MIDAS
Users Guide

Volume A

System

MIDAS Release 92NOV

Image Processing Group
European Southern Observatory
Karl–Schwarzschild–Straße 2
D–8046 Garching bei München
Federal Republic of Germany

Section	Title	Date
Chapter 1	Introduction	1-November-1991
Chapter 2	Cook-Book	1-November-1991
Chapter 3	Monitor & Syntax	1-November-1992
Chapter 4	Data Structures	NA
Chapter 5	Table File System	1-November-1992
Chapter 6	Graphics & Image Display	1-November-1992
Chapter 7	Data Exchange Format	1-November-1991
Chapter 8	Fitting of Data	15-January-1988
Appendix A	Detailed Command Description	1-November-1992
Appendix B	Command Summary	1-November-1992
Appendix C	Acknowledgements	1-November-1992
Appendix D	Site Specific Implementation	1-November-1992
Appendix E	Release Notes	1-November-1992

Contents

List of Figures

List of Tables

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 two volumes:

Volume A: describes the basic MIDAS system with all general purpose facilities such as MIDAS Control Language, all available commands, data input/output (including plotting and image display), table system (MIDAS Data Base). Site specific features are given in an appendix.

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

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.

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 two sets of general interfaces for application programs to data structures, namely: a) the "Standard Interfaces" for general I/O and image access, and b) the "Table Interfaces" for access to table structures. 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

The MIDAS system is available, free of charge, to all non–profit research institutes, whereas other organisations or companies may be charged a nominal fee to cover distribution. Institutes interested in using MIDAS must sign a User Agreement before distribution material can be shipped. The necessary forms can be obtained by contacting the Image Processing Group at ESO/Munich. MIDAS is distributed in source code copyrighted by ESO with all rights reserved. Institutes receiving the MIDAS system are not allowed to redistribute it to other sites without explicit written permission from ESO. The use of the MIDAS system for data reduction should be properly acknowledged in papers and publications. It is recommended to refer to the specific MIDAS version used, e.g. 91NOV, in the acknowledgement.

The availability of new releases is announced through electronic mail. Requests can be submitted either on special MIDAS Request Forms or through e-mail quoting the user agreement number. Currently, MIDAS has one yearly release in November (*e.g.* the release in 1992 is will be named 92NOV). The current system at ESO/Munich is frozen several months prior to the official release. The official release is based on this version, which is extensively tested both at ESO and at a number of β-test sites. Problems detected during these tests are corrected in the official release version which is then given free for distribution. The release is is available on a wide range of tape media and additionally, it can be obtained through an ftp account.

Updates of the manual and other documentation take somewhat longer to prepare and print and will be sent to all sites which receive a new MIDAS release. Updates are normally not sent to individual users. Copies and updates of documentation can be obtained by writing to the Image Processing Group at ESO/Munich.

1.4 Support

The MIDAS system is supported in a variety of ways. If people encounter problems which cannot be solved locally (*e.g.* through the manual) they can use the MIDAS **Hot–Line** service. This service will provide answers to MIDAS related questions received through the following list of electronic mail and telex addresses:

- `earn/bitnet: midas@dgaeso51`
- `uucp: midas@eso.uucp`
- `internet: midas@eso.org`
- `span: eso::midas`

- `Telefax: +49 89 32006480 (attn.: MIDAS HOT-LINE)`
- `Telex: 528 282 22 eo d (attn.: MIDAS HOT-LINE)`

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 Hot–Line**.

In urgent cases, users can use a special MIDAS Support telephone service at ESO on the number +49-89-32006-456. This line is connected to the MIDAS Users Support which is able directly to answer questions concerning MIDAS or investigate the problem in more complicated cases. Although this telephone service is available we prefer that questions or requests are submitted in writting via the MIDAS **Hot–Line**. This makes it easier for us to process the requests properly. A database with problem reports and answers is available for interogation using the STARCAT utility at ESO. General information concerning the MIDAS system should be addressed to Image Processing Group, European Southern Observatory, Karl-Schwarzschild-Straße 2, D-8046 Garching bei München (attn: Resy de Ruijsscher).

Besides these support services, MIDAS User meetings are arranged to discuss status and developments inside and outside of ESO, normally in connection with the ESO/ST-ECF Data Analysis Workshop. Further, a newsletter, the ESO-MIDAS Courier, is issued twice a year.

1.5 Requirements for Running MIDAS

MIDAS can run on computers with either VAX/VMS[2] or UNIX[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 SPAECstations, HP 700 series, IBM RS/6000 systems. The availability for a specific system can be checked by asking the MIDAS Hot–line.

- 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

[2]Trademarks of Digital Equipment Corporation
[3]Trademark of AT& T

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.

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.

- 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.
- Graphics: 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.* Tektronix 4010/4015, X Window System, version 11 and PostScript (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. IDI-routines are available for DeAnza IP 8500 and X Window. X Window System, version 11, will be supported as the general interface to workstations. 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 Image Processing Group.

1.5.2 Software

- System: VAX/VMS or UNIX operating systems.
- Compilers: FORTRAN-77 and C compilers.
- 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

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

- 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.

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
- 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

The above documents can be obtained by contacting the Image Processing Group (e.g. via the HOT-LINE).

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 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 as well as a printed version of the help text (see the Appendices).

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 primitive commands
- MIDAS application commands

- procedure control commands

The MIDAS commands are listed in alphabetical order and explained in detail in Appendix A. 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 procedure control commands are described in Chapter 3.

2.3 Getting Started

The first thing to do is to login to the computer which you would like to use for your data reductions. The detailed procedure for getting permission to use the computer and getting allocated disk space for your data reductions can be found in Appendix C. Assuming that you have succeeded in logging into the computer, the following subsections describe typical use of MIDAS.

When you have logged in you should check that you have sufficient disk space in the directory in which you are working. For reductions of images an area of the order of 50 Mbytes would be adequate while for spectra reductions and analysis of final results less space is needed. Now you are ready to start the MIDAS system:

- Type `INMIDAS` on a VMS system or `$inmidas` on a UNIX system. This will initiate the MIDAS environment and the terminal should respond with:

  ```
  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 **??**. You can find the full explanation of the individual commands 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*.

- In some installations a number of tutorial commands are available. (see Table 2.1) They provide an illustration of different parts of the system.

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
```

```
$ 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
                                                         !one of the 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
```

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` (for VMS systems) or `$gomidas` (for UNIX systems).

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

2.3.3 Executing System Commands

It is possible to execute commands of the operating system inside MIDAS. 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.

2.3.4 Some Useful Commands

In table 2.2 you will find a list of some of the most frequently used MIDAS commands. Refer to the detailed command description or the on–line HELP for more details

Command	Description
HELP *command*	display help for *command*
HELP/QUAL *qualifier*	help for all commands with the given *qualifier* such as IMAGE, TABLE, CATALOGUE, etc.
SET/CATAL	enable cataloging
CREATE/ICAT	create a catalogue of image files
READ/ICAT	list the image frames available
LOAD/IMAGE *frame*	load and display *frame* on the image display
MODIFY/LUT	interactively change the look–up table
COMPUTE/IMAGE	perform image arithmetic
EXTRACT/TRACE	interactively extract a line from an image
PLOT/TRACE	plot the extracted line
ZOOM	zoom an image on the cursor
TUTORIAL/*tutorial*	execute one of the existing *tutorial*s

Table 2.2: List of Often Used Commands

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 to distinguish them from programs written in a language like FORTRAN-77 or C.
  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 the same directory where your data files are stored.

Type `SETMIDAS` on a VMS system or `setmidas` on a UNIX system if you want to change the defaults for the mode and the MIDAS work directory.
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.

To start MIDAS, type `INMIDAS` on a VMS system or `inmidas` on a UNIX system. This will initialise 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 private MIDAS procedures.
- 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`
- Depending upon how your MIDAS system manager installed MIDAS you will be asked to enter the identification of a MIDAS unit as a two-character string.

Usually a two-character string beginning with `X`, `Y` or `Z` defines a unit with no image display capabilities and `A0`, `A1`, ... or `00`, `01`, ..., `99` indicate a unit with image display functionality.
So `A1`, `XA`, `YF` or `ZS` will all be valid units. Upper and lower case characters are treated alike. If you work in *Parallel* mode you have to use different MIDAS units for each session.
The MIDAS unit is appended to the names of all MIDAS internal files which are created at startup time.

Now the terminal screen is cleared and the current MIDAS version as well as the computer you are on and operating system used are displayed. Finally the prompt string

```
Midas 001>
```

appears on the terminal screen. You are now ready to execute any of the MIDAS commands that are currently available.

The internal MIDAS files all reside in the MIDAS working directory, but the data files are taken from the current default directory unless the complete file specification is given in the data file name.
To switch from one data directory to another you have to terminate MIDAS via `BYE`, change the default directory `SET DEF` (VMS) or `cd` (UNIX) and `GOMIDAS` (VMS) or `gomidas` (UNIX) to continue with MIDAS in the new directory.

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> $directory` (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).

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 stepsize 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, practically all 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` .

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.

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. The only restriction is that the `first` *character of the names of MIDAS data frames must be an alphabetical character (*`a` *–* `z`*). All following characters may be any character except '+', '-', '*', '^', '(' and ')', furthermore, the character '/' is invalid in Unix and '$' is invalid in VMS.*
Thus `abcdef.image`*,* `flat_sky.bdf`*,* `k12324_fit.zzz` *are valid names for MIDAS data files, whereas* `1mariposa.ima`*,* `a+b.bdf` *and* `(xyztest.bdf` *are not. 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 C 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 currently not supported.
Each descriptor also has a name (max. 15 chars.) and a length. 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" while you are editing it.

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/DESC 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/DESC imgfile Descname/C/1/7 "       "
```

would fill `Descname` with 7 blanks.

```
WRITE/DESC 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/DESC 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/DESCRIPTOR` defaults the first parameter to an image and appends the file type `.bdf` if none is given by the user.

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`.

3.3.2 Specifying Keywords

As is the case for descriptors, keywords also have a name (max. 8 chars.), a type and a length. 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*. Mixed types are currently not supported.
In order to write a value to a keyword, the same format as for descriptors is used.

```
WRITE/KEY INPUTC/C/1/8 AKeyword
```

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

```
WRITE/KEY AZTEC/I/1/2 17,-22
```

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

Some of the commands dealing with keywords are:
`READ/KEY, WRITE/KEY, SHOW/KEY, DELETE/KEY, COPY/KEY, COMPUTE/KEY`.
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 values 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:

```
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 labelled 'MAGNITUDE' and value 20.0 in the reference column (this reference column must have been defined before via the SET/REF 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`.

Some of the commands dealing with tables are: `READ/TABLE, WRITE/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 arcseconds (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]
```

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

3.4 Command Syntax

After start–up, the MIDAS monitor prompts you to interact with MIDAS by entering command–lines on the terminal.
To enter a command-line over more than one terminal/window line, use the continuation character, a minus sign (`-`), as the last character of a line (the total command line is limited to 256 characters). If you want to enter more than one command on a single line, separate the commands with a semicolon and space (`; `).
Each 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 command parameters (max. 8) 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)
  - cursor arrays
  - overlay tables
    (internally these structures are stored in MIDAS table files)

All user input and output from the MIDAS commands is recorded in an ASCII logfile which serves as a hardcopy of a MIDAS session.
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 (may be 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. Nearly all commands need a qualifier, but there is only one qualifier per command (e.g. `comm/qual1/qual2` is unsupported in MIDAS). Command and qualifier are separated by a `'/'` (slash). In case you omit the qualifier, the default qualifier of that command is used by MIDAS.
The default qualifier of a MIDAS command may be displayed via `SHOW/COMMAND command`. 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 a 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.

Most 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` With the command CREATE/DEFAULTS you can change the preset defaults for each MIDAS command.
To abort a MIDAS command, use Ctrl/C (sometimes you also have to hit Return), which will return control to you. The exception are commands which interact with a display/graphics window in the X11–Environment where you run the risk of losing the MIDAS display/graphics window server, which must then be re–initialized, see chapter 6 for details.

3.4.1 Command Recalling

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/key 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/com
```

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 lowercase.
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/KEY 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).

Note

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.* `$ RENAME file.typ.7 lola.bdf`*.*
In UNIX the repetition of tokens may cause trouble. Consider the following:
`Midas 123> load/ima 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`

3.4.2 Command Line Editing

The commands in the internal command buffer may also be edited. To edit such a 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. 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).
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'`. `CLEAR/BUFFER` empties the complete 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 command no. 999).

3.4.3 Command Line Suspension

If, while entering a command, you realise 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! All MIDAS commands are also described in detail in the appendices of this manual.
The command `HELP/SUBJECT` will provide some cross–references. If you don't know under which command name a certain function is implemented in MIDAS, enter `HELP/SUBJECT` *function*. If on–line information exists for that function, all related MIDAS commands will be listed.
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.
If you work in an X-Window environment start up `XHelp`, the graphical user interface to the MIDAS Help facility, by entering the MIDAS command `CREATE/GUI HELP`.
Some of the other `HELP` features are given in the following table.

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/KEY *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

Table 3.1: Help Features

There is a set of MIDAS command procedures that fulfill various functions that are very application specific. Therefore, it did not seem worthwhile to implement them as general MIDAS commands. Use HELP/APPLIC to get information about these procedures.

Normally, the HELP text is not written to the MIDAS log file (to save space) but if you wish to include this, you may do so by setting keyword LOG(3) to 1 via the MIDAS command WRITE/KEY LOG/I/3/1 1 . The MIDAS command WRITE/KEY LOG/I/3/1 0 will disable the logging of HELP. To print out the help text, use PRINT/HELP.

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...

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` 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 commands `LOG/...` manipulate that file. 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 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 MIDAS Command Language 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 10 levels deep).

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 additional *Command Language* commands cannot be used interactively and are listed below:

`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`
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 stepsize
  ...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, but `ELSEIF` is one word!)

`INQUIRE/KEY key prompt-string`
Demand value for *key* from the user

`label:`
Declare a *label*,e.g. `HOME:`

`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

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 the current directory or MID_WORK:),
or in APP_PROC:, or in STD_PROC:, or in CON_PROC:, respectively

`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/KEY reskey = expression`
Evaluate an algebraic *expression* involving keys and constants, store result in *reskey*

`SET/FORMAT I-format E-format`
Define *format*s 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

The following "immediate" commands can be executed in order to set values of various data structures:

`key = expression`	Short form of COMPUTE/KEY key = expression
`frame,descr = value`	Set descr of frame to value
`frame[x,y,z] = value`	Set pixel with coordinates (x,y,z) of frame to value
`table,column,row = value`	Set table element to value

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.

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 command 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
@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.

The maximum size of a single parameter is 80 characters, but all parameters together may not exceed 256 characters (which is the maximum size of a command line). The size 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 10 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 ({ and }) (cf. section 3.6.2):

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

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

```
READ/KEY OUTPUTC
@@ test ESO-Garching
WRITE/KEY 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. section 3.8.1).$

Note

You can also use apostrophes to indicate symbol substitutions (e.g. 'P1'). However, it is preferable to use { and } instead, since it 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/PAR P1 999 NUMBER "Enter first input number:  " 22,1024
WRITE/KEY 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/PAR` 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 it is checked that it is a valid MIDAS file name (first character must be an alphabetic character). Also, DEFINE/PARAMETER translates catalog numbers, e.g. #27 to the corresponding file name.
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. For an example see section 3.6.6.

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/PAR P1 999 N/CONT "Enter first input number:"
IF PARSTAT(1) .EQ. 1 -
  WRITE/KEY INPUTI/I/7/1 {P1}        !store contents of P1 in INPUTI(7)
```

then @@ exa3 KB will not yield any error.
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!

If we modify exa3.prg to:

```
!+
!  Example 4, MIDAS procedure exa4.prg
!+
CROSSREF IN_FILE OUT_FILE METHOD ALPHA
DEFINE/PAR P1 ?   IMA "Enter name of input file:  "
DEFINE/PAR P2 ?   IMA "Enter name of result file:  "
DEFINE/PAR P3 ?   C  "Enter method:  "
DEFINE/PAR P4 999 NUM "Enter alpha value:  " 22,1024
WRITE/KEY 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/KEY` 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/PAR P1 ?   IMA "Enter name of input file:  "
DEFINE/PAR P2 ?   IMA "Enter name of result file:  "
WRITE/KEY IN_B " " all                 ! fill keyword IN_B with blanks
INQUIRE/KEYW IN_B "Which filter, enter LOW or HIGH: "
```

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/KEY` command (will be 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 is used to distinguish among keywords, descriptors, pixel values and table elements:

`{star}`	refers to the value stored in the keyword "star"
`{galaxy,disk}`	refers to the contents of descriptor "disk" of frame "galaxy.bdf"
`{galaxy[x,y]}`	refers to the value of the image pixel at coordinate x,y of the 2–dimensional frame "galaxy.bdf"
`{dust,:particles,7}`	refers to the element of the table "dust.tbl" in column labelled ":particles" and row 7
`{dust,#2,77}`	refers to 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)`. 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).

Let us look at an example of this:

```
!+
!  Example 6, MIDAS procedure exa6.prg
!+
DEFINE/PAR P1 ?  N "Enter alpha value:  " -88.5,912.4
DEFINE/PAR P2 ?  N "Enter loop_count:  " 1,999
WRITE/KEY VAR/R/1/1 0.                                  ! init key VAR
VAR = {P1} * 3.3                      ! set VAR to 3.3 * (contents of P1)
!
LOOP:                                                   ! declare label LOOP
VAR = 1.+VAR                                          ! set VAR = 1.0 + VAR
WRITE/OUT NAXIS = {myframe,naxis}
WRITE/OUT {myframe[@10,@20]}
WRITE/OUT {mytable,:DEC,@7}
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:

```
NAXIS = 0002
3.456000E+00
-7.030000E+01
NAXIS = 0002
3.456000E+00
-7.030000E+01
```

assuming, that the image frame `myframe.bdf` is a 2–dimensional frame, the pixel value at pixel coordinates `[10,20]` is 3.456 and the element at column labelled :DEC and row 7 of table `mytable.tbl` is -70.3 .

Any algebraic expression using the operators +, −, ∗, / and parentheses (,) is supported by the command `COMPUTE/KEY` and its short form `'key = expression'`.

Note

For character keywords `COMPUTE/KEY` only supports character concatenation ('//'). If you want to write a character string into a character keyword, use `WRITE/KEY` instead. Therefore, if we had written `VAR = P1 * 3.3` instead of `VAR =` {`P1`} `* 3.3` in the procedure `exa5.prg`, MIDAS would have protested because no multiplication is permitted for character keywords.

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 may be controlled via the command `SET/FORMAT I-format` for integer symbols and `SET/FORMAT E-format` (or `F-format`) for real and double precision symbols. Integer symbols are then encoded via I–format (with leading zeroes not suppressed) and real or double precision symbols as E–format or F–format:

```
!+
!  Example 7, MIDAS procedure exa7.prg
!+
WRITE/KEY INPUTI 12                          ! set INPUTI(1) to 12
WRITE/KEY INPUTR 12.345                  ! set INPUTR(1) to 12.345
WRITE/KEY INPUTD 1234.5678           ! set INPUTD(1) to 1234.5678
WRITE/OUT {inputi(1)} {inputr(1)} {inputd(1)}
SET/FORMAT I2                    ! use format I2.2 for integer symbols
SET/FORMAT E12.8           ! use format E12.8 for real/double symbols
WRITE/OUT {inputi(1)} {inputr(1)} {inputd(1)}
SET/FORMAT I5                    ! use format I5.5 for integer symbols
SET/FORMAT F12.4           ! use format F12.4 for real/double symbols
WRITE/OUT {inputi(1)} {inputr(1)} {inputd(1)}
```

The command `@@ exa7` will yield:

```
0012 1.23450E+01 1.23457E+03                    default is I4 and E15.5
12 1.23450003E+01 1.23456780E+03                    uses I2 and E12.8
00012 12.3450 1234.5678                             uses I5 and F12.4
```

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

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/PAR 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.

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/KEY 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.
Note, that `ENDDO` has to be written as one word!

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/PAR P1 ?  IMA "Enter root_name for input frames:  "
DEFINE/PAR P2 ?  IMA "Enter root_name for output frames:  "
SET/FORMAT I4                                          ! we need 4 digits
WRITE/KEY N/I/1/1 0
WRITE/KEY NN/I/1/2 0,0
!
DO N = 1 50                                    ! default increment is 1
  NN(1) = 2*N
  NN(2) = NN(1)-1
  COMPUTE/IMA {P2}{N} = {P1}{NN(2)}+{P1}{NN(1)}            ! sum up
  LOAD/IMA {P2}{N}                             ! display the result frame
ENDDO
```

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

3.6.4 Local Keys

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/KEY N/I/1/1 0
DO N = 1 10
  @@ test
ENDDO
```

```
!+
!  MIDAS procedure test.prg
!+
WRITE/KEY N/I/1/1 0
DO N = 1 12
  WRITE/KEY 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* keys 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 and in 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 keys 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/KEY INPUTI/I/12/1 {N}
ENDDO
```

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

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.
Note that the ELSEIF must be given as one word; ELSE IF will not work (the same holds for ENDIF)!
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/KEY INPUTC beer
WRITE/KEY OUTPUTC wine
WRITE/KEY 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 lowercase 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/PAR P1 ?  N "Enter number:  "
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/PAR P1 ?  N "Enter number:  "
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/PAR P1 ?  N "Enter number:  "
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/PAR P6 + NUMBER "Enter first input number:  "
IF P6(1:1) .EQ. "+" THEN
 WRITE/KEY INPUTC NONE          ! no P6 entered, set INPUTC accordingly
ELSE
 WRITE/KEY INPUTI/I/7/1 {P6}        ! store contents of P6 in INPUTI(7)
 WRITE/KEY 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/PAR command again, because testing "+" against a numerical interval would lead to an error:

```
!+
!  Example 14b, MIDAS procedure exa14b.prg
!+
DEFINE/PAR P6 + NUMBER "Enter first input number:  "
IF P6(1:1) .EQ. "+" THEN
 WRITE/KEY INPUTC NONE          ! no P6 entered, set INPUTC accordingly
ELSE
 DEFