson's method also called in the one–dimensional case the tangents method. The Taylor development of the function limited to the first order is taken around some initial guesses of the parameters. The resulting linear system

$$J(a^{(k)})^T \; J(a^{(k)}) \; \Delta a^{(k)} \;\; = \;\; - \, J(a^{(k)}) \; r(a^{(k)})$$

gives thus a correction to the solution and

$$a^{(k+1)} \;\; = \;\; a^{(k)} \;\; + \;\; \gamma \; \Delta a^{(k)}$$

is taken as the new approximation of the optimum. The relaxation factor γ is a parameter of the method. The convergence of the process towards the solution of the non–linear minimization problem has been proven for locally convex $\chi^2(a)$ or under other assumptions impossible to detail here. These conditions are not generally fulfilled in real problems. Moreover, the algorithm ignores the second order conditions and therefore, may end on a saddle point or never converge. Two different relaxation factors may lead to different solutions or one may give convergence and the other one not. No general rule can be given for the choice of a good relaxation factor.

8.1.2 The Modified Gauss–Newton Method.

From a first guess of the parameters $a^{(1)}$, a sequence $a^{(2)}, a^{(3)}, \ldots$ is generated and is intended to converge to a local minimum of $\chi^2(a)$. At each iteration, one computes

$$a^{(k+1)} \;\; = \;\; a^{(k)} \; + \alpha^{(k)} \; d^{(k)}$$

where $d^{(k)}$ is a certain descent direction and $\alpha^{(k)}$ is a real coefficient which is chosen such that $\chi^2(a^{(k)} + \alpha^{(k)} \; d^{(k)})$ is approximately minimum. The direction $d^{(k)}$ is ideally the solution of the Newton equation

$$H(a^{(k)}) \; d^{(k)} \;\; = \;\; - \, g(a^{(k)})$$

which can also be rewritten

$$[J(a^{(k)})^T\ J(a^{(k)})\ +\ B(a^{(k)})]\ d^{(k)}\ =\ -\ J(a^{(k)})\ r(a^{(k)})\ .$$

Neglecting the second derivatives matrix $B(a^{(k)})$, we obtain the "normal equations" and the Gauss–Newton direction

$$J(a^{(k)})^T\ J(a^{(k)})\ d^{(k)}\ =\ -\ J(a^{(k)})\ r(a^{(k)})$$

This so–called Gauss–Newton method is intended for problems where $\|B(a)\|$ is small. If the Jacobian $J(a)$ is singular or near singular or if $\|r(a)\|$ is very large (the so–called large residuals problem), the Gauss–Newton equation is not a good approximation of the normal equations and the convergence is not guaranteed.

The algorithm implemented here is a modification of that Gauss–Newton method, that allows convergence even for rank deficient Jacobians or for large residuals. The Gauss–Newton direction is computed in $V_1\ =\ \Im m\ [J(a^{(k)})^T\ J(a^{(k)})]$, the invariant space corresponding to the non–null eigenvalues. A correction is taken in V_2, the orthogonal of V_1, according to the second derivatives if the decrease of the objective function at the last iteration is considered too small. The Hessian matrix is estimated using finite differences of the gradient.

This method requires the availability of the derivatives and as the number of gradient evaluations is almost p at each iteration, it is recommended for problems with a small number of parameters, let us say $p\ \leq\ 10$

8.1.3 The Quasi–Newton Method.

This is identical to the modified Gauss–Newton method, except in the way that the Hessian matrix is approximated.

This matrix is first initiated to zero. At each iteration, a new estimation of the Hessian is obtained by adding a rank one or two correction matrix to the last estimate such that $H^{(k+1)}$, the estimate of the Hessian matrix at the $k+1^{th}$ iteration, satisfies

$$(J(a^{(k+1)})^T\ J(a^{(k+1)})\ +\ H^{(k+1)})\ (x^{(k+1)} - x^{(k)})\ =\ J(a^{(k+1)})\ r(a^{(k+1)})\ -\ J(a^{(k)})\ r(a^{(k)})$$

The so-called BFGS updating formulas are applied in this algorithm

$$H^{(0)}\ =\ 0 \qquad H^{(k+1)}\ =\ H^{(k)}\ +\ C^{(k)}$$

$$C^{(k)} = \frac{1}{\alpha^{(k)} y^{(k)T} d^{(k)}} y^{(k)} y^{(k)T} - \frac{1}{d^{(k)T} W^{(k)} d^{(k)}} W^{(k)} d^{(k)} d^{(k)T} W^{(k)}$$

where

$$W^{(k)} = J(a^{(k+1)})^T\ J(a^{(k+1)}) + H^{(k)}$$

and

$$y^{(k)} = J(a^{(k+1)}) r(a^{(k+1)}) - J(a^{(k)}) r(a^{(k)})\ ,$$

please see Gill, Murray and Pitfield (1972) for more details. After some iterations and around the optimum, $H^{(k)}$ converges to the Hessian.

This method requires the knowledge of the derivatives and, as the gradients are only computed once per iteration and consequently, the Hessian is more roughly approximated than with the modified Gauss–Newton method, this is better designed for a great number of parameters i.e. $p > 10$.

8.1.4 The Corrected Gauss–Newton No Derivatives.

This method is identical to the Gauss–Newton method where the Jacobian is estimated by finite differences and the Hessian by second order differences.

It does not require the programming of the derivatives but makes a lot of function computations. Its use has to be restricted to problems where the derivatives are really too difficult to write. It is slower and less precise than the two last algorithms.

8.2 Function Specification

The functions to be fitted to data are linear combinations of a set of, so called, "basic" functions. Basic functions are either defined in the system or defined by the user as external FORTRAN routines. The actual combination of basic functions is defined via an interactive editor, (MIDAS command `EDIT/FIT`)

Basic functions are specified by the name, the independent variable(s) and parameter(s), with optional guesses for the parameters, following the syntax:

```
name(var1[,...];par1[,...]) [par1=value] ...
```

The function name `name` defines the basic function used, it can be a system function, as defined in the table 8.1, or a external function with name `USER00`, ..., `USER09`. In this case, the corresponding file(s) `USER00.FOR`, ..., `USER09.FOR` will exist in the working area and will contain the definition of the routines following the syntax described in the next section.

The number of independent variables of the function is determined by the string `var1[,...]`. The actual names of the independent variables are considered as dummy names but their number has to coincide with the actual number of parameters of the function. All the functions defined in a given fit **must** have the same number of independent variables.

Parameters are defined by unique names after the semicolon in the function specification. Parameters are interpreted according to the position and to the number of independent variables in the function.

A parameter is generally given a first guess on the same line, as `pari=value`, it can also be fixed to a given value or kept proportional to another parameter. The parameter is defined as fixed with the symbol @ immediately following the value as `pari=value@`. Linear constrains between parameters are defined as `pari=parj*value` or `pari=parj/value`.

According to these rules, a one dimensional gaussian function is specified with the `EDIT/FIT` command as

```
1 GAUSS(X;A,B,C) A=10.  B=3200.@ C=1.
```

where X is the dummy name of the only independent variable, the first parameter, defining the maximum of the function, is called A, initialized to 10, the second parameter, defining the position of the gaussian, is called B, and its fixed value is 3200 in world coordinates, and the FWHM is the parameter C, with initial value 1.

A linear combination of a gaussian and a Cauchy distribution, centered at the same position is specified as

```
1 GAUSS(X;A1,B1,C1) A1=10.  B1=3200.  C1=1.
2 CAUCHY(X;A2,B2,C2) A2=A1/10.  B2=B1 C2=4.
```

in this case, the maximum of the Cauchy distribution is determined by the corresponding parameter of the Gaussian.

We include in table 8.1 a summary of the system basic functions; the actual mathematical expressions, with the meaning of the function parameters are given in section 8.8.

POLY(X;A,B,...)	polynomial (1D, 2D)
LOG(X;A,B,C)	natural logarithm
EXP(X;A,B,C)	exponential
SIN(X;A,B,C)	sinus
TAN(X;A,B,C)	tangent
SINC(X;A,B,C)	sinc
SINCS(X;A,B,C)	sinc square
GAUSS(X;A,B,C)	(FWHM) Gaussian distribution (1D)
GAUSS(X,Y;A,B,C,D,E,F)	(FWHM) Gaussian distribution (2D)
GAUSSA(X;A,B,C)	(Standard) Gaussian distribution (1D)
CAUCHY(X;A,B,C)	Cauchy distribution (1D)
CAUCHY(X,Y;A,B,C,D,E,F)	Cauchy distribution (2D)
LORENTZ(X,Y;A,B,C,D,E,F)	Modified Cauchy (Lorentz) distribution
LAPLACE(X;A,B,C)	Laplace distribution
TRIANG(X;A,B,C)	Triangular distribution
POISSON(X;A,B,C)	Poisson distribution
IGAUSS(X;A,B,C)	Integrated (FWHM) Gaussian distribution (1D)
IGAUSSA(X;A,B,C)	Integrated (Standard) Gaussian distribution (1D)

Table 8.1: Basic Fit Functions

8.3 External Functions

If the set of basic functions provided by the system as listed below is not sufficient for your own purpose, it is possible to define user functions. To do this, the user has to provide the code of the function(s) as a FORTRAN routine, in his own area, in files named USER00.FOR,... USER09.FOR. The command CREATE/FUNCTION will compile the routine(s) and link them with the corresponding system programs (primitives). A library with the local definitions of the routines USER00,...,USER09 and the executable code will be created in the user area. With this scheme, it is possible to fit the external functions `USER0i` as if they where basic functions.

Here is a template to write a user defined function:

```
C+
C.NAME
C       USER0i
C
C.DESCRIPTION
C       ...
C
C.INPUT ARGUMENTS:
C   NIND          INTEGER           Number of independent variables
C   X (NIND)      REAL              Array of NIND elements with the
C                                   independent variable
C   NPAR          INTEGER           Number of parameters
C   PARAM (NPAR) DOUBLE PRECISION  Array of NPAR elements with the
C                                   values of the parameters
C
C.OUTPUT ARGUMENTS:
C   Y             DOUBLE PRECISION  Value of the function
C   DERIV (NPAR) DOUBLE PRECISION  Array of NPAR elements with the
C                                   partial derivatives of the
C                                   function for each parameter
C-
      SUBROUTINE USER0i(NIND,X,NPAR,PARAM,Y,DERIV)
      IMPLICIT NONE
C     ..
C     .. Scalar Arguments ..
      INTEGER NIND,NPAR
      DOUBLE PRECISION Y
C     ..
C     .. Array Arguments ..
      REAL X(NIND)
      DOUBLE PRECISION DERIV(NPAR),PARAM(NPAR)
C     ..
C     .. Local Scalars ..

C     ..
C     .. Local Arrays ..

C     ..
C     .. Executable Statements

      RETURN
      END
```

The variable Y must contain the value of the basic function at the parameter value PARAM and the array DERIV has to receive the value of the partial derivatives, except if the method used is CGNND (the abbreviation of Corrected Gauss–Newton No Derivative). In the user functions, it is recommended to scale the parameters in such a way that their absolute values lies in a small scale range let us say in $[0.1 + 10.]$. It is advised to use this scheme to test and debug new functions that can later on be included in the system supported set.

8.4 The Fitting Process.

The typical sequence of operations for a FITting process would be first to create the approximating function, to choose relatively to your problem and your needs the FIT options, then to execute the real least squares approximation and finally to store and view the results. This corresponds to the typical sequence of MIDAS instructions :

```
EDIT/FIT fitname
SET/FIT options
FIT/...  nfeval,prec image (or table and cols)
COMPUTE/FIT output
```

The commands have been designed so that defaults exists for almost all the parameters, (see description in Volume C of the MIDAS User Guide).

EDIT/FIT has been described in the last section.

The MIDAS command SET/FIT is used to specify the different options of the FIT command, for instance the method to be applied or the type of used functions. The instruction

```
SET/FIT METHOD=CGNND PRINT=1 WEIGHT=S FUNCT=BLACBODY FCTDEF=USER
```

declares that the Corrected Gauss–Newton no derivatives method is to be applied, that at each iteration, a display of the intermediate result will be performed, that the weighting factors are statistical, that the name of the approximating function is BLACBODY, and that this BLACBODY function which contains user defined functions has already been built in the user area. The appendix or the MIDAS interactive HELP facility will give you the complete description of the SET/FIT command.

The command SHOW/FIT displays the actual selected FIT options.

The FIT instruction is performing the least squares approximation itself. It has a slightly different syntax if the fitting concerns a table or an image.

```
FIT/TABLE nfeval[,prec,[metpar]] table :depcol[,:wgt] :indcol,...
FIT/IMAGE nfeval[,prec,[metpar]] image
```

nfeval is the maximum number of function evaluations that can be performed, prec is the precision on the parameters i.e. the program stops if

$$\|a^{(solution)} - a^{(found)}\| \le prec\ (1 + \|a^{(found)}\|) \quad ,$$

and metpar are the specific method parameters (for instance in NR : the relaxation factor). The latter have not generally to be given as they can be deduced by the program. For stiff problems, they can thus be overwritten by the user. Any non–given parameter is defaulted. Consult the appendix or use the MIDAS interactive HELP to get a complete description. For instance, the instruction

```
FIT/IMAGE 100,0.001 PROFILE
```

executed after the preceeding `SET/FIT`, will execute a non–linear least squares approximation using the `CGNND` method. The program will stop if more than 100 computations of the approximating function have been performed or if the solution has been found with a precision of 10^{-3}.

8.5 Outputs

To check any typing error or missing specification, first are displayed the options, the required precision, the maximum number of function evaluations and the method parameters.

The frequency of the intermediate displays are controlled by the `SET/FIT PRINT=iter`. It includes the display of the iteration number, the actual number of function evaluations, the sum of the squares of the residuals, the so-called reduced chi, the percentage of decrease of the reduced chi since last iteration, and, except for the NR method, the norm of the gradient and the dimension of the space V_1 spanned by the Jacobian. The reduced chi square is the

$$\frac{\chi^2(a^{(k)})}{degree\ of\ freedom}$$

In any case, this is followed by the value of the parameters. Except for the `NR` method, the value of the gradient and the singular values of the Jacobian matrix are added.

At the end, a diagnostic message telling you if the convergence was reached or if any numerical failure occurred during the algorithm. The different messages are:

```
  --> METHOD : Convergence achieved <--
*** ERR-1-METHOD : Bad initializations ... Aborting ***
*** ERR-METHOD : Likely an error in forming the derivatives ***
*** ERR-NR  : Problems in inverting matrix  ***
*** WARN-2-METHOD : No convergence reached ***
*** WARN-3-METHOD : Final parameters not really satisfactory ***
*** ERR-4-METHOD : No convergence in singular value decomposition ***
*** WARN-5-METHOD : parameters only a good estimation ***
*** ERR-i-METHOD : Final parameters are not satisfactory ***
```

In the last message i varies from 6 to 8 and the greater i is, the less reliable are the final value of the parameters. For warnings and errors numbered more than 3, it is recommended to perform another `FITting` with different initial guesses. If warning 2 is displayed, do again the `FITting` starting with the last computed value of the parameters (`nfeval` < 0 in the `FIT/...` command). If an error in the derivatives is reported, check your user functions code.

The diagnostic will be followed by the covariance matrix if you set `iter` to a negative value in `SET/FIT PRINT=`.

Finally, the found optimal value of the parameters with their estimated standard deviation are listed.

8.6 Tutorial

A tutorial procedure (TUTORIAL/FIT) shows how to use the fitting package in the simple case of a 1D–image consisting of two overlapping gaussians on top of a non–linear background with additional noise. It is recommended to run the tutorial while reading this section and if possible, on a graphic terminal.

Two functions are copied into your area if you run the example: TEST to generate the artificial data, and FUNCTION with the "model" to be fitted.

The artificial image, to be used in the example, is created as follows: First it creates a reference image, called REF, to provide the definition interval of the independent variable. Then the command COMPUTE/FUNCTION creates the 1D frame with the gaussian profiles on top of the background. Finally, some noise is added to the data. The resulting frame, PROFILE is displayed on the graphic screen.

Now the 'model' FUNCTION will be fitted to the frame PROFILE, using the command FIT/IMAGE. The 'model' was copied into your area already, but you could define it using the editor as:

```
EDIT/FIT FUNCTION
```

This command allows you to create or modify FIT–files (ref Volume C of the MIDAS User guide and Appendix C for use of the EDIT command on your terminal). In our example, the user will edit the three basic functions e.g. as follows:

```
1 GAUSS(X;A1,A2,A3) A1=50.  A2=95.  A3=45.
2 GAUSS(X;A4,A5,A6) A4=A1 A5=135.  A6=A3
3 POLY(X;A,B,C) A=0.  B=0.  C=0.
```

where two of the parameters of the second gaussian, height and FWHM, are related to the parameters of the first Gaussian.

The different fitting methods are then successively applied, changing regularly the options through the SET/FIT. The exact sequence of instructions is:

```
SHOW/FIT
FIT/IMAGE 11,1.,0.5 PROFILE FUNCTION
SET/FIT METHOD=CGNND
SET/FIT PRINT=0
FIT/IMAGE 30,.5 PROFILE FUNCTION
SET/FIT METHOD=QN
SET/FIT PRINT=-4
FIT/IMAGE 30,.5 PROFILE FUNCTION
SHOW/FIT
SET/FIT METHOD=MGN
FIT/IMAGE 30,.5 PROFILE FUNCTION
```

It is possible to compare efficiency, precision and effects.

Finally, the fitted result is computed as:

```
COMPUTE/FIT FITTED = FUNCTION
```

and the fitted frame is plotted on top of the original data.

Individual components of the fit can be selected with the command SELECT/FUNCTION. In the example, the sequence of commands

```
SELECT/FUNCTION FUNCTION 1,3
COMPUTE/FIT FIT1 = FUNCTION
SELECT/FUNCTION FUNCTION 2,3
COMPUTE/FIT FIT2 = FUNCTION
SELECT/FUNCTION FUNCTION ALL
```

is used to compute the two gaussian components on top of the background. The results are also plotted.

The full compatibility between image and tabular formats for input and output means that, in our example, the fitted parameters can be used to compute fitted values in a table, using the COMPUTE/FIT command as follows:

```
COMPUTE/FIT table :outcol = fitname(:incol)
```

where `table` is the name of a table containing the independent variable in the column `:incol`, fitted values are stored in the column `:outcol`.

To begin with, it is advised to consult the appendix or use the MIDAS interactive HELP about EDIT/FIT, SET/FIT, CREATE/FUNCTION, REPLACE/FUNCTION, FIT, FIT/IMAGE, FIT/TABLE, COMPUTE/FIT.

8.7 Command Summary

Table 8.2 summarizes the commands which are implemented in the context of functions and least squares fitting.

8.8 Basic Functions

8.8.1 Polynomials (1D and 2D)

$$\begin{aligned}\mathtt{POLY}(x;a,b,c,\ldots) &= a+bx+cx^2+\cdots \\ \mathtt{POLY}(x,y;a,b,c,\ldots) &= a+bx+cy+\cdots\end{aligned}$$

8.8.2 Logarithmic and Exponential Function

$$\begin{aligned}\mathtt{LOG}(x;a,b,c) &= a\ln(b+cx) \\ \mathtt{EXP}(x;a,b,c) &= a\exp(b+cx)\end{aligned}$$

```
COMPUTE/FIT outima [= funct[(refima)]]
COMPUTE/FIT table:out[,:error] [= funct[(:col1,...)]]
COMPUTE/FUNCTION outima = funct(refima)
COMPUTE/FUNCTION table:out = funct(:col1,...)
CREATE/FUNCTION userfunc1[,...]
EDIT/FIT [funct]
FIT/IMAGE [nfeval[,prec[,metpar]]] [image[,wgt]] [funct]
FIT/TABLE [nfeval[,prec[,metpar]]] table dep[,wgt] ind [funct]
MODIFY/FIT table seqno [funct]
REPLACE/FUNCTION userfunc1[,...]
SAVE/FIT table seqno [funct]
SELECT/FUNCTION funct number[,...]
SELECT/FUNCTION funct ALL
SET/FIT [METHOD=mname] [PRINT=iter] [WEIGHT=wgttyp] [FUNCT=fname] [FCTDEF=where]
SHOW/FIT
```

Table 8.2: Fitting Commands

8.8.3 Trigonometric Functions

$$\begin{aligned} \texttt{SIN}(x;a,b,c) &= a\sin(b+cx) \\ \texttt{TAN}(x;a,b,c) &= a\tan(b+cx) \end{aligned}$$

8.8.4 Sinc and Sinc Square

$$\texttt{SINC}(x;a,b,c) = a\sin(b+cx)/(b+cx)$$

$$\texttt{SINCS}(x;a,b,c) = a\,\mathrm{sinc}^2(b+cx)$$

8.8.5 Distributions

$$\begin{aligned}
\mathtt{GAUSS}(x;a,b,c) &= a\exp\left[-\ln 2\left(\tfrac{2(x-b)}{c}\right)^2\right] \\
\mathtt{GAUSS}(x,y;a,b,c,d,e,f) &= a\exp\left[-\ln 2\left(\tfrac{(x-b)^2}{d^2}+\tfrac{(y-c)^2}{e^2}-\tfrac{2f(x-b)(y-c)}{d\,e}\right)\right] \\
\mathtt{GAUSSA}(x;a,b,c) &= \tfrac{a}{\sqrt{(2\pi)}c}\exp\left[-\tfrac{1}{2}\left(\tfrac{(x-b)}{c}\right)^2\right] \\
\mathtt{IGAUSS}(x;a,b,c) &= a\int_{-\infty}^{x}\exp\left[-\ln 2\left(\tfrac{2(u-b)}{c}\right)^2\right]du \\
\mathtt{IGAUSSA}(x;a,b,c) &= \tfrac{a}{\sqrt{(2\pi)}c}\int_{-\infty}^{x}\exp\left[-\tfrac{1}{2}\left(\tfrac{(u-b)}{c}\right)^2\right]du \\
\mathtt{CAUCHY}(x;a,b,c) &= a\left[1+\left(\tfrac{2(x-b)}{c}\right)^2\right]^{-1} \\
\mathtt{LORENTZ}(x;a,b,c,d) &= a\left[1+\left(\tfrac{2(x-b)}{c}\right)^2\right]^{-d} \\
\mathtt{POISSON}(x;a,b,c) &= \tfrac{ab^x\exp(-b)}{\Gamma(x+1)} \\
\mathtt{LAPLACE}(x;a,b,c) &= a\exp\left[-\ln 2\left(\tfrac{2|x-b|}{c}\right)\right] \\
\mathtt{TRIANG}(x;a,b,c) &= a\left(1-\tfrac{|x-b|}{c}\right)
\end{aligned}$$

8.9 References

A good introduction to optimization theory and a description of the the modern minimization techniques can be found in Gill, Murray and [1]. [2] deals with the particular problem of parameter estimation; chapters concerning the different estimators and their properties, and the interpretation of the estimates are remarkable. Updating formulas for Quasi–Newton methods are discussed in [4].

The reading of the cited chapters of [2] will allow an error–free interpretation of the results of the optimization algorithms. It is therefore recommended.

Bibliography

[1] Gill P.E., Murray W., and Wright M.H., 1981: *Practical Optimization*, Academic Press. London

[2] Bard Y., 1974: *Non–linear Parameter Estimation*, Academic Press, London.

[3] Gill P.E., Murray W., 1978: *Algorithms for the solution of non-linear least squares problems*, SIAM J. of Num. An., vol 15, pp 977-992

[4] Gill P.E., Murray W., and Pitfield, 1972: *The implementation of two revised algorithms for unconstrained optimization*, Rep. NAC 11. Nat. Phys., Lab., Teddington. England.

Appendix A

Command Summary

Below is an alphabetical list of all basic MIDAS commands. The following abbreviations have been used where appropriate for the command parameters:

input image:	*infr infr_1, infr_2...*
output image:	*outfr outfr_1, outfr_2...*
input tables:	*intab intab_1, intab_2...*
output tables:	*outtab outtab_1, outtab_2...*
display channel:	*chan chan_1*
device:	*device*

A.1 Core Commands

`@@ proc` *[par1] ... [par8]*
execute a MIDAS procedure
`@ proc` *[par1] ... [par8]*
execute a procedure in MID_PROC (MIDAS core procedures)
`@a proc` *[par1] ... [par8]*
execute a procedure in APP_PROC (MIDAS application procedures)
`@s proc` *[par1] ... [par8]*
execute a procedure in STD_PROC (MIDAS standard reduction procedure)

`ADD/ACAT` *[cat_name] frame_list*
add entries to an ASCII file catalog
`ADD/FCAT` *[cat_name] file_list [lowstr,histr]*
add entries to a fitfile catalog
`ADD/ICAT` *[cat_name] frame_list [lowstr,histr]*
add entries to an image catalog
`ADD/TCAT` *[cat_name] table_list [lowstr,histr]*

add entries to a table catalog

`ALIGN/CENTER` *inframe refframe incent_x,_y refcent_x,_y*
compute start coordinates for inframe to match with refframe center

`ALIGN/IMAGE` *intab reftab [option] [overlay_flag] [residual_flag]*
compute transformation coefficients for two rotated images

`APPLY/CONVERSION` *IMTB ima tab threshold*
convert a "mask" image to a table

`APPLY/CONVERSION` *TBIM tab ima npx1,npx2 sta1,sta2,stp1,stp2 bg,fg*
convert a table to a "mask" image

`APPLY/EDGE` *inframe outframe [thresh]*
do edge detection on an image

`APPLY/MAP` *outframe = inframe mapframe control_flags*
use an image frame like a Lookup Table

`APPLY/THIN` *inframe outframe*
apply thinning algorithm

`ASSIGN/DEFAULT`
assign default devices

`ASSIGN/DISPLAY` *[dev] [file_name]*
define output device for display

`ASSIGN/GRAPHICS` *[device] [option]*
define the graphic device for plot output

`ASSIGN/INPUT` *[dev] [file_name]*
define input device for writing

`ASSIGN/PRINT` *[dev] [file_name]*
define output device for printing

`AVERAGE/AVERAGE` *[in_specs] [out_specs] [out_opt] [draw_flag]*
compute average over subimage

`AVERAGE/COLUMN` *out = in [start,end] [SUM]*
average image columns

`AVERAGE/IMAGES` *out = in_specs [merge] [null] [av_option] [dat_intval]*
average images

`AVERAGE/KAPPA` *[in_specs] [out_specs] [out_opt] [draw_flag] [no_iter]*
compute average (kappa-sigma clipping) over subimage

`AVERAGE/MEDIAN` *[in_specs] [out_specs] [out_opt] [draw_flag]*
compute average (median value) over subimage

`AVERAGE/ROW` *out = in [start,end] [SUM]*
average image rows

`AVERAGE/WEIGHTS` *out = in_specs [merge] [null] [av_option] [dat_intval]*
average weighted images

`AVERAGE/WINDOW` *out = in_specs [meth] [bgerr,snoise]*
compute average of (consistent) pixel values

`BLINK/CHANNEL` *[cha1,cha2,..] [intval] [area_option]* blink between Image Display channels

`BYE` *[proc]*
terminate a MIDAS *session + return to the host system*

`CENTER/GAUSS` *[in_specs] [out_specs] [out_opt] [curs_specs] [wsize] [zw_option] [invert_flag]*
find intensity weighted center

`CENTER/IQE` *[in_specs] [out_specs] [out_opt] [curs_specs] [wsize] [zw_option] [invert_flag]*
find intensity weighted center + get angle of major axis

`CENTER/MOMENT` *[in_specs] [out_specs] [out_opt] [curs_specs] [wsize] [zw_option] [invert_flag]*
find intensity weighted center

`CHANGE/DIRECTORY` *direc*
change the default (current) directory for MIDAS

`CLEAR/ACAT`
deactivate the ASCII file catalog

`CLEAR/ALPHA`
clear the alpha-numerics memory of the image display

`CLEAR/BACKGROUND` *[no_secs]*
put Midas session into "foreground" mode

`CLEAR/BUFFER`
clear the command buffer + reset command numbers

`CLEAR/CHANNEL` *[chanl]*
clear + initialize memory channel

`CLEAR/CONTEXT` *[context]*
remove command definitions of a context

`CLEAR/DISPLAY`
reset image display

`CLEAR/FCAT`
disable automatic catalog functions for fit files

`CLEAR/GRAPHIC`
erase the screen of the graphic window or terminal

`CLEAR/ICAT`
disable automatic catalog functions for image frames

`CLEAR/ITT` *[chanl]*
bypass ITT on display of memory

`CLEAR/LUT`
bypass LUT *on image display*

`CLEAR/OVERLAY`
disable graphics/overlay plane of display

`CLEAR/SCROLL` *[chanl]*
reset scroll values

`CLEAR/TCAT`
: disable automatic catalog functions for table files

`CLEAR/ZOOM` *[chanl]*
: clear zoom

`CLOSE/FILE` *file_id*
: close an ASCII *file*

`COMPUTE/AIRMASS` *frame [long] [lat]*
`COMPUTE/AIRMASS` *alpha delta ST [exptime] [long] [lat] [date] [UT]*
: compute airmass (from sec z)

`COMPUTE/BARYCORR` *date UT alpha delta [longitude] [latitude]*
`COMPUTE/BARYCORR` *table.tbl [longitude] [latitude]*
`COMPUTE/BARYCORR` *image alpha delta [longitude] [latitude]*
: correct universal times and radial velocities to center of sun or barycenter of solar system

`COMPUTE/COLUMN` *res_frame.column = arithmetic_expression*
: do arithmetics on columns of an image

`COMPUTE/DIMAGE` *[outspec =] expression*
: compute arithmetic expression

`COMPUTE/DPIXEL` *[outspec =] expression*
: compute expression on pixel basis

`COMPUTE/DPLANE` *res_frame.plane = arithmetic_expression*
: do arithmetics on planes of images

`COMPUTE/HISTOGRAM` *result = table col [bin [min [max]]]*
: table-to-image or table-to-table histogram transformation

`COMPUTE/IMAGE` *[outspec =] expression*
: compute arithmetic expression

`COMPUTE/KEYWORD` *key = arithmetic_expression*
: compute values of a keyword

`COMPUTE/PIXEL` *[outspec =] expression*
: compute expression on pixel basis

`COMPUTE/PLANE` *res_frame.plane = arithmetic_expression*
: do arithmetics on planes of images

`COMPUTE/PRECESSION` *alpha delta equinox0 equinox1*
`COMPUTE/PRECESSION` *table.tbl equinox0 equinox1*
: precess equatorial coordinates from one epoch to another

`COMPUTE/REGRESSION` *table column = name[(ind)] [d_type]*
: compute result of a regression

`COMPUTE/ROW` *res_frame.row = arithmetic_expression*
: do arithmetics on rows (lines) of images

`COMPUTE/ST` *date UT [longitude]*
`COMPUTE/ST` *table.tbl [longitude]*
`COMPUTE/ST` *image [longitude]*
: calculate geocentric Julian date (JD) and local mean sidereal time (ST) from civil date and universal time (UT)

`COMPUTE/TABLE` *table column = expression*
compute arithmetic or string operations on table columns
`COMPUTE/UT` *date ST [longitude]*
`COMPUTE/UT` *table.tbl [longitude]*
`COMPUTE/UT` *image [longitude]*
calculate geocentric Julian date (JD) and universal time (UT) from civil date and local mean sidereal time (ST)
`COMPUTE/WEIGHTS` *input_specs [window_specs]*
determine weights for command AVERAGE/WEIGHTS
`COMPUTE/XYPLANE` *result_cube = expression*
compute arithmetic expression on xy_planes of cubes
`COMPUTE/XZPLANE` *result_cube = expression*
compute arithmetic expression on xz_planes of cubes
`COMPUTE/ZYPLANE` *result_cube = expression*
compute arithmetic expression on zy_planes of cubes
`CONNECT/BACK_MIDAS` *unit wait_specs b_char method*
connect "command syntax" to another MIDAS
`CONVERT/COORDS` *image coord_string [dsplay_flag]*
frame pixel ro world coordinates conversion (and vice versa)
`CONVERT/TABLE` *image = table x[,y] z refima [method] [par]*
`CONVERT/TABLE` *image = table x[,y] refima FREQ*
table to image conversion
`CONVOLVE/IMAGE` *frame psf result*
convolve image with point spread function
`COPY/AD` *ASCII_file dest_frame dest_desc [max_len]*
copy contents of ASCII *file to character descriptor of dest_frame*
`COPY/DA` *source_frame source_desc ASCII_file*
copy descriptor of source frame to ASCII *file*
`COPY/DD` *source_frame source_desc dest_frame dest_desc*
copy descriptors of source frame to destination frame
`COPY/DIMA` *source_frame source_desc dest_frame*
copy descriptor of source frame to new image
`COPY/DISPLAY` *[out_dev] [stop_flg] [ITTdef] [LUTnam] [prflag] [prmode]*
make a hardcopy of the display on output_device
`COPY/DK` *source_frame source_desc dest_key*
copy descriptor of source frame to keyword
`COPY/DKEY` *source_frame source_desc dest_key*
copy descriptor of source frame to keyword
`COPY/GRAPHICS` *[device] [plotfile]*
copy the existing plot file to the graphic device
`COPY/ID` *source_frame dest_frame dest_desc*
copy image data to descriptor of destination frame
`COPY/II` *source_frame dest_frame dest_format flags*
copy source frame to destination frame

`COPY/IT` *inframe outable [column]*
copy image into table
`COPY/KD` *source_key dest_frame dest_desc*
copy keyword to descriptor of destination frame
`COPY/KEYWORD` *source_key dest_key [M_unit]*
copy keywords of same type
`COPY/KI` *source_key dest_frame*
copy keyword to new frame
`COPY/KT` *keyword table [column ...] element*
copy keyword into table element
`COPY/LSDD` *list source_frame dest_frame*
copy list of descriptors of source frame to descriptors of dest_frame
`COPY/LSDK` *list source_frame*
copy list of descriptors of source frame to keywords
`COPY/LSKD` *list dest_frame*
copy list of keywords to descriptors of destination frame
`COPY/OVERLAY` *[chanl] [append_flag]*
copy all graphics and text from the overlay plane to image channel
`COPY/PSCR` *image [PS_file] [options] [prnt_flags] [scales] [LUT_specs]*
convert MIDAS *images (bdf-files) to PostScript format*
`COPY/TABLE` *intable outable [organization]*
copy source table to destination table
`COPY/TEIMA` *table column rows [npix] [coor] [prefix]*
Convert a 3D-table element to a MIDAS *image file*
`COPY/TI` *intable outimage*
copy table into image
`COPY/TK` *table [column ...] element keyword*
copy table element into keyword
`COPY/TT` *intable incolumn [outable] outcolumn*
copy a table column to an other existing table
`COPY/ZOOM` *[out_dev] [stop_flg] [ITTnam] [LUTnam] [prflag] [prmode]*
make a copy of the zoom window on output device
`CREATE/ACAT` *[cat_name] [dir_spec]*
create a catalog of files in the current directory
`CREATE/COLUMN` *table column [unit] [format] [type]*
create a table column
`CREATE/COMMAND` *comnd text*
create a "user" command
`CREATE/CURSOR` *[dspid] [wind_specs] [Xstation]*
create a cursor window
`CREATE/D_COMMAND` *comnd text*
create a directory "user" command
`CREATE/DEFAULT` *comnd def1 def2 ... def8*
create special defaults for MIDAS command

`CREATE/DISPLAY` *[dspid] [dspinfo] [meminfo] [alph_flag] [gsize] [Xstation]*
  create a display window
`CREATE/FCAT` *[catname] [dir_spec]*
  create a catalog of fit files in the current directory
`CREATE/FILTER` *frame [dim_specs] [frame_specs] [filt_type] [coefs]*
  create filter frame
`CREATE/GRAPHICS` *[graph_id] [graph_spec] [gsize] [Xstation]*
  create a graphics window
`CREATE/ICAT` *[catname] [dir_spec] [descr]*
  create a catalog of images in the current directory
`CREATE/IMAGE` *frame [dim_specs] [frame_specs] [func_type] [coefs]*
  create an image
`CREATE/LUT` *LUT_table H_specs S_specs I_specs cyclic_flag*
  create a colour lookup table
`CREATE/RANDOM` *name [dims] [starts,steps] [func_type] [coefs] [seed]*
`CREATE/RANDOM` *name = ref_frame [func_type] [coefs] [seed]*
  create a random image
`CREATE/ROW` *table row_position number_of_rows*
  add one or several rows at a given position of a table
`CREATE/TABLE` *table ncol nrow file [format_file] [organization]*
  create a table
`CREATE/TCAT` *[catname] [dir_spec] [descr]*
  create a catalog of tables in the current directory
`CREATE/VIRTUAL` *virtual table*
  create a virtual table from a physical table
`CREATE/ZOOM` *[dspid] [wind_specs] [Xstation]*
  create a zoom window
`CUTS/IMAGE` *frame [cut_specs]*
  display or set low + high cut values of an image frame

`DEBUG/MODULE` *[low_lev,hi_lev] [switch]*
  run MIDAS modules in debug mode
`DEBUG/PROCEDURE` *[low_lev,hi_lev] [switch]*
  run MIDAS procedures in debug mode
`DECONVOLVE/IMAGE` *frame psf result [no_iter] [cont_flag]*
  deconvolve image with point spread function
`DELETE/ACAT` *[catalog] [conf_flag] [range]*
  delete files with entry in ASCII file catalog
`DELETE/COLUMN` *table column_sel*
  delete table column(s)
`DELETE/COMMAND` *[comnd]*
  delete user defined command
`DELETE/CURSOR` *[disp]*

delete cursor window(s) on XWindow displays

`DELETE/DEFAULTS` *[comnd]*
delete special defaults for command

`DELETE/DESCRIPTOR` *frame descr_list [stop_flag]*
delete descriptors of frame

`DELETE/DISPLAY` *[disp]*
delete display window(s) on XWindow displays

`DELETE/FCAT` *[catalog] [conf_flag] [range]*
delete fit files with entry in catalog

`DELETE/FIT` *name [conf_flag]*
delete a fit file

`DELETE/GRAPHICS` *[grap]*
delete graphic window(s) on XWindow displays

`DELETE/ICAT` *[catalog] [conf_flag] [range]*
delete image frames with entry in catalog

`DELETE/IMAGE` *name [conf_flag]*
delete an image frame

`DELETE/KEYWORD` *key*
delete user defined keyword

`DELETE/LOGFILE`
delete current logfile

`DELETE/ROW` *table row_position number_of_rows*
delete one or several rows of a table

`DELETE/TABLE` *name [conf_flag]*
delete a table file

`DELETE/TCAT` *[catalog] [conf_flag] [range]*
delete table files with entry in catalog

`DELETE/TEMP`
delete temporary MIDAS files

`DELETE/ZOOM` *[disp]*
delete zoom window(s) on XWindow displays

`DISCONNECT/BACK_MIDAS` *unit [server_flag]*
disconnect from a background MIDAS

`DISPLAY/CHANNEL` *[chanl]*
display contents loaded in an Image Display channel

`DISPLAY/LUT` *[switch]*
en/disable display of current LUT

`DRAW/ANY` *[intens]*
draw manually or with input from file into the overlay channel

`DRAW/ARROW` *[in_spec] [coord_ref] [draw_opt] [intens] [nocurs] [key_flag]*
draw arrows in the overlay channel

`DRAW/CIRCLE` *[in_spec] [coord_ref] [draw_opt] [draw_specs] [nocurs] [key_flag] [arc_defs]*
draw circles in the overlay channel

`DRAW/CROSS` *[in_spec] [coord_ref] [draw_opt] [intens] [nocurs] [key_flag]*
draw crosses in the overlay channel
`DRAW/ELLIPS` *[in_spec] [coord_ref] [draw_opt] [draw_specs] [nocurs] [key_flag] [arc_defs]*
draw ellipses in the overlay channel
`DRAW/IMAGE` *frame [chanl] [scale] [center] [cuts] [over] [iaux] [fix]*
draw intensities of a line of an image into display channel
`DRAW/LINE` *[in_spec] [coord_ref] [draw_opt] [intens] [nocurs] [key_flag]*
draw straight line in the overlay channel
`DRAW/RECTANGLE` *[in_spec] [coord_ref] [draw_opt] [intens] [nocurs] [key_flag]*
draw rectangles in the overlay channel
`DRAW/SLIT` *[in_spec] [coord_ref] [draw_opt] [intens] [nocurs] [key_flag]*
draw IUE slits in the overlay channel

`ECHO/FULL` *[levla,levlb]*
show substitutions in MIDAS *procedure files*
`ECHO/OFF` *[levla,levlb]*
suppress display of input from MIDAS procedure files
`ECHO/ON` *[levla,levlb]*
display input from MIDAS procedure files
`EDIT/TABLE` *table [edit_option] [col] [row]*
interactive table editor
`EQUALIZE/HISTOGRAM` *frame descr itt_name*
perform histogram equalization
`EXECUTE/CATALOG` *com_string parm1 ... parm7*
execute a MIDAS *procedure or command for all entries in a catalog*
`EXECUTE/TABLE` *table command-string*
execute command on all rows of table
`EXTRACT/CTRACE` *[step] [frame] [plot_flag] [zw_option]*
extract a column from displayed image
`EXTRACT/CURSOR` *[subfr] [xpx,ypx] [loop_flag]*
extract a subframe via cursor
`EXTRACT/IMAGE` *out = in intval*
`EXTRACT/IMAGE` *out = in center_spec loff roff*
extract a subimage from an image frame
`EXTRACT/LINE` *out = in[..] step*
extract a 1-dim line from a 2-dim frame
`EXTRACT/REFERENCE_IMAGE` *in ref out thresh*
extract subimage according to reference image
`EXTRACT/ROTATED_IMAGE` *steps frame*
extract a rotated subimage from displayed image
`EXTRACT/RTRACE` *[step] [frame] [plot_flag] [zw_option]*
extract a row from displayed image

`EXTRACT/SLIT` *[in_option] [resframe] [slit_specs]*
    extract a subimage defined by a fixed slit from image
`EXTRACT/TRACE` *[step] [frame] [plot_flag] [cut_option] [zw_option]*
    extract a line from displayed image

`FFT/FINVERSE` *inr ini outr outi*
    make inverse discrete Fourier transform
`FFT/FPOWER` *inr ini outr outi pow_spec*
    make discrete Fourier transform and power spectrum
`FFT/FREQUENCY` *inr ini outr outi*
    make discrete Fourier transform with frequency scaling
`FFT/IMAGE` *inr ini outr outi*
    make discrete Fourier transform
`FFT/INVERSE` *inr ini outr outi*
    make inverse discrete Fourier transform
`FFT/POWER` *inr ini outr outi pow_spec*
    make discrete Fourier transform and power spectrum
`FILTER/COSMIC` *inframe outframe sky,gain,ron,[ns],[rc] [mask]*
    remove cosmic ray events.
`FILTER/DIGITAL` *frame outframe [filter_specs] [subimage] [options]*
    use digital filter on an image
`FILTER/GAUSS` *in out [radx,rady] [gauss_specs] [subima] [filtnam] [options]*
    use Gaussian filter on an image
`FILTER/MAX` *frame outfram [xyradius] [subima] [options]*
    apply maximum filter to an image
`FILTER/MEDIAN` *frame outfram [filt_specs] [flag] [subima] [options]*
    smooth an image with median filter
`FILTER/MIN` *frame outfram [xyradius] [subima] [options]*
    apply minimum filter to an image
`FILTER/SMOOTH` *frame outfram [filter_specs] [flag] [subima] [options]*
    smooth an image by averaging
`FIND/MINMAX` *frame*
    find min, max of frame and corresponding pixel numbers
`FIND/PIXEL` *frame lo,hi [inout_flag] [first_flag] [out_spec] [maxnpix] [area]*
    find first/all pixel(s) with a value in/outside interval [low,high]
`FIT/FLAT_SKY` *outframe = inframe [in_specs] [order] [back_surface]*
`FIT/FLAT_SKY` *inframe [in_specs] [order] [back_surface]*
    Approximate background of image by a surface
`FLIP/IMAGE` *frame [flag]*
    flip an image around an axis

`GET/CURSOR` *[output] [option] [marker] [curs_specs] [zw_option]*
    read cursor coords from display

`GET/GCURSOR` *[output_spec] [app_flag] [max]*
    read and store cursor coordinates from the graphics display
`GET/IMAGE` *frame [input_source] [ITT_flag]*
    read image from displayed image channel
`GET/ITT` *out_specs [chanl] [sect]*
    read currently active ITT from image display
`GET/LUT` *out_specs [get_specs] [ITT] [format] [range]*
    read currently active LUT from image display
`GROW/CUBE` *frame no_planes frame_list*
    expand 2-dim/3-dim frame
`GROW/IMAGE` *out = in [start,step,no] [lincol_specs] [lincol_flag]*
    expand single line into 2-dim image

`HELP` *[help_topic]*
    display info about help_topic
`HELP/APPLIC` *[proc]*
    display header information of application procedure
`HELP/CL` *[command]*
    display help for commands only used in MIDAS procedures
`HELP/CONTRIB` *[proc]*
    display header information of procedures in the Midas 'contrib' area
`HELP/KEYWORD` *key*
    explain contents of given key
`HELP/QUALIF` *[qualif]*
    display all commands with given qualifier
`HELP/SUBJECT` *[topic]*
    display information related to given topic

`INDISK/ASCII` *in_file [out_file] [npix_string]*
    read ASCII *file from disk + convert to Midas image*
`INDISK/FITS` *in_files [out_spec] [option] [check_flag]*
    read FITS *files from disk*
`INFO/DESCR` *frame descr*
    get type and size of descriptor
`INFO/IMAGE` *frame [option]*
    get internal info of frame
`INFO/KEYWORD` *keyword*
    get type, size, lockno and mod-time of keyword
`INFO/SETUP` *[setup]*
    display all the information about a Setup
`INITIALIZE/DISPLAY` *[noLUT,maxLUTsz,minLUTsz] [ownLUT] [M_unit] [fonts] [colmode]*

initialize the image display

`INITIALIZE/SETUP` *[setup]*
initialize the variables of a Setup

`INSERT/IMAGE` *subframe modframe [startx,y,z]*
insert a subframe into another frame

`INTAPE/FITS` *file_specs file-id device [flags]*
read frames from magtape in FITS/IHAP format

`INTERPOLATE/II` *outima inima refima [s] [degree]*
interpolate Image to Image

`INTERPOLATE/IT` *outtab i,d inima [s] [degree]*
interpolate Image to Table

`INTERPOLATE/TI` *outima intab i,d refima [s] [degree]*
interpolate Image to Image

`INTERPOLATE/TT` *outtab i,d intab i,d [s] [degree]*
interpolate Table to Table

`ITF/IMAGE` *inframe table coli,colo scal outframe*
ITF correction

`JOIN/TABLE` *intab1 :X1,[:Y1] intab2 :X2,[:Y2] outtable [tolX,tolY]*
join table files

`LABEL/ANY` *filename*
write text labels stored in a file on the image display

`LABEL/DISPLAY` *labl [position] [mode] [option] [size] [key_flag]*
write a label on the image display

`LABEL/GRAPHIC` *label [x_pos,y_pos[,mm]] [angle] [size] [pos_ind]*
write a label_string on the graphics device

`LOAD/IMAGE` *frame_spec [chanl] [scale] [center] [cuts] [dirs] [fix]*
load image into display device

`LOAD/ITT` *in_specs [chanl]*
load intensity transfer table to Image Display

`LOAD/LUT` *in_specs [load_specs] [disp_flag] [format]*
load colour lookup table into Image Display

`LOAD/TABLE` *table x y [ident] [symbol] [size] [color] [conn_flag]*
load table into overlay channel of Image Display

`LOCK/KEYWORD` *key_list lockno*
lock keyword(s)

`LOG/OFF`
suppress logging

`LOG/ON`
enable logging

`LOG/TOF`

write top_of_form into logfile

`MAGNITUDE/CENTER` *[in_specs] [out_specs] [Fsiz,Nsiz,Bsiz] [out_opt] [center_params] [curs_specs] [zw_option]*
compute magnitude in center
`MAGNITUDE/CIRCLE` *[in_specs] [out_specs] [Fsiz,Nsiz,Bsiz] [out_opt] [center_params] [curs_specs] [zw_option]*
compute magnitude within circular aperture
`MAGNITUDE/RECTANGLE` *[in_specs] [out_specs] [pix_area] [out_opt] [center_params] [curs_specs] [zw_option]*
compute magnitude within square aperture
`MERGE/TABLE` *intable [intable ...] outtable*
merge table files
`MODIFY/AREA` *[source] [resfram] [degree] [constant] [drawflg]*
remove bad data from a circular pixel-area in an image
`MODIFY/COLUMN` *source_def res_frame [col_type] [column_coords]*
approximate values in a column
`MODIFY/CUTS` *[image] [cursor_spec]*
modify cut values of full frame or in cursor selected windows
`MODIFY/DISPLAY` *[target]*
toggle the state of a display window (mapped Window ¡-¿ Icon)
`MODIFY/GCURSOR` *frm_in frm_out y-coord xstart,xend no_curs,degree*
interactive modification of pixel values in a frame
`MODIFY/GRAPHICS` *[target]*
toggle the state of a graphics window (mapped Window ¡-¿ Icon)
`MODIFY/ITT` *[method] [value] [prflag]*
modify the currently active ITT
`MODIFY/LUT` *[method] [colour] [prflag]*
modify the currently active LUT
`MODIFY/PIXELS` *[source] [resfram] [arfacts] [xdeg,ydeg,niter] [drawflg] [noise] [ima_sec]*
approximate pixel-area in an image
`MODIFY/ROW` *source_def res_frame [row_type] [row_coords]*
approximate values in a row

`NAME/COLUMN` *table column [new-column] [unit] [format]*
redefines label/unit/format of a column

`OPEN/FILE` *filename flag file_control_key*
open an ASCII *file for reading or writing*
`OUTDISK/FITS` *in_files [out_spec] [option] [out_type]*
write Midas frames to FITS *files on disk*

`OUTDISK/SFITS` *in1[,in2[,...]] [out] [flags]*
: convert set of MIDAS *files to a single FITS file on disk*

`OUTTAPE/FITS` *[catalog[,list]] device [flags] [density,block] [type]*
: write to device in FITS *format*

`OVERPLOT/AXES` *[x_axis_spec] [y_axis_spec] [x_sc,y_sci[,x_off,y_off]] [x_lab] [y_lab]*
: overplot a coordinate box with tickmarks and labels

`OVERPLOT/AXES` *[coord_str] [x_lab] [y_lab]*
: overplot a coordinate box around a displayed frame

`OVERPLOT/COLUMN` *frame [x-coord] [y_start,y_end] [offset] [l_type]*
: overplot a column of a frame on a graphic device

`OVERPLOT/COLUMN` *frame [x-coord] [y_start,y_end] [offset] [l_type]*
: overplot a column of a frame on a graphic device

`OVERPLOT/CONTOUR` *frame [coord_str] [contours] [sm_par]*
: overplot contour map of 2-dim. frame with smoothing option

`OVERPLOT/DESCRIPTOR` *frame [descr] [start,end] [offset]*
: overplot the contents of a descriptor

`OVERPLOT/ERROR` *table [col1] [col2] col3 [direct] [bar]*
: overplot table error column

`OVERPLOT/GRAY` *frame [coord_str] [gray_lev] [sm_par] [gray_ness] [options]*
: overplot gray scale map of 2-dim. frame with smoothing option

`OVERPLOT/GRID` *grid*
: overplot a grid on an existing coordinate box

`OVERPLOT/HISTOGRAM` *tab col [offset] [bin[,min[,max]]] [exc] [log] [opt]*
: overplot histogram of a column in the table

`OVERPLOT/HISTOGRAM` *frame [offset] [log] [opt]*
: overplot the histogram of an image

`OVERPLOT/KEYWORD` *[key_name] [start,end] [offset]*
: overplot the contents of a keyword

`OVERPLOT/LINE` *[l_type] [x_sta,y_sta [x_end,y_end]]*
: overplot a line on a graphic device

`OVERPLOT/ROW` *frame [y-coord] [x_start,x_end] [offset] [l_type]*
: overplot a row (line) of a frame on a graphic device

`OVERPLOT/SYMBOL` *[s_type] [x_coord,y_coord] [s_size]*
: overplot a symbol

`OVERPLOT/TABLE` *table [plane1] [plane2] [x_sc,y_sc[,x_off,y_off]] [symbols] [lines] [flag_dir]*
: plot table data on selected plotting device

`OVERPLOT/VECTOR` *fram_a fram_b [coord_str] [scale_r] [pos_range] [sm_par] [head]*
: overplot vector map from two 2-dim. images with smoothing option

`PLAYBACK/FILE` *name*
: playback MIDAS *commands from an ASCII file*

`PLAYBACK/LOGFILE` *file*
　　playback MIDAS *commands from a previous logfile*
`PLOT/AXES` *[x_axis_spec] [y_axis_spec] [x_sc,y_sc[,x_off,y_off]] [x_lab] [y_lab]*
　　plot a coordinate box with large and small tickmarks and labels
`PLOT/AXES` *[coord_str] [x_lab] [y_lab]*
　　plot a coordinate box around a displayed frame
`PLOT/COLUMN` *frame [x_coord] [y_sta,y_end] [x_sc,y_sc,x_off,y_off]*
　　plot a column of an image on a plotting device
`PLOT/CONTOUR` *frame [coord_str] [x_sc,y_sc[,x_off,y_off]] [contours] [c_type] [sm_par]*
　　plot contour map of 2-dim. image with smoothing option
`PLOT/DESCRIPTOR` *frame [descr] [start,end] [x_sc,y_sc[,x_off,y_off]]*
　　plot a descriptor on plotting device
`PLOT/GRAY` *frame [coord_str] [x_sc,y_sc[,x_off,y_off]] [gray_lev] [sm_par] [gray_ness] [gray_opt]*
　　plot gray scale map of 2-dim. image with smoothing option
`PLOT/HISTOGRAM` *tab col [x_sc,y_sc[,x_off,y_off]] [bin[,min[,max]]] [exc] [log] [opt]*
　　plot histogram of a column in the table
`PLOT/HISTOGRAM` *frame [x_sc,y_sc[,x_off,y_off]] [exs] [log] [opt]*
　　plot the histogram of an image
`PLOT/KEYWORD` *[key_name] [start,end] [x_sc,y_sc[,x_off,y_off]]*
　　plot the contents of a keyword
`PLOT/PERSPECTIVE` *frame [coord_str] [alt,azi] [scal,offs] [sm_par] [xy_flag]*
　　tree dim. representation of a 2-dim. frame, with smoothing option
`PLOT/ROW` *frame [y_coord] [x_sta,x_end] [x_sc,y_sc[,x_off,y_off]]*
　　plot a row (line) of an image on a plotting device
`PLOT/TABLE` *table [plane1] [plane2] [x_sc,y_sc[,x_off,y_off]] [symbols] [lines] [flag_dir]*
　　plot table data on selected plotting device
`PLOT/VECTOR` *frame_a frame_b [coord_str] [x_sc,y_sc[,x_off,y_off]] [scale_r] [range] [sm_par] [head]*
　　plot vector map from two 2-dim. images with smoothing option
`PRINT/ACAT` *[cat_name] [lowno,hino]*
　　print ASCII file catalog entries
`PRINT/DESCR` *frame [descr_list] [disp_flag]*
　　print descriptor values
`PRINT/FCAT` *[cat_name] [lowno,hino]*
　　print fit file catalog entries
`PRINT/HELP` *[help_topic]*
　　print info about help_topic
`PRINT/HISTOGRAM` *table column [bin [min [max]]]*
　　print statistics of a column
`PRINT/ICAT` *[cat_name] [lowno,hino]*

print image catalog entries

`PRINT/IMAGE` *frame_specs [pixel_specs] [hide_header_flag]*

print image data values

`PRINT/KEYWORD` *[key_list] [since]*

print contents of keywords

`PRINT/LOGFILE` *[page_specs]*

print contents of logfile

`PRINT/TABLE` *table [column ...] [elem1 [elem2]] [N] [width]*

`PRINT/TABLE` *table [elem1 [elem2]] [form] [N]*

print table values on the device/file specified via ASSIGN/PRINT

`PRINT/TCAT` *[cat_name] [lowno,hino]*

print table catalog entries

`PROJECTION/TABLE` *intable outable column_selection*

projection of one or more columns from a table

`READ/ACAT` *[cat_name] [lowno,hino]*

read ASCII *Catalog entries*

`READ/COMMANDS` *[proc]*

read commands from a procedure + store into command buffer

`READ/DESCR` *frame [descr_list] [disp_flag]*

display descriptor values

`READ/FCAT` *[cat_name] [lowno,hino]*

read fit file catalog entries

`READ/FILE` *file_id cbuf_key [maxrd]*

read an ASCII *file*

`READ/HISTOGRAM` *table column [bin [min [max]]]*

display statistics of table column

`READ/ICAT` *[cat_name] [lowno,hino]*

read Image Catalog entries

`READ/IMAGE` *frame_specs [pixel_specs] [hide_header_flag]*

display image data values

`READ/KEYWORD` *[key_list] [disp_flag] [since] [Midunit]*

display contents of keywords

`READ/SETUP` *setup*

read the contents of the variables related to a Setup

`READ/TABLE` *table [column_sel] [row_sel] [form]*

display table elements

`READ/TCAT` *[cat_name] [lowno,hino]*

read Table Catalog entries

`REBIN/II` *outima inima refima [func] [param] [intop]*

nonlinear rebin Image to Image

`REBIN/IT` *outtab i,d[,b] inima [func] [param] [intop]*

nonlinear rebin Image to Table

REBIN/LINEAR *in out [stepx,stepy] [offx,offy] [stx,sty] [fluxcons] [proc_mode]*
REBIN/LINEAR *in out [refframe] [fluxcons] [proc_mode]*
    rebin an image linearly
REBIN/ROTATE *in out [rot_specs] [ref_frame] [ref_flag]*
    rotate + rebin an image
REBIN/SPLINE *in out [stepx,stepy] [offx,offy] [startx,starty]*
REBIN/SPLINE *in out [refframe]*
    rebin an image using cubic splines
REBIN/STATISTICS *in out [xpix,ypix] [stat_val]*
    rebin image using statistical values as new pixel values
REBIN/TI *outima intab i,d[,b] refima [func] [param] [intop]*
    nonlinear rebin Table to Image
REBIN/TT *outtab i,d[,b] intab i,d[,b] [func] [param] [intop]*
    nonlinear rebin Table to Table
REGRESSION/LINEAR *table y x1,x2,...*
    linear regression on table columns
REGRESSION/POLY *table y x1[,x2] d1[,d2]*
    polynomial fit on table columns
REGRESSION/ROBUST *tab y x1[,x2,..,xn] [file] [out_col] [res_col]*
    Robust multi-variate regression by Least Median of Squares
REGRESSION/TABLE *table1 x1[,x2] table2 y1[,y2] degree tol [guess]*
    polynomial fit of variables in two tables (not yet implemented)
RENAME/FIT *old new [history] [overwrite]*
    rename a fit file
RENAME/IMAGE *old new [history] [overwrite]*
    rename an image frame
RENAME/TABLE *old new [history] [overwrite]*
    rename a table frame
REPLACE/IMAGE *in out [test/]low,hi=express1[,express2]*
    replace pixels according to intensity
REPLACE/POLYGON *in,intab out test/low,hi=value*
    replace pixels inside polygon
REPORT/PROBLEM *[errfile]*
    send error reports and comments to the person(s) in charge of MIDAS
RESAMPLE/IMAGE *in out stepping*
    resample an image
RESET/DISPLAY
    reset Xwindow display after Control C
RESTORE/NAME *[file_spec] [verbose] [history] [overwrite] [descr]*
    change file name according to descr. FILENAME
RETRO/TAB *table*
    retrofit 3-dim table to old 90NOV format
ROTATE/1DIM *in out nop_flag*
    rotate a 1-dim profile around its startpoint

`ROTATE/CLOCK` *in out [factor]*
    rotate an image by multiples of 90 degrees clockwise
`ROTATE/COUNTER_CLOCK` *in out [factor]*
    rotate an image by multiples of 90 degrees counter_clockwise
`RUN` *progr*
    execute a program inside the MIDAS environment

`SAVE/REGRESSION` *table name*
    save results of a regression
`SCROLL/CHANNEL` *[chanl] [scrolx,scroly]*
    scroll given ImageDisplay channel
`SEARCH/FCAT` *[cat_name] search_string [options]*
    search in fit file catalog for frame with matching descriptor IDENT
`SEARCH/ICAT` *[cat_name] search_string [options]*
    search in image catalog for frame with matching Ident field
`SEARCH/TCAT` *[cat_name] search_string [options]*
    search in table catalog for table with matching Ident field
`SELECT/TABLE` *table logical-expression*
    select table entries
`SET/ACAT` *[cat_name]*
    make given catalog the "active" ASCII file catalog
`SET/BACKGROUND` *[method] [echo] [sleep_time]*
    put Midas session into "background" mode
`SET/BUFFER` *[no_lines]*
    set up command buffer for MIDAS
`SET/CONTEXT` *cntxt*
    enable new context
`SET/CURSOR` *[curs_no] [curs_form] [curs_coords] [flag]*
    set cursor form and position
`SET/DISPLAY` *[colour_mode]*
    set up Image Display for RGB or pseudo colours
`SET/FCAT` *[cat_name]*
    make given catalog the "active" fit file catalog
`SET/FORMAT` *[format_specs]*
    set formats for replacement of Midas data
`SET/GCURSOR` *[curs_no] [curs_form]*
    set cursor form in graphics window
`SET/GRAPHICS` *option1[=value1] option2[=value2] ...*
    set plot characteristics
`SET/ICAT` *[cat_name]*
    make given catalog the "active" image frame catalog
`SET/ITT` *[chanl]*
    enable ITT *for Image display channel*

SET/LUT *[sect]*
  enable usage of colour lookup tables
SET/MIDAS_SYSTEM *option=value*
  set different modes and options for Midas
SET/OVERLAY
  enable graphics overlay
SET/REFCOLUMN *table column*
  define column as reference in table access
SET/TCAT *[cat_name]*
  make given catalog the "active" table file catalog
SHIFT/IMAGE *inframe outframe [x,yshift]*
  shift the pixels in an image
SHOW/ACAT *[cat_name] [display_flag]*
  show no. of entries in an ASCII file catalog
SHOW/BACK_MIDAS *unit [disp_option]*
  show info related to background MIDAS *sessions*
SHOW/CHANNEL *[chanl]*
  show info related to ImageDisplay channel
SHOW/CODE *command_string [option]*
  display the procedure which implements the command_string
SHOW/COMMAND *[comnd/qualif]*
  display MIDAS commands
SHOW/CONTEXT *[name]*
  show enabled contexts
SHOW/DEFAULTS
  display all special defaults
SHOW/DESCR *frame [descr_list] [flag]*
  show existing descriptors with name, type and size
SHOW/DISPLAY
  show current status of ImageDisplay + Graphics
SHOW/FCAT *[cat_name] [display_flag]*
  show no. of entries in a fit file catalog
SHOW/GRAPHICS *device_name*
  show the setup parameters for plotting
SHOW/ICAT *[cat_name] [display_flag]*
  show no. of entries in an image catalog
SHOW/KEYWORD *[keyword]*
  display contents of keyword data base
SHOW/TABLE *table*
  display table parameters
SHOW/TCAT *[cat_name] [display_flag]*
  show no. of entries in a table catalog
SHRINK/IMAGE *in_image out_image [option]*
  shrink 2 or 3-dim image to 1-dim image

SORT/FCAT *[cat_name]*
    sort a fit file catalog
SORT/ICAT *[cat_name]*
    sort an image catalog
SORT/TABLE *table keys*
    sort table according to (ascending) values
SORT/TCAT *[cat_name]*
    sort a table catalog
STATISTICS/IMAGE *[frame] [area] [bins] [lo,hi_exc] [option] [outtab] [plotflg] [format]*
    calculate statistics of a frame
STATISTICS/TABLE *table column*
    simple statistics on a table column
STORE/FRAME *key frame [indx] [exit_label]*
    store frame or entries of catalog into key
SUBTRACT/ACAT *[cat_name] frame_list*
    remove entries from an ASCII file catalog
SUBTRACT/FCAT *[cat_name] frame_list*
    remove entries from a fit file catalog
SUBTRACT/ICAT *[cat_name] frame_list*
    remove entries from an image catalog
SUBTRACT/TCAT *[cat_name] frame_list*
    remove entries from an table catalog
SYNCHRONIZE/MIDAS
    write keyfile and logfile to disk
SYNCHRONIZE/TIME *option*
    do time synchronization functions

TRANSLATE/SHOW *proc option*
    translate MIDAS *procedure and display resulting code*
TRANSPOSE/CUBE *inframe [outframe] [plane_spec]*
    rearrange the planes of a cube
TRANSPOSE/IMAGE *inima outima [diagonal]*
    transpose image
TUTORIAL/EXTRACT
    demonstrate some of the different EXTRACT commands
TUTORIAL/FILTER
    explain the usage of filters
TUTORIAL/GRAPHICS *option*
    explain the use of the graphics packages
TUTORIAL/HELP
    explain usage of the HELP command
TUTORIAL/ITT *[plotflag]*

explain the usage of ITT's
`TUTORIAL/LUT` *[plotflag]*
show some standard LUT's and related MIDAS commands
`TUTORIAL/SPLIT`
explain the usage of "split screen", i.e. multiple windows
`TUTORIAL/TABLE`
explain usage of tables

`UNLOCK/KEYWORD` *key_list lockno*
unlock keyword(s)

`$ comnd`
execute a host system command
`VIEW/IMAGE` *[frame] [out_tab] [plot_option] [g,zhardcopy]*
view an image with a "looking glass"

`WAIT/BACK_MIDAS` *[unit]*
wait until command in background MIDAS *terminates*
`WAIT/SECS` *[no_of_secs]*
suspend MIDAS *monitor for no_of_secs seconds*
`WRITE/COMMANDS` *[procnam] [par1] [par2] ... [par8]*
save commands from command buffer + write into a procedure
`WRITE/DESCR` *frame descr data [flg]*
store values into a descriptor
`WRITE/DHELP` *frame descr text*
store help-text/comments for an existing descriptor
`WRITE/FILE` *file_id charbuf*
write into an ASCII *file*
`WRITE/IMAGE` *frame [pix_specs] data [all_flg] [process_flg]*
`WRITE/IMAGE` *CURSOR,option data [process_flg]*
update values of image frame
`WRITE/KEYWORD` *key data [flg]*
write values into a keyword
`WRITE/OUT` *text_spec [section] [label]*
display text on terminal
`WRITE/SETUP` *[setup]*
modify the variables of a Setup
`WRITE/TABLE` *table column row_sel value*
Store a value into a table

`XCORRELATE/IMAGE` *temp spec result shift* correlate 2 similar 1-dim frames over 2*(shift) bandwith

`ZOOM/CHANNEL` *[zoom_fact] [center]*
zoom image on image display
`ZOOM/OVERLAY` *[zoom_fact] [center]*
zoom image + overlay together

A.2 Application Commands

`ASSOCIATE/RANK` *table col1 col2 [action]*
rank-order correlation coefficient

`BIN/TABLE` *table col1 col2 [bin] [min] [max] [sigma]*
creates a table, bin.tbl with averages of col2 in bins of col1

`COMPARE/2SAM` *table col1 col2*
kolmogorov two-sample test
`COMPUTE/FIT` *image[,error] [= function[(refima)]]*
compute fitted image values
`COMPUTE/FIT` *table y[,error] [= function[(ind)]]*
compute fitted table values
`COMPUTE/FUNCTION` *image = function[(refima)]*
compute function values, result as image
`COMPUTE/FUNCTION` *table y = function[(ind)]*
compute function values, result in table column
`CREATE/FUNCTION` *fun1[,fun2...] [library_specs]*
define user functions for fitting
`CREATE/GUI` *[name]*
Creates graphical user interfaces
`CREATE/STAR` *in_frm in_tab out_frm [n_size] [frm_specs] [dmin,dmax] [radius]*
create the profile of a reference star by adding and averaging

`EDIT/FIT` *function*
interactive function editor

`FILTER/ADAPTIV` *frame outframe [maskframe] [type] [shape] size k noise*
adaptive filtering of an image
`FIT/IMAGE` *[nfeval[,prec[,metpar]]] [image[,wgt]] [funct]*

fit image values

`FIT/TABLE` *[nfeval[,prec[,metpar]]] table :dep,[:wgt] :ind [funct]*
fit table

`FTEST/VAR` *table col1 col2*
f-test for different variances

`GET/FIT` *table [image]*
create a table for fitting subimages

`IDENTIFY/CURSOR` *table ident x [y] [error]*
identify table entries from display

`IDENTIFY/GCURSOR` *table ident x [y] [error]*
identify table entries from graphic terminal

`INTEGRATE/APERTURE` *[in_specs] [out_tab] [radius]*
integrate the flux within an aperature

`INTEGRATE/LINE` *frame [y_coo] [x_sta,x_end] [n_cur,deg] [batch] [x-pos,range]*
integrate area in a (spectral) line

`INTEGRATE/STAR` *[in_specs] [out_tab] [parameters] [mode]*
computes flux, radius and background of stars previously centered

`KSTEST/1SAM` *table col distri coeffs*
kolmogorov one-sample test

`MODIFY/FIT` *table seq_no [name]*
modify fit parameters

`PRINT/FIT` *func_name*
print function parameters

`READ/FIT` *func_name*
display fitted function parameters

`REGISTER/SESSION` *session directory file table*
Register a session in the session manager

`REPLACE/FUNCTION` *fun1[,fun2...]*
replace user functions for fitting

`SAVE/FIT` *table seq_no [name]*
save results of a regression

`SELECT/FUNCTION` *name number[,...]*
select function components
`SET/FIT` *par=value [par=value ...]*
set parameters for the FITTING package
`SHOW/FIT`
display parameters used in FITTING package
`SORT/COLUMN` *input output*
column oriented sorting of the pixels of a frame.
`SORT/ROW` *input output*
row oriented sorting of the pixels of a frame.
`STEST/MEAN` *table col1 col2*
student t-test for different means

`TUTORIAL/ALIGN`
explain the alignment of two images
`TUTORIAL/FIT`
explain the modelling of table and image data by fitting non-linear functions.

A.3 Standard Reduction Commands

A.3.1 ccdred

`ALIGN/MOSAIC` *in_frm in_tab out_frm method,data [nxrsub,nyrsub] [xref,yref] [x_size,y_size]*
Align the elements of the mosaiced frame
`BIAS/CCD` *[in_fram] [out_fram] [bs_fram]*
Correct the input frame for the bias offset using a bias frame
`COMBINE/CCD` *exp [in_spec] [out_fram]*
Combined a number of CCD *frames of the same exposure type*
`CREATE/MOSAIC` *in_cat out_frm out_tab nx_sub,ny_sub [not1,not2,...] [nocol,norow]*
Mosaic a set of (infrared) ccd frames
`DARK/CCD` *[in_fram] [out_fram] [dk_fram]*
Correct input frame for dark current offset using a dark current frame
`FIT/MOSAIC` *in_frm in_msk in_tab out_frm [match] [nxrsub,nyrsub] [xref,yref] [x_size,y_size]*
Align and match the elements of the mosaiced frame
`FIXPIX/CCD` *[in_fram] [out_fram] [fix_table] [fix_meth]*
Do a correction of bad pixels in the input frame
`FLAT/CCD` *[in_fram] [out_fram] [ff_fram]*
Do a flat field correction of the input frame

`FRCOR/CCD` *[in_spec] [out_frm] [xboxmn,xboxmx] [yboxmn,yboxmx] [clip] [lowsig,higsig]*
Make fringe correction frame(s0 from sky frames

`FRINGE/CCD` *[in_fram] [out_fram] [fr_fram] [fr_scale]*
Do a fringe correction of the input frame

`HELP/CCD` *[keyword]*
show the parameter setting of the current CCD *session*

`ILLCOR/CCD` *[in_spec] [out_frm] [xboxmn,xboxmx] [yboxmn,yboxmx] [clip] [lowsig,higsig]*
Make flat field illumination correction frame(s)

`ILLFLAT/CCD` *[in_spec] [out_frm] [xboxmn,xboxmx] [yboxmn,yboxmx] [clip] [lowsig,higsig]*
Apply correction to a flat field to remove illumination pattern

`ILLUMINATION/CCD` *[in_fram] [out_fram] [il_fram]*
Do an illumination correction of the input frame

`INIT/CCD` *[name]*
Initialize the CCD *package, optionally using the setting of a saved session*

`LOAD/CCD` *[intr]*
Load instrument/detector specifications into the CCD *context*

`MATCH/MOSAIC` *in_frm in_tab out_frm method,data [match] [nxrsub,nyrsub] [xref,yref] [x_size,y_size]*
Align and match the elements of the mosaiced frame

`MKREDT/CCD` *out_tab*
Create CCD *empty table with columns for science and calibration frames*

`OVERSCAN/CCD` *[in_fram] [out_fram] [sc_area] [mode]*
Correct the input frame for the bias offset in the overscan region

`REDUCE/CCD` *[in_spec] [out_frm]*
Do the (partial) calibration of one or more frames

`SAVE/CCD` *name*
save current CCD *session*

`SET/CCD` *keyw=value [...]*
Define the values of parameters in the current CCD *session.*

`SHIFT/MOSAIC` *out_tab [curs_opt] [csx,csy] [clear_opt]*
Get x and y shifts of the subraster in the mosaic frame

`SHOW/CCD` *[subject]*
Show (part of) the setup of the CCD *package*

`SKYCOR/CCD` *[in_spec] [out_frm] [xboxmn,xboxmx] [yboxmn,yboxmx] [clip] [lowsig,higsig]*
Make sky illumination correction frame(s)

`SKYCOR/CCD` *[in_spec] [out_frm] [xboxmn,xboxmx] [yboxmn,yboxmx] [clip] [lowsig,higsig]*
Apply sky observation to flat field to remove illumination pattern

`TRIM/CCD` *[in_fram] [out_fram] [im_sec] [del_flg]*
Extract the useful data from the ccd frame.

A.3.2 ccdtest

`TEST1/CCD` *in_cat [out_id] [meth] [option]*
Combine bias frames stored in a catalogue and display it
`TESTB2/CCD` *in_frm [out_id] [row_ran] [col_ran]*
Compute row and column average of a (averaged) bias frame
`TESTB3/CCD` *in_frm [out_id] [area] [size] [option]*
Find the hot pixels in a (combined) bias frame
`TESTB4/CCD` *in_frm [out_id] [area] [size,fac]*
Make a histogram of the pixel intensities and rebin the input frame
`TESTB5/CCD` *in_cat [out_id] [area] [size,fac]*
Do the statistics of the bias frame is a catalogue.
`TESTBA/CCD` *in_cat [out_id] [meth] [row_ran] [col_ran] [area] [size,fac]*
Do a series of tests of a catalogue of bias frames
`TESTC/CCD` *in_frm [rows] x_pix [colums] y_pix*
Compute the horizontal and vertical charge transfer efficiency.
`TESTD/CCD` *in_cat [out_id] [dec_fac]*
Do a test on a catalogue of dark current frames
`TESTF1/CCD` *in_cat [out_id] [meth] [area] [exp_ran] [option]*
Combine the flat frame in the input catalogue and display
`TESTF2/CCD` *in_frm [out_id] [area] [thresh] [option]*
Find the cold pixels in the combined low count flat.
`TESTFA/CCD` *in_cat [out_id] [meth] [area] [exp_ran] [thresh]*
Do a series of tests on a catalogue of low count flat frames
`TESTS/CCD` *in_frm1 inf_frm2 [out_frm] n_exp [dec_fac]*
Find the shutter error distribution
`TESTT1/CCD` *in_cat [out_id] [area] [option]*
Display the linearity and transfer curves of pairs of flat frames.
`TESTT2/CCD` *in_tab [out_id] [select] [tim_int]*
Fit the linearity cursves and determine the shutter offset
`TESTT3/CCD` *in_tab [out_id] [select]*
Fit the transfer curve and determine the ADU *conv. factor and RON*
`TESTTA/CCD` *in_cat [out_id] [area] [tim_int] [select]*
Do linearity and transfer tests on a catalogue of flat frames

A.3.3 do

`ASSOCIATE/IMA` *ost exptype rule_table outtable [flag] [nexp]*
associates to scientific exposures a set of suitable calibration images

`CLASSIFY/IMAGE` *table descr outcol outchar*
  classify images according to one or several rules.
`CREATE/CRULE` *table rule*
  create an classification rule for a given Observation Summary Table
`CREATE/OST` *file_specs [file_pref] intable outtable flag*
  create an Observation Summary Table
`GROUP/ROW` *table incol outcol*
  group the rows of a table by the value of one of its column

A.3.4 echelle

`AVERAGE/TABLE` *frame table xy_col outcol [size]*
  Read pixels in a frame at positions defined by a table.
`BACKGROUND/ECHELLE` *in out [radx,rady,step] [degree] [smooth] [method]*
  estimate interorder scattered light of an echelle spectrum
`BACKGROUND/SMOOTH` *input output [radx,rady] [niter] [visu]*
  estimate interorder scattered light of an echelle spectrum
`CALIBRATE/ECHELLE` *[defmtd] [wlcmtd]*
  performs order definition and wavelength calibration
`CLEAN/ECHELLE`
  Clears contexts Echelle and Spec and removes process tables
`CONVERT/ECHELLE` *input output domain function param option*
  resample echelle orders
`DEFINE/ECHE` *[ordref] [width1,thres1,slope] [defmtd] [defpol]*
  define echelle order positions
`DEFINE/HOUGH` *[ordref] [nbord] [hwid] [hough_par] [thresh] [degx,degy] [hot_thres,step] [hough_setup]*
  define echelle order positions; automatic detection by Hough transform.
`DEFINE/SKY` *ima [nsky] [possky] [half_width]*
  defines limits of the sky windows
`DISPLAY/ECHELLE` *image [g_flag]*
  Optionally creates a display and graphic windows and scales an image to be displayed.
`ERROR/ECHELLE` *command keyword*
  Low-level error message generator for the echelle package
`EXTR/ECH` *input output [params] [method]*
  extract echelle orders
`EXTR/OPT` *input output slit,ord1,ord2 [ron,g,sigma] [table] [coeffs]*
  weigthed extraction of echelle orders
`EXTRACT/ORDER` *inp out sl,ang,off meth table coeff [ord1,ord2]*
  Extract echelle orders and produces a frame in space pixel-order
`EXTRACT/SKY` *in out [mode]*
  Extracts sky spectrum.

`FILTER/ECHELLE` *input output*
  filter echelle frame for cosmic ray hits and subtract background
`FLAT/ECHELLE` *[flat] [correct] [blaze]*
  subtract background from flat-field image and approximate blaze profile
`HELP/ECHELLE` *keyword [mode]*
  Provides short help on an echelle session keyword
`HOUGH/ECHELLE` *input [scan] [step,nbtr] [nbord] [flags] [hwid] [thres] [params]*
  perform Hough transform and orders detection on a flat-field frame
`IDENT/ECHEL` *[wlc] [lincat] [dc] [tol] [wlcloop] [wlcmtd] [guess,[shift]] [ccdbin]*
  perform wavelength calibration of echelle spectra
`INIT/ECHELLE` *[name]*
  initializes echelle parameters
`INITIAL/EMMI` *[ref] [grism]*
  Initializes the Echelle context for a given EMMI *configuration*
`KEYDEL/ECHELLE` *[table]*
  Deletes echelle session keywords
`LOAD/CALIBRATION`
  display wavelength calibration result
`LOAD/ECHELLE`
  display echelle orders (and optionally background) positions
`LOAD/IDENTIFICATION`
  display initial identifications.
`LOAD/SEARCH`
  Loads on display the position of the lines found by the SEARCH/ECHELLE *command.*
`MERGE/ECHELLE` *inframe outframe [params] [method]*
  merge echelle orders
`MERGE/OPTIMAL` *rebima weight out [delta]*
  Optimal weighted merging of echelle orders
`OFFSET/ECHELLE` *[image] [range] [cover] [ordtab] [mode]*
  Determines the offset along the slit between the order coefficients and a given echelle spectrum.
`OVERLAP/ECHELLE` *rebima order*
  Plots the overlap region between adjacent orders n and n+1
`PLOT/CALIBRATE` *[ord1,ord2]*
  plot dispersion relation in echelle reduction
`PLOT/ECHELLE` *frame [ord1,ord2] [printer]*
  plot extracted echelle orders.
`PLOT/IDENTIFICATION` *frame [ord1,ord2] [printer]*
  plot line identifications in echelle reduction
`PLOT/RESIDUAL` *[ord1,ord2]*
  plot dispersion residuals in echelle reduction

`PLOT/SPECTRUM` *in [start,end]*
    plots a rebinned spectrum in wavelength range
`PREPARE/BACKGROUND` *[step] [init] [back_tab] [order_tab] [descr]*
    low-level command; create table back.tbl
`PREPARE/WINDOW` *catalogue flat-bkg lhcuts*
    prepare echelle images for the command AVERAGE/WINDOW
`REBIN/ECHELLE` *input output sample*
    rebin echelle orders into wavelength
`REDUCE/ECHELLE` *input output [bkcor]*
    reduction of echelle spectra.
`REGRESSION/ECHELLE` *[defpol] [niter] [absres] [kappa]*
    fit 2-dim. polynomial to order positions (defpol limited to 5,5)
`REPEAT/ECHELLE` *[scalx,scaly] [response]*
    iterate on the response computation
`RESPONSE/ECHELLE` *[std] [fluxtab] [response]*
    compute instrument response
`RIPPLE/ECHELLE` *input output [params] [method] [option]*
    correct for the blaze function
`ROTATE/ECHELLE` *cat,ima root-name [mode] [flip_axis] [angle] [o_time]*
    rotate (and optionally flip) echelle images
`SAVE/ECHELLE` *name*
    saves current echelle session
`SAVINIT/ECHELLE` *ima,tab mode*
    saves/reads echelle session keywords as descriptor of an image/table
`SCAN/ECHELLE` *frame [scan-par]*
    update echelle keywords SCAN and IMSIZE.
`SEARCH/ECHELLE` *frame [width2,thres2]*
    search for emission lines
`SEARCH/ORDER` *[ordref] [w,t,s] [ordtab] [defmtd]*
    define echelle order positions
`SELECT/BACKGROUND` *[all]*
    interactive unselection of background reference positions
`SET/ECHELLE` *par=value [...]*
    set echelle keywords
`SHOW/ECHELLE`
    show echelle session
`SUBTRACT/BACKGROUND` *input bkg output [bkgmtd] [bkgvisu]*
    compute and subtract background from input frame.
`TUTORIAL/ECHELLE`
    demonstrates main commands of echelle package
`UPDATE/ECHELLE` *image*
    low-level command handling image geometry in world-coordinates
`UPDATE/ORDER` *image [offset]*
    Updates order definition coefficients and background table.

`VERIFY/ECHELLE` *file [type]*
check consistency of frame size against predefined values.

A.3.5 irac2

`ACUTS/IRAC2 [image] [load] [plot] [upper]`
Display an image with cuts mean-3*sig and mean+upper*sig

`CMASK/IRAC2 ffield clnffield lthrshold,hthrshold [dispflag]`
create a mask of bad pixels using a flatfield.

`DCOMB/IRAC2 [select] [seqname] [accsky] [align] output [trim] [tag]`
Sky subtract and combine dithered images.

`FFIELD/IRAC` *obj_frame ff_frame out_frame*
Flat Field an IRAC *frame*

`FOCUS/IRAC2 seqnum [focout] [create]`
Used to determine the best focus from a focus sequence.

`LAST` *[num]*
Gives very brief information on the most recent exposures

`MASK/IRAC2 inframe outframe`
replaces bad pixels by closest good pixel

`MKFLAT/IRAC` *lamp_on lamp_off flat_field*
Make a flat field

`OBSLIST/IRAC2 [start] [end]`
Lists a subsection of the IRAC2B *OST (Observation Summary Table)*

`OBSREP` *start end*
Print out a subsection of the IRAC2B *OST (Observation Summary Table)*

`QL/IRAC2 image1 image2 [outimage]`
Subtracts one IRAC2 image from another taking into account the detector integration times.

`RCOMB/IRAC2 select [align] output`
Combine frames created with the command RCOMB/IRAC2

`SEEING/IRAC2`
Determine the seeing, defined as the FWHM *of stellar images, of* IRAC2 images.

`SSUB/IRAC` *obj_frame sky_frame out_frame*
Sky Subtract an IRAC *frame*

A.3.6 irspec

`BADPIX/IRSPEC` *in out [l=load_opt.] [dir=clean_opt.] [debug=debug_opt.]*
Clean image of fixed pattern of bad pixels

`CALIBRATE/IRSPEC` *ima*

Apply on-line (mechanical) wavelength calibration.

`CALIBRATE/IRSPEC` *ima_ref mode=define*

Define and store in ima_ref precise calibration from sky/lamp lines

`CALIBRATE/IRSPEC` *ima ref=ima_ref*

Apply precise wavelength calibration to frame ima from parameters stored in ima_ref

`DEFINE/IRSPEC` *image table [mode] [threshold] [number] [load_option]*

Define fixed pattern of bad pixels and store it into a table.

`FLAT/IRSPEC` *in_flat in_dark out [l=load_opt.] [t=threshold] [v=vignetted_value]*

Create a normalized flat from an input flat frame.

`FLUX/IRSPEC` *in_ima response_ima out_ima [smooth=s1,s2] [shift=sh] [norm=normalize_option] [rect=rectify_option]*

Flux calibrate a spectrum (either 2D *or 1D) using a response frame* created using RESPONSE/IRSPEC

`MERGE/IRSPEC` *prefix_ima i1,i2[,i3] out_table [excl=#pixels_excluded] [corr=correct_option] [ref=#reference_image] [plot=plot_option] [format=i_format]*

Merge 1D *spectra into a table forcing connection of overlapping regions*

`RECTIFY/IRSPEC` *in out [l=load_opt.] [tilt=tilt_value] [ref=reference_row]*

Rectify tilted spectral lines.

`RESPONSE/IRSPEC` *in_ima flux_table out_response_ima [yrows=y1,y2,y3,y4] [obs=observation_mode] [norm=normalize_option] [rect=rectify_option]*

Create a response frame from a standard star 2D *spectrum and* a flux table.

`SKYSUB/IRSPEC` *ima_obj ima_sky out factor[,shift[,deltax,deltay]] [sky=sky_table] [force=force_sky_to_zero] [cuts=cuts_values] [debug=debug_option]*

Perform obj-sky correcting for variation and shift of sky lines.

`STANDARD/IRSPEC` *in_ascii_file out_table interp_method [degree=degree] [step=wavelength_step] [limits=wl1,wl2] [units=wavelength_units] [plot=plot_option]*

Create standard star flux table from a "flux ascii file".

`SUBTRACT/IRSPEC` *in_ima out_ima degree [exclude=area_to_exclude] [cont=continuum_image] [load=load_option]*

Fit and subtract, row by row, polynomial to a given image.

`TUTORIAL/CALIBR`

demonstration of wavelength calibration commands in IRSPEC

`TUTORIAL/IRSPEC`

General tutorial for the package IRSPEC

`TUTORIAL/SKYSUB`

Tutorial for sy substraction with IRSPEC *package*

A.3.7 long

`APPLY/DISPERSION` *in out [y] [coef]*
    Apply the dispersion relation to a 1D *spectrum and generates a table*

`BATCH/LONG`
    Prepare the Batch Reduction user interface

`CALIBRA/FLUX` *in out [resp]*
    Correct an image for the instrumental response function

`CALIBRATE/LONG` *[tol] [deg] [mtd] [guess]*
    Wavelength calibration of 1D *and long-slit spectra*

`CALIBRATE/TWICE`
    Performs the wavelength calibration on a selected set of lines.

`CLEAN/LONG`
    Clear context Long

`COMBINE/LONG` *catalog output [mtd]*
    Average a catalog of images

`EDIT/FLUX` *[resp]*
    Edit the instrumental response table

`ERASE/LONG`
    Interactive rejection of dispersion relation nodes.

`ESTIMATE/DISPERSION` *wdisp wcent [ystart] [line] [cat]*
    Estimate a linear approximation of the dispersion relation

`EXTINCTION/LONG` *in out [scale] [table] [col]*
    correct spectra for interstellar or atmopsheric extinction

`EXTRACT/AVERAGE` *in out [obj] [sky] [mtd]*
    extract a long-slit spectrum by averaging rows

`EXTRACT/LONG` *in out [sky] [obj] [order,niter] [ron,g,sigma]*
    Optimal extraction of a long-slit spectrum

`GCOORD/LONG` *[number] [outtab]*
    Get coordinates from the display window

`GRAPH/LONG` *[size] [position] [id]*
    Creates a graphic window

`HELP/LONG` *[keyword]*
    provides information about session keywords.

`IDENTIFY/LONG` *[wlc] [ystart] [lintab] [tol]*
    Interactive calibration of lines in an arc spectrum

`INITIALIZE/LONG` *[session]*
    Initialises parameters of context long

`INTEGRATE/LONG` *std [flux] [resp]*
    Generates an intermediate response table from a standard star spectrum

`LINADD/LONG` *in w,bin [y] [mtd] [mode] [line]*
    Adds entries to the table line.tbl

`LOAD/LONG` *image [scale_x,[scale_y]]*

`MAKE/DISPLAY`
    Creates a display window
`NORMALIZE/FLAT` *in out [bias] [deg] [fit] [visu]*
    Normalisation of flat-fields
`PLOT/CALIBRATE` *[mode]*
    Plot wavelength calibration identifications.
`PLOT/DELTA` *[mode]*
    Plot the fitted dispersion relation and allow interactive rejection of arc lines.
`PLOT/DISTORTION` *wave [delta] [mode]*
    Plot the fitted position of arc lines in wavelength/y-coordinate space.
`PLOT/FLUX` *[fluxtab]*
    Plot the flux table
`PLOT/IDENT` *[wlc] [line] [x] [id] [wave]*
    Plot interactive identifications
`PLOT/RESIDUAL` *[y] [table]*
    Plots residual after wavelength calibration
`PLOT/RESPONSE` *[resp]*
    Plots the response correction function
`PLOT/SEARCH` *[mode] [table]*
    Plot the results of SEARCH/LONG
`PLOT/SPECTRUM` *table*
    Plots a 1D *spectrum in table format, as supplied by APPLY/DISPERSION*
`PREPARE/LONG` *in [out] [limits]*
    Extracts sub-images from an image or a catalog.
`REBIN/LONG` *in out [start,end,step] [mtd] [table]*
    Rebin a long-slit spectrum using the row-by-row method
`RECTIFY/LONG` *in out [reference] [nrep] [deconvol_flag] [line]*
    rectify geometrically a distorted 2-D *spectrum*
`REDUCE/INIT` *partab*
    Initialises the batch reduction parameters
`REDUCE/LONG` *input*
    Batch reduction of long-slit spectra.
`REDUCE/SAVE` *partab*
    Saves the batch reduction parameters
`RESPONSE/FILTER` *std [flux] [resp]*
    Generate a response image by filtering based method.
`RESPONSE/LONG` *[plot] [fit] [deg] [smo] [table] [image] [visu]*
    Converts the response correction from table to image format.
`SAVE/LONG` *session*
    Saves session keywords
`SEARCH/LONG` *[in] [thres] [width] [yaver] [step] [mtd] [mode]*
    search for spectral features in a long-slit spectrum
`SELECT/LINE`

Select lines identified in all rows of an arc spectrum.

`SET/LONG` *key=value [...]*
    Assigns a value to long-slit session keywords

`SHOW/LONG` *[section]*
    Displays values of session keywords.

`SKYFIT/LONG` *input output [sky] [degree] [mode] [r,g,t] [radius]*
    fit polynomial to spatial flux distribution in windows of every column

`TUTORIAL/LONG`
    demonstrate commands of the package Long

`VERIFY/LONG` *file mode*
    Checks conformity of files in the long-slit context

`XIDENT/LONG` *[wlc] [ystart] [lintab] [tol]*
    Invoke the identification graphical user interface

A.3.8 optopus

`CREATE/OPTOPUS` *inp_file [out_tab] [fmt_file] [old_equinox]*
    create input table for HOLES/OPTOPUS command

`DRILL/OPTOPUS` *in_table [name]*
    write OPTOPUS *drill command file*

`HOLES/OPTOPUS` *[inp_tab] [out_tab] [HH,MM,SS.sss] [+/-DD,AM,AS.ss] [ac_flag] [p_flag] [old_eq,new_eq]*
    detetermine holes positions on Optopus plate.

`MODIFY/OPTOPUS` *[table]*
    plot positions of holes on plate and enable rejection of objects.

`PLOT/OPTOPUS` *[table] [label] [EW_flip_flag]*
    plot positions of holes on Optopus plate.

`PRECESS/OPTOPUS` *[inp_tab] [new_equinox]*

`p recess RA` *and DEC coordinates in table created by CREATE/OPTO.*

`REFRACTION/OPTOPUS` *[inp_tab] [out_tab] [year,month,day] [exp] [lambda1,lambda2] [start_st_slot,end_st_slot] [opt_st] [ast_flag]*
    correct for atmospheric refraction X *and Y coord. on Optopus plate.*

`RESTORE/OPTOPUS` *table*
    restore previously saved session parameters.

`SAVE/OPTOPUS` *table*
    save session parameters in descriptors

`SET/OPTOPUS` *option1[=value1] option2[=value2] ...*
    set Optopus context parameters

`SHOW/OPTOPUS`
    show session parameters.

`ZOOM/OPTOPUS` *[table] [zooming_factor]*
    blow up section of Optopus plate and enable rejection of objects.

A.3.9 pisco

`REDUCE/PISCO` *catalog table sky calibration [mode]*
perform complete reduction of polarimetric data

A.3.10 spec

`CENTER/HISTOGRAM` *image*
Median estimate and scale estimates of an image
`COMPUTE/PARAL` *ra dec st wave refw*
Computes parallactic angle and atmospheric differential refraction
`CONTINUUM/SPEC` *in out [radius/meth] [type] [smooth] [degree]*
Fitting of a spectrum continuum by smoothing splines
`CORRELATE/LINE` *table_1 table_2 [pixel] [cntr,tol,rg,st] [pos,ref,wgt] [ref_value] [outima]*
Cross-correlation between table columns.
`CUMULATE/HISTOGRAM` *in out*
Transforms a histogram image into the cumulated histogram
`DEBLEND/LINE` *infile [fitim] [fitpar] [method] [contin] [input] [intab]*
multiple component Gaussian fitting of spectral lines
`DISPERS/HOUGH` *[wdisp] [wcent] fr_specs [line] [cat] [mode] [range] [vflag]*
Determination of dispersion relations by HT
`EXTINCTION/SPECTRUM` *inframe outframe scale [table] [col]*
correct spectra for interstellar or atmopsheric extinction
`FILTER/RIPPLE` *frame outframe period [start,end]*
correct 1-dim. images for periodic ripple (Reticon)
`GRAPH/SPEC` *[size] [position] [id]*
Creates a long graphic window adpated for spectroscopy
`HST/SPEC` *fits_wave fits_flux table*
converts two HST *FITS files providing wavelength and flux information*
`MERGE/SPECTRUM` *spec1 spec2 out [interval] [mode] [var1] [var2]*
merge two 1D *spectra*
`NORMALIZE/SPECTRUM` *inframe outframe [mode] [table] [batch_flag]*
approximate continuum of 1-dim. spectra for later normalization
`OVERPLOT/IDENTIFICATION` *[table] [xpos] [ident] [ypos]*
overplot line identifications
`REFRACTION/LONG` *inim outim [mode]*
Differential atmospheric correction for slit spectra
`REGRESSION/ROBUST` *tab y x1[,x2,..,xn] [file] [out_col] [res_col]*
Robust multi-variate regression by Least Median of Squares
`ROTATE/SPEC` *cat [root] [meth] [flip] [angle] [mode]*
rotate (and optionally flip) a catalog of images
`SEARCH/LINE` *frame w,t[,nscan] [table] [meth] [type]*

search for spectral lines

`VERIFY/SPEC` *file dir keyw [type]*
: low-level spec command checking the existence of calibration tables

A.4 Contributed Commands

A.4.1 astromet

`ASTROMETRY/COMPUTE` *mes option out trail*
: Convert the coordinates from the measured xy ot RA,Dec vice versa

`ASTROMETRY/EDIT` *std plot*
: Delete/undelete the astrometric standards

`ASTROMETRY/POS1`
: Interactive procedure for the POS1 astrometry package

`ASTROMETRY/TRANS` *std mes center pla,cat schmidt,blink tol xterm,yterm std*
: Compute the astrometric transformation parameters of a data

A.4.2 cloud

`COMPUTE/ABSORPTION` *inframe outframe [cm_table] [ap_table] [psframe]*
: computes a synthetic 1dim. absorption spectrum

`COMPUTE/EMISSION` *outframe [em_table]*
: computes a synthetic 1dim. emission spectrum

`CREATE/PSF` *[outframe] fwhm*
: creates a 1-dim. image of a normalized gaussian

A.4.3 daophot

`ALLSTAR/DAOPHOT`
: do simultaneous multiple-profile-fitting

`DAOMID/DAOPHOT` *table*
: convert a DAOPHOT table into a MIDAS table

`DAOPHOT/DAOPHOT`
: do precise photometry and astrometry in a 2-dim frame

`MIDDAO/DAOPHOT` *table*
: convert a MIDAS table into a DAOPHOT table

A.4.4 esolv

`FROMOD/ESOLV` *[mode] [colour] [intable] [column]*
: retrieve frames from optical disk

`MTABLV/ESOLV [table] col1 col2 l_frac col3`
  find semidiameter of ellipses at given fraction of light
`STATPL/ESOLV table col1 select disp`
  computes mean and sd of table file column given in p2
`TABFLV/ESOLV [table] ascii_file flag`
  lists the contents of special table file
`TEXLV/ESOLV [table] tex_file`
  prepare Tex file with selected parameters from ESOLV

A.4.5 geotest

`CREATE/ART_IMAGE` *frame frame dims [starts,steps] [func_type] [coefs]*
`CREATE/ART_IMAGE` *frame = ref_frame [func_type] [coefs]*
  create artificial image
`CREATE/RAMP` *image [slope] [angle] [dimension]*
  generate uniform sloping image, with mean flux per pixel of 100 units
`CREATE/SPC1 image [slope] [ampl] [period] [phase] [dim]`
  generate sinusoidal, sloping 1-dimensional image
`CREATE/SPC2 image [period] [slope] [phase] [dimension]`
  generate a discrete 1-dimensional image
`CREATE/SPC3 image psf_option centring table boxwidth-or-fwhm`
  generate an artificial spectrum with lines
`CREATE/WAVE` *image [amplitude] [period] [dimension]*
  generate 2-dimensional sinusoidal background image

A.4.6 invent

`ANALYSE/INVENTORY` *frame in_tab [out_tab] [ver_par] [deb_mode] [out_psf]*
  verify the used table of objects and calculates the image parameters
`CLASSIFY/INVENT table`
  classify the analysed objects into stars, galaxies and spurious objects
`SEARCH/INVENTORY` *frame table*
  search objects in an image frame and store the parameters
`SET/INVENTORY` *par1 [par2]*
  display and modify the values of the keywords used by Inventory
`SHOW/INVENTORY` *par1 [par2]*
  display the values of the keywords used by the Inventory package

A.4.7 mva

`CLUSTER/TABLE` *intable outable [method]*
  hierarchical clustering

`CMDS/TAB input_table output_table ncols._output_table`
  multidimensional scaling

`CORRES/TAB input output row/column_analysis ncolumn outable`
  correspondence analysis

`EDIST/TAB input_table output_table`
  standard distances

`KNN/TAB` *training_table no._of_gp.1_members test_table no._of_NNs*
  discriminant analysis

`LDA/TAB Input_table Output_table`
  Fisher's linear discriminant analysis.

`MDA/TAB input_table output_table eigenvectors`
  discriminant analysis

`MST/TABLE intable outtable grid_size`
  create minimal spanning tree for position table

`PARTITION/TABLE` *intable outable [no_of_class] [alg] [min. card] [s_value]*
  non-hierarchical clustering

`PCA/TAB in_tab out_tab option row/col_anal ncols_table eigenvectors`
  principal components analysis

`PLOT/TREE` *intable [col_ref]*
  plot output created by minimal spanning tree algorithm

A.4.8 pepsys

`CONVERT/PHOT`
`Helps you make a new table of observational data`
`MAKE/HORFORM`
`Make a blank FORM` *to fill in with horizon-obstruction data*
`MAKE/PHOTOMETER`
`generates or checks the instrumental table file for a photometer`
`MAKE/PLAN`
`generates a photometric observing plan`
`MAKE/STARFILE` *arglist*
`Helps you make a new file of program or standard stars`
`REDUCE/PHOT`
`Reduces tables of observational data`

A.4.9 romafot

`ADAPT/ROMAFOT` *int_tab [thres] [fac_int] [fac_sky] [fac_hol] [x_siz,y_siz]*
  derive trial values for fitting a new frame

`ADSTAR/ROMAFOT` *in_frame out_frame [reg_tab] [cat_tab] [x_dim,y_dim] [n_sub]*
  create an artificial image with subframes added at random positions

`ANALYSE/ROMAFOT` *frame [cat_tab] [int_tab] [sigma,sat]*
  INPUT MODE select all stars within selected subfields;
  OUTPUT MODE check at the results of the fit operation and select
`CBASE/ROMAFOT` *frame_1 frame_2 [out_tab1] [out_tab2]*
  create two tables for coordinate transformation
`CHECK/ROMAFOT` *cat_tab reg_tab err_mag*
  examine number of artificial stars recovered and check the accuracy
`CTRANS/ROMAFOT` *int_tab [tab_1] [tab_2] [pol_deg]*
  find transformation of coordinates and apply to an intermediate table
`DIAPHRAGM/ROMAFOT` *frame [regi_tab] [rego_tab] ap_rad*
  do aperture photometry with fixed diaphragm
`EXAMINE/ROMAFOT` *int_tab [hmin,hmax]*
  examine quality of fitted objects and flag badly fitted ones
`FCLEAN/ROMAFOT` *cat_tab inti_tab [into_tab]*
  selects windows in intermediate table present in catalogue table
`FIND/ROMAFOT` *frame [cat_tab]*
  select objects using the image display
`FIT/ROMAFOT` *frame [int_tab] [thres,sky] [sig,sat,tol,iter] [meth,[beta]] [fit_opt] [mean_opt]*
  determine characteristics of stellar images by non-linear fitting
`GROUP/ROMAFOT` *frame [area] [cat_tab] [int_tab] [thres] [wnd_max] [end_rad,sta_rad] [wnd_perc]*
  automatic grouping of objects
`MFIT/ROMAFOT` *frame [int_tab] [thres,sky] [sig,sat,tol,iter] [meth,[beta]] [fit_opt] [mean_opt] [mod_file]*
  determine characteristics of stellar images by non-linear fitting
`MODEL/ROMAFOT` *[mod_file]*
  compute (sub)pixel values for a model observation
`REGISTER/ROMAFOT` *int_tab reg_tab [wnd_opt] [obj_opt]*
  computes and store the absolute quantities in the registration table
`RESIDUAL/ROMAFOT` *in_frame out_frame diff_frame [reg_tab]*
  compute reconstructed image and difference with original image
`SEARCH/ROMAFOT 07-AUG-1989 RHW`
  do the actual search for objects above a certain threshold
`SELECT/ROMAFOT` *frame [int_tab] [wnd_size]*
  select objects and store the positions in intermediate table
`SKY/ROMAFOT` *frame [sky_tab] [area] [nrx,nry] [min,max]*
  determines intensity histogram and sky background in selected areas

A.4.10 surfphot

`COMPUTE/FCOEFF` *infram orient rin,rout,rstep outtab*
  compute fourier coefficients of azimuthal profiles in spiral galaxies

`COMPUTE/GRID` *angle*
: create image and table with distorted rect. and evenly spaced grid

`COMPUTE/SKY` *infram1 infram2 caltab method sky_factor*
: compute the sky background and restitute the frame

`FILTER/FILL` *inframe outframe rx,ry thresh*
: fill up low-flux (below threshold) pixels with nearby high-flux pixels

`FIND/PAIR` *intab1 intab2 outtab colums [errors] [coo_sys]*
: match (pair) two coordinates tables and produce an output table

`FIND/POSINC` *infram x_pos,y_pos rin,rout,rstep*
: find the position angle and inclination of a galaxy

`FIT/BACKGROUND` *outframe = inframe(s) [deg,it] [clp1,clpn] [skew] [outbck]*
`FIT/BACKGROUND` *outframe = inframe(s) [coef] [outbck]*
`FIT/BACKGROUND` *inframe(s) [deg,it] [clp1,clpn] [skew] [outbck]*
`FIT/BACKGROUND` *inframe(s) [coef] [outbck]*
: compute 2-dim. polynomial fit of the background

`FIT/ELL1 inframe outframe l_iso,h_iso x_cen,y_cen max_rad`
: fit an ellips with respect to predefined center

`FIT/ELL2 inframe pol_opt iso_tol iso_levels [center[ [radius] [sky_level]`
: fit an ellips with respect to predefined center

`FIT/ELL3 inframe outframe [step] [x,y] [low,high] [min,max] [opt]`
: fit ellipses to the isophotes of an object in a 2-dim. frame

`FIT/POSINC` *infram orient rin,rout,rstep region*
: fit the position angle and inclination to 2nd and 4th harmonic

`INTEGRATE/ELLIPS` *frame [ellips_par] [flag]*
: integrate pixel intensities within ellipse in 2-dim. image

`NORMALIZE/IMAGE` *infram outfram trunc_vals control_vals*
: normalize and truncate a frame

`REBIN/DECONVOLVE` *frame psf result zoom_x,zoom_y n_iter*
: rebin image linearly in space and simultaneously deconvolve it with psf

`RECTIFY/IMAGE` *in out table [nrep] [deconvol_flag]*
: rectify geometrically a distorted direct image

`SUBTRACT/SKY` *inframe outframe nx,ny*
: remove sky by subtraction of histogram-modeled substitute-sky

A.4.11 tsa

`AOV/TSA` *intab outima start step nsteps [order] [cover]*
: compute analysis of variance periodogramme

`BAND/TSA` *intab [maxobs] Evaluate frequency band for time series analysis*

`COVAR/TSA` *intab1 intab2 outtab start step nsteps scale*
: compute discrete covariance function for unevenly sampled data

`DELAY/TSA` *intab1 intab2 outtab start step nsteps [func,mode] [parm]*
: compute chi2-time lag function

`INTERPOLATE/TSA` *intab outtab func parm*
  Interpolate an unevenly sampled series using its covariance function
`NORMALIZE/TSA` *intab1 outtab column [mode]*
  Normalize mean and variance to 0 and 1
`ORT/TSA` *intab outima start step nsteps [order]*
  compute multiharmonic AOV *periodogram*
`POWER/TSA` *intab outima start step nsteps*
  Compute discrete power spectrum for uneven sampling by slow method
`SCARGLE/TSA` *intab outima start step nsteps*
  Compute Scargle periodogramme for unevenly spaced observations
`SET/TSA set global keywords for TSA` *context*
`SHOW/TSA show global keywords for TSA` *context*
`SINEFIT/TSA` *intab outtab freque order iter*
  fit sine (Fourier) series, subtract it from input and return residuals
`TABLE/TSA` *inascii [outtab] [type] [mxcol] convert ASCI table into MIDAS table*
`WIDTH/TSA` *inima [width] [centre] Evaluate line width and profile*

A.5 Procedure Control Commands

`BRANCH` *var comparisons labels*
  multi-way branching
`CROSSREF` *labl1 labl2 labl3 labl4 labl5 labl6 labl7 labl8*
  define cross reference labels for the 8 parameters
`DEFINE/LOCAL` *key_def data A lower_levels_flag*
  define the maximum no. of parameters for a procedure
`DEFINE/PARAMETER` *Pi default type/option prompt_str low_lim,hi_lim*
  define default, type and valid interval for parameter i
`DO` *loopvar = start end [step]*
  define a DO *loop (as in FORTRAN)*
`ENTRY` *proc*
  define begin of procedure in a file with different name than the procedure
`GOTO` *label*
  branch to command line containing label:
`IF` *par1 op par2 command_string*
  execute conditional statement
`INQUIRE/KEY` *key prompt_string*
  get terminal input in a MIDAS *procedure*
`LABL:`
  define a label, LABL *in this example*
`RETURN` *par1 par2 par3*
  return to calling procedure (or terminal) and optionally pass up to 3

parameters back

‘

A.6 Commands Grouped by Subject

In the following list, general MIDAS commands are given grouped in main application areas. Only the most common commands are listed to make the list easier to use. Commands used for special types of data reduction are given in appropriate chapters in the main part of this manual.

A.6.1 MIDAS System Control

@@	Execute a MIDAS procedure
BYE	Terminate the MIDAS session
CHANGE/DIRECTORY	Change the default (current) directory for MIDAS
CLEAR/CONTEXT	Clear current context level or all levels
COMPUTE/KEYWORD	Compute values of a keyword
CONNECT/BACK_MIDAS	Connect "command syntax" to another MIDAS
CREATE/COMMAND	Create a user command
CREATE/DEFAULTS	Create special defaults for MIDAS commands
DEBUG/PROCEDURE	Run MIDAS procedures in debug mode
DEBUG/MODULE	Run MIDAS modules in debug mode
DELETE/COMMAND	Delete a user defined command
DELETE/DEFAULTS	Delete special defaults for command
DELETE/IMAGE	Delete an image frame
DELETE/KEYWORD	Delete user defined keyword
DELETE/LOG	Delete log file
DISCONNECT/BACK_MIDAS	Disconnect from a background MIDAS
ECHO/FULL	Show substitutions in program files
ECHO/OFF	Suppress display of input from program files
ECHO/ON	Display input from program files
LOG/OFF	Disable logging
LOG/ON	Enable logging
PLAYBACK/LOG	Playback log file
READ/KEYWORD	Display contents of keywords
RENAME/IMAGE	Rename an image frame
RUN	Execute program inside MIDAS
SAVE/COMMANDS	Save commands from command window in a procedure
SET/CONTEXT	Set new context level
SET/FORMAT	Format for "number-to-string" conversion
SET/MIDAS_SYSTEM	Set different modes and options for MIDAS
WAIT/BACK_MIDAS	Wait until command in background MIDAS terminates

WAIT/SECS	Suspend MIDAS monitor for no_of_secs second
WRITE/COMMANDS	Store commands from a procedure into the command window
WRITE/KEYWORD	Store values into a keyword
WRITE/OUT	Write out text

A.6.2 Help and Information

HELP	Display help info for a command
HELP/...	Display info about various topics
INFO/...	Get information about frames, descriptors and specific setup
PRINT/HELP	Print help information
PRINT/LOG	Print log file
SHOW/COMMANDS	Display MIDAS commands
SHOW/DEFAULTS	Display all special defaults

A.6.3 Tape Input and Output

INDISK/...	Read data from disk in FITS or ASCII format
INTAPE/FITS	Read data from tape in FITS or IHAP format
OUTTAPE/FITS	Write data to tape in FITS format

A.6.4 Image Directory and Header

ADD/xCAT	Add one or more entries to a catalogue
COPY/DD	Copy descriptors from one file to another
CREATE/xCAT	Create a catalogue
DELETE/...	Delete a frame
DELETE/DESCRIPTOR	Delete a descriptor
INFO/DESCRIPTOR	Get type and size of descriptor
READ/DESCRIPTOR	Read descriptors
RENAME/...	Rename a frame
SORT/xCAT	Sort entries in a catalogue
SUBTRACT/xCAT	Remove an entry from a catalogue
WRITE/DESCRIPTOR	Write a descriptor
WRITE/DHELP	Write descriptor help

A.6.5 Image Display

BLINK/CHANNEL	Blink between channels
CLEAR/ALPHA	Clear the alpha-numerics memory
CLEAR/CHANNEL	Clear and initialize memory channel
CLEAR/DISPLAY	Reset monitor
CLEAR/LUT	Bypass LUT in screen segment on monitor
CLEAR/SPLIT	Disable split screen
CLEAR/ZOOM	Clear zoom

COPY/CHANNEL	Copy image memory channels
COPY/DISPLAY	Hard copy of image display
CREATE/CURSOR	Create cursor window
CREATE/ZOOM	Create zoom window
CREATE/DISPLAY	Create a display window (using Xwindow)
CUTS/IMAGE	Set display thresholds for image
DELETE/DISPLAY	Delete the display windows
DISPLAY/CHANNEL	Display image loaded into channel
DRAW/...	Draw rectangle and other figures in the overlay plane
EXTRACT/CURSOR	Extract a subframe from the frame currently displayed
EXTRACT/ROTATED	Extract a rotated subimage from displayed image
EXTRACT/TRACE	Extract interactively a line from an image
GET/CURSOR	Coordinates from display device by cursor
GET/IMAGE	Read currently loaded image from channel
INITIALIZE/DISPLAY	Initialize the image display
LABEL/DISPLAY	Write character string on display device
LOAD/CURSOR	Display cursor into display device (DeAnza only)
LOAD/IMAGE	Load image into display device
LOAD/ITT	Load an intensity transfer table
LOAD/LUT	Load a colour lookup table into display unit
LOAD/TABLE	Display table data on image display
MODIFY/LUT	Modify the currently active lookup table
SCROLL/CHANNEL	Scroll image on given channel
SET/CURSOR	Set cursor form and position
SET/DISPLAY	Define colour display control, size of screen etc. (DeAnza only)
SET/LUT	Enable use of colour lookup table
SET/SPLIT	Enable split screen (DeAnza only)
SHOW/CHANNEL	Show information related to channel
VIEW/IMAGE	Explore an image interactively
ZOOM/CHANNEL	Zoom image on display

A.6.6 Graphics Display

ASSIGN/GRAPHICS	Define plotter output device and replot
CLEAR/GRAPHICS	Clear graphic screen
COPY/GRAPHICS	Copy the plot file to the specific graphic device
CREATE/GRAPHICS	Create a graphic window (using Xwindow)
CUTS/IMAGE	Set plot thresholds (high and low) for image
DELETE/GRAPHICS	Delete the graphic windows
GET/GCURSOR	Coordinates from graphic device by cursor
LABEL/GRAPHICS	Plot text in an existing plot
OVERPLOT/ERROR	Overplot table error column

OVERPLOT/HISTOGRAM	Overplot histogram of table column or image
OVERPLOT/ROW	Overplot row/line of image data on previous plot
OVERPLOT/TABLE	Overplot table data on previous plot
PLOT/AXES	Plot a coordinate box with large and small tickmarks and labels
PLOT/CONTOUR	Contour plotting of an image
PLOT/DESCRIPTOR	Plot an entry in a descriptor
PLOT/HISTOGRAM	Plot a histogram of a table column or an image
PLOT/ROW	Plot row/line of an image
PLOT/PERSPECTIVE	Perspective plotting (3-dim.) of an image
PLOT/TABLE	Plot table data
SET/GRAPHICS	Set plot characteristics like scaling
SHOW/GRAPHICS	Show graphic characteristics

A.6.7 Image Coordinates

CENTER/...	Find center
GET/CURSOR	Coordinates from image display via cursor
GET/GCURSOR	Get coordinates from graphics device by cursor
READ/DESCRIPTOR	List reference coordinates
WRITE/DESCRIPTOR	Write reference coordinates

A.6.8 Coordinate Transformation of Images

ALIGN/IMAGE	Calculate linear transformation between 2 images
EXTRACT/IMAGE	Extract part of image
FLIP/IMAGE	Flip image in x and/or y
GROW/IMAGE	Repeat one scan line to make 2 dim images
INSERT/IMAGE	Insert a subimage into father image
REBIN/II	Logarithmic, exponential, $r^{1/4}$ frequency rebin
REBIN/LINEAR	Pixel rebinning of image
REBIN/ROTATE	Rotate an image any angle
REBIN/SPLINE	Rebin an image with cube splines
REBIN/WAVE	Rebin 1-D image to linear wavelength
RECTIFY/IMAGE	General geometric correction
ROTATE/CLOCK	Rotate clockwise 90 degrees
TRANSPOSE/CUBE	Rearrange planes of 3-dim data cube
TRANSPOSE/IMAGE	Transpose an image

A.6.9 Image Arithmetic

AVERAGE/AVERAGE	Compute simple average of all pixels in a subimage
AVERAGE/COLUMN	Compute average of image columns
AVERAGE/IMAGE	Calculate the average of images
AVERAGE/ROW	Compute average of image rows
AVERAGE/WINDOW	Compare images, then take the meaning

COMPUTE/COLUMN	Perform arithmetic expression on image column
COMPUTE/IMAGE	Compute arithmetic expression of images
COMPUTE/PIXEL	Perform arithmetic operations on images using pixel coordinates
COMPUTE/ROW	Compute arithmetic expression on image scan lines
COMPUTE/..PLANE	Do arithmatic on planes of a data cube

A.6.10 Filtering

CONVOLVE/IMAGE	Convolve image with given point spread function
CREATE/FILTER	Create filter image
DECONVOLVE/IMAGE	Deconvolve image with point spread function
FILTER/GAUSS	Use Gauss filter on image
FILTER/MAX	Apply maximum filter to an image
FILTER/MEDIAN	Median filter image
FILTER/MIN	Apply minimum filter to an image
FILTER/SMOOTH	Smooth an image
FFT/IMAGE	Compute discrete fourier transform of a complex input frame
FFT/INVERSE	Compute inverse discrete fourier transform of a complex input frame

A.6.11 Image Creation and Extraction

COPY/II	Copy image frames
CREATE/IMAGE	Create new image
CREATE/RANDOM	Create a new image from a random distribution
EXTRACT/CURSOR	Extract a subframe from the frame displayed
EXTRACT/IMAGE	Extract part of an image
EXTRACT/LINE	Extract a line from a frame
EXTRACT/ROTATED	Extract a rotated image
EXTRACT/SLIT	Extract subimage defined by fixed slit
EXTRACT/TRACE	Extract line from an image
INDISK/ASCII	Read ASCII file from disk
INDISK/FITS	Read FITS file from disk
INSERT/IMAGE	Insert a subimage into father image

A.6.12 Transformations on Pixel Values

FIT/FLAT_SKY	Correct an image for sky variations
ITF/IMAGE	Transform pixel values in an image
MODIFY/AREA	Remove bad pixel from circular area
MODIFY/CURSOR	Change pixel values in image by cursor
MODIFY/GCURSOR	Change pixel values in image by graphic cursor
MODIFY/PIXEL	Change pixel values in image

REPLACE/IMAGE	Modify pixel values in given intensity interval
REPLACE/POLYGON	Replace pixel values inside a polygon

A.6.13 Numerical Values of Image Pixels

FIND/MINMAX	Display (ands tore) max and min value
FIND/PIXEL	Find first pixel with a value inside or outside the interval
FIT/FLAT_SKY	Fit background image
INTEGRATE/APERTURE	Integrate flux inside aperture
INTEGRATE/LINE	Integrate pixel-values over area in image
MAGNITUDE/CIRCLE	Compute the magnitude of the specified object by integrating over the central area defined by a circular aperture
MAGNITUDE/RECTANGLE	Compute the magnitude of the specified object by integrating over the central area defined by a rectangular aperture
MODIFY/CURSOR	Change pixel values in image by cursor
MODIFY/GCURSOR	Change pixel values in image by graphic cursor
MODIFY/PIXEL	Change pixel values in image
PLOT/HISTOGRAM	Plot histogram of pixel values in image
PRINT/IMAGE	Print an image
READ/IMAGE	List pixel values into image
STATISTICS/IMAGE	Calculate statistics of an image
WRITE/IMAGE	Change pixel values in image (world coordinates)

A.6.14 Spectral Analysis

CALIBRATE/LINE	Calculate coefficients for wavelength calibration
CENTER/...	Compute center of line
CONVERT/TABLE	Make image from table values
EXTINCTION/SPECTRUM	Correct 1-D image for extinction
IDENTIFY/GCURSOR	Identify table entries from graphic display
IDENTIFY/LINE	Equate X positions to wavelengths
INTEGRATE/GCURSOR	Integrate line interactively
MODIFY/GCURSOR	Change data in line interactively
OVERPLOT/IDENT	Overplot line identifications
PLOT/IDENT	Plot line identifications
REBIN/...	Linear or non-linear image rebinning
RESPONSE/SPECTRUM	Make file for flux correction, response curve
SEARCH/LINE	Search calibration lines

A.6.15 Least Squares Fitting

COMPUTE/FIT	Compute fitted image or table

COMPUTE/FUNCTION	Compute function values of image or table
EDIT/FIT	Define function for fitting
FIT/IMAGE	Least squares fitting in image
FIT/TABLE	Least squares fitting in table
PRINT/FIT	Print fitted values
READ/FIT	Read fitted values
SELECT/FUNCTION	Select functions to be fitted
SET/FIT	Control execution of fitting
SHOW/FIT	Display control parameters

A.6.16 Table File Operations

BIN/TABLE	Create a table with averages of col2 in bins of col1
COMPUTE/HISTOGRAM	Compute histogram for a table column
COMPUTE/REGRESSION	Compute column from regresion coefficients
COMPUTE/TABLE	Compute arithmetic expression between columns
CONVERT/TABLE	Compute image from table data
COPY/TT	Copy keys from table to table file
CREATE/COLUMN	Create new column in a table file
CREATE/TABLE	Create a table file
DELETE/COLUMN	Delete column from an element in a table file
EDIT/TABLE	Change value of entry in table file
MERGE/TABLE	Merge two table files
NAME/COLUMN	Insert a label name for a column
PRINT/TABLE	Print table
READ/TABLE	List elements of a table file
REGRESSION/POLYNOMIAL	Compute regression between column in table file
SELECT/TABLE	Select a subtable
SHOW/TABLE	List table directory
SORT/TABLE	Order a table file
STATISTICS/TABLE	Computes low order statistics for a column

Appendix B

Acknowledgements

B.1 General

It is of course never possible to adequately acknowledge the many small, but extremely useful, comments which the Data Management Division have received from many colleagues both within ESO and outside. Nevertheless, we would like to express our gratitude to those who have helped to make MIDAS what it is today and hopefully what it will be in the future. In addition, we would like to try to specifically acknowledge certain major contributions.

B.2 Packages and Commands

The fitting routines in MIDAS were developed in close collaboration with O. Richter and later upgraded by Ph. Defert. The INVENTORY programs were developed and written by A. Kruszewski during his extended visits to ESO. The multivariate statistical package has been developed in close collaboration with F. Murtagh. The package for 1-dimensional spectral reductions was designed and tested in collaboration with D. Baade and M. Rosa. The ROMAFOT package for crowded field photometry was developed by R. Buonanno, C. Buscema, C. Corsi, I. Ferraro, and G. Iannicolo at the Osservatorio Astronomico di Roma. The implementation of ROMAFOT in MIDAS was done in collaboration with R. Buonanno. M. Tapia and A. Moneti collaborated in the development of the IRSPEC reduction package. Marguerite Pierre developed modelling commands for interstellar absorption work and contributed to the Long Slit package. P. Stetson created a MIDAS compatible version of DAOPHOT-II.

The digital filter to remove cosmic ray events from single frames was contributed by P. Magain and M. Remy. Several routines used in the COMPUTE command for calculations of airmass, barycentric correction, ST, UT and Julian data were kindly made available by D. Gillet. The FILTER/ADAPTIVE command was kindly provided by G. Richter. The OPTOPUS context was implemented by A. Gemmo. The PISCO context was implemented by M. Schloetelburg and O. Stahl. The astrometry context ASTROMET was made available for MIDAS by O. Hainaut. The original code was written by R. West.

Significant contributions were also added in the application area. A number of important applications were made by ESO La Silla (or in close collaboration with them) such as the new Long Spectral Package and the XAlice graphic user interface for spectral analysis.The IRSPEC reduction was revised by E. Oliva, while an image restoration and co-addition application, based on ideas of L. Lucy, was added by R. Hook (ST-ECF). A Time Series Analysis context, which includes analysis of non-equally spaced data, was made by A. Schwarzenberg-Czerny. The Wavelet package was implemented by J. L. Starck and contains both general routines for wavelet transforms and special applications *e.g.* for deconvolution. Finally, the PEPSYS context was introduced by A.T. Young as the first application in a new context for calibrations of point-source photometry.

B.3 Libraries

B.3.1 AGL

The plotting package in MIDAS is based on the low level routines in the Astronet Graphic Library (AGL) which was developed and is maintained by the Italian ASTRONET. The implementation of the AGL library in MIDAS was done with the help of L. Fini.

B.3.2 IDI

In the early implementations of IDI routines for XWindows the Trieste Observatory (Mauro Pucillo, Paolo Santin, Fabio Pasian) provided a prototype for X10 which has been used for our further developments.

B.4 Manual

This manual has been typeset in TEX and LATEX using an extensive set of macros provided by H.-M. Adorf.

Appendix C

Site Specific Implementation

This appendix describes the site specific hardware setup and implementations used in ESO, Garching.

C.1 Hardware Setup

This section gives short description of the hardware configuration of the general ESO computer facilities in Garching used for image processing. All relevant information concerning ESO's computing facilities are collected and available via URL:

http://www.eso.org/it/

The main installation contains a number of UNIX workstations and Servers (most of which are SUN/SPARC compatible). A complete list can be found at URL:

http://www.eso.org/it/unix.html

under Computing Facilities for each division. In addition several X-terminal are available. All the systems are interconnected through a Local Area Network using TCP/IP protocols.

C.1.1 UNIX Workstations

A number of SUN/SPARC workstations and X-terminals for general MIDAS use are located in the basement. They run UNIX and are configured with the X11 window system. You can login by giving the **Userid/Password** allocated to you. When working in the ESO X11 environment, it is necessary to point the cursor on the X11 OSF/Motif window in question to get access to it. Several MIDAS sessions may be run on a workstation at the same time using the `parallel` option when starting MIDAS with the `inmidas` command. Different MIDAS unit numbers can also be given specified on the command line.

C.1.2 Printer and Plotter Queues

To output listings and hard copies of plots a number of devices can be used. These can be used through a number of system queues which spool the output to the appropriate devices. The majority of these printers are PostScript compatible. Printer are listed at URL http://www.eso.org/it/printers.html.

To print the log of the MIDAS session the `PRINT/LOG` command is used. This command will produce the output on printers, most of which are located in the User room.

Hardcopies of plots are made using the `COPY/GRAPH` and the `PLOT` commands (see Chapter 6). The first parameter of the `ASSIGN/GRAPH` command specifies the plotting queue/device to be used. The `help` command in MIDAS can be used to get an up to date list of output devices.

C.1.3 X11 Window systems

Image display windows can be created and deleted from MIDAS using the commands `CREATE/DISPLAY` and `DELETE/DISPLAY`, respectively. All MIDAS display commands will work on these window displays, some have to be implemented in software which makes them slow *e.g.* `ZOOM` and `SCROLL`. The cursor is implemented through the mouse on which the left button is **Enter** and the middle button (or right button on a 2 button mouse) is **Exit**. When the cursor is positioned on a feature it is also possible to use the **Return** key to read the cursor location. Some commands use a second cursor which is implemented through the arrow key on the keyboard.

C.1.4 Film Hardcopy

This Section describes the Film hardcopy devices available at ESO, Garching. Only a system for recording 35mm Colour Slides is offered.

Film print queues

They are implemented as standard UNIX print queues and can be accessed with the UNIX `lpr` command. Files written to a queue **must** be in the correct format (for further see URL http://www/eso.org/it/unix.html).

In the case of Colour Slide the MIDAS command `COPY/DISPLAY SLIDE` can be used. This command will both create the PostScript image file and submit it to the slide queue.

Processing of Film Hardcopies

Depending on the number of entries, images in the colour queue will be checked and activated once per week. Development of the films may take upto two weeks after which they will be send to the user. Thus, a turn-a-round time of approximately one month must be expected when using film hardcopy.

C.1.5 Tape I/O

Several tape drives (1/2 inch, QIC, 8mm Exabyte, DAT) are available on the UNIX system. They can be accessed from any UNIX workstation on the network. Which tapes are available and their locations are listed in URL http://www.eso.org/it/tapes.html. The step by step procedure is as follows:

1. Login on a UNIX workstation.
2. Run MIDAS with **inmidas**.
3. Mount your tape on a drive connected (see below). If you want to write on the tape remember to enable writing.
4. Use the `intape` and `outtape` commands to access the tape unit using the logical device name *e.g.* **tape0**. or the physical device name *e.g.* **/dev/nrst0** or **ws1:/dev/nrst0** for a tape unit on remote host **ws1**. Since the recording density for UNIX tape units is defined by their name, the device name will overrule the density given on the command line! Some 1/2 inch drives require the density to be set manually on the drive as well. During reading the tape drive will itself sense the tape density used. Thus, the generic tape name can normally be given when reading.
5. Dismount your tape from the drive so that others can use it.
6. Logout from MIDAS with **bye** and from the workstation.

Concerning the usage of tape media please note the folowing:

1. Tape drives can be selected from any host in the same domain. To get access to drives in other domains you need to have another account with the same name in the remote domain.
2. Tape devices have to be accessed via theirs system names;
3. New tapes are from factory write enable.
4. Tapes can be obtained from the helpdesk, room 220/1, 2nd floor.
5. The MIDAS error `Permision denied` occurs when the given tape is write protected (check remarks below) or when it is allocated to another user (use `deallocate` command)
6. Concerning 1/2 inch tapes:
   - Support is limited and may be discontinued soon.
   - For each 1/2 tape drive can be accessed in two densities: high (6250 bpi) and medium (1600 bpi). The mode that is used to access the data depends on the device name.
   - 1/2 inch magtape drives have to be set to enable remote density selection: with the drive OFF-LINE press the button DENSITY until LED "REMOTE DEN" comes on. The device name used with the command "INTAPE/MIDAS" will actually set the density.

- 1/2 inch magtape drives can *read* tapes of any density if the drive is set with "REMOTE DEN" on. That is, for reading there is no difference between, e.g. tape2, tape2h, and tape2m.
- 1/2 inch tapes are write enable when they are mounted with the write enable ring and the LED "WRITE ENBL" comes on.

7. Concerning 8mm tapes:
   - 8mm-High drives can also read 8mm-Low density tapes, but not vice versa.
   - The 8mm tape is write enable when the red switch in front of the tape is in "REC" position.
8. Concerning QIC tapes:
   - QIC-150 drives can only write on high density tapes DC600XTD or DC6150 however they can read QIC-24 low density tapes DC300XL or DC600A.
   - QIC-24 drives use only low density tapes DC300XL or DC600A.
   - QIC tapes are write protected when the round switch in the tape points to the "SAVE" position; write enable otherwise.
9. Concerning DAT tapes:
   - DDS drives use Sony 60/90/120 min.
   - DAT tapes are write enable when the swith in front of the tape closes the hole.
   - An up to date list of available tape drives is posted on the bulletin board in the user room.

C.2 Operating Systems

The main operating system used for image processing in ESO is UNIX. On the SUN workstation the X11 OSF/Motif window system is used to provide virtual terminals or displays. For those who are not familiar with UNIX systems this section tries to provide a very basic introduction. For more information, the interested user is referred to the various UNIX or SUN publications. A few copies of these manuals and other documentation are available through OSC Helpdesk. More information can be found on-line (URL http://www.eso.org/it/, item Unix/UNIX help for Users)

C.2.1 Login Procedures

The X11 window system will display a login window which has user identification and password files. If the screen is black you may have to move the mouse or press a key on the keyborad to exit the screen save mode. First enter your `userid` and then your password each terminated with the return key. This will start an automatic login procedure (see OSF/Motif manual for customising the window manager). In case you don't have an account on the systems you should contact Helpdesk (tel 6333).

C.3 Data Format Compatibility

The internal binary data formats of the VAX and SUN systems are different which makes it impossible to share data files (*e.g.* image or table). The SUN systems store data starting with the most significant bits (*i.e.* big endian) and use IEEE floating point format while VAX's are byte swapped and have a proprietary floating point format.

In order to exchange data files between these systems it is necessary to use a machine independent format *e.g.* FITS. For this reason and because internal MIDAS data structures may change, it is **strongly** recommended to save data in FITS format.

Appendix D

Release Notes

D.1 Current Status

This appendix contains on the following pages the Release Notes for the **98NOV**release of MIDAS. A listing of the MIDAS NEWS-file which gives an overview of the modifications and improvements of the system for the present release has been added.

D.2 Installation

The **98NOV** release of ESO–MIDAS can run on most UNIX systems, and is most likely to run under DEC/VMS 4.7 and DEC/OPEN-VMS 6.0 (or higher). If you are going to install ESO-MIDAS on your system you have retrieved a copy of this release through ftp. However, as have been announced, as of 1995 new applications are not tested automatically on VMS/Open VMS systems.

The monitor and low level interfaces are coded in C which means the portable MIDAS requires a C compiler for installation. Many applications are written in FORTRAN-77 and can therefore only be installed if either a FORTRAN-77 compiler or a FORTRAN-to-C convertion package (*e.g.* `f2c` for UNIX) is available. The installation of MIDAS has been certified with the public domain GNU ANSI-C compiler which can be obtained from the Free Software Foundation, and verified with the quality control tool Purify from Pure-Software.

The instructions for installation are given in separate documents for either VMS or UNIX systems which are included in the distribution kit. Read the appropriate one carefully and proceed as described (the procedure may have been modified compared with previous installations!). Basic knowledge of your local operating system (e.g. VMS or UNIX) is assumed and required.

One area may require special attention during the first installation. It relates to the image display interfaces that, for MIDAS, must conform to the standard IDI interfaces. IDI's are provided only for the X-Window system, Version 11. For other display devices a set of IDI routines has to be written.

For a few commands the NAG mathematical library is required. If you do not have this library MIDAS can still be installed, however, some commands may not be available. In future versions we will try to reduce our dependency on NAG. A list of programs using NAG and the routines are given in Section D.5.

Starting with the 89NOV release a separate installation of the lower level AGL library is no longer required. The library, i.e. those parts needed for the MIDAS plot facilities, is fully integrated within the MIDAS directory structure and will be generated as every other MIDAS subroutine library. For a full installation of the AGL library refer to the Astronet Documentation Facility, Trieste.

D.3 Software Modifications

The MIDAS release **98NOV** is the 12th official release of portable MIDAS. A number of new applications have been added to this version. Given below is a overview of the changes and improvements compared with the previous release:

1. ...
2. ...
3. Several new commands have been included in the .... package related to spectral analysis:
    - ...
    - ...

For a more complete list of all updates/additions use the MIDAS command `HELP [NEWS]` after having installed this release. A hardcopy of it can be obtained via `PRINT/HELP [NEWS]`.

D.4 Manual Updates

Several sections of the MIDAS User's Manual have been updated in this release. A list of the parts to be replaced is given below :

Volume A, Titlepage		to be replaced
Volume A, Chapter 1	Introduction	to be replaced
Volume A, Chapter 3	Monitor and Syntax	to be replaced
Volume A, Chapter 3	Graphics and Display	to be replaced
Volume A, Appendix D	Release Notes	to be replaced
Volume B, Titlepage		to be replaced
Volume B, Chapter 12	Time Series Analysis	to be replaced

D.5 Use of NAG Library

The NAG mathematical library is still used in a few MIDAS commands. A list of these programs and routines are given below:

Program	Package	NAG Routines
fitimag	Fit	e04fdf, e04fcf, e04gcf, e04hev,
		e04gbf, e04gef, e04gdf, e04ycf,
		e04jaf, e04hbf, e04jbf, e04kaf,
		e04hcf, e04kbf, e04kcf, e04kdf
genran	General	g05cbf, g05ddf, g05daf, g05dbf,
		g05def, g05edf, g05eyf, g05ecf,
		g05dff
echripp1	Echelle	e04gdf

A set of dummy routines are provided for sites that do not have a NAG library. This implementation enables sites to use the FIT package even without the NAG library (in this case, only the Newton-Raphson method is supported).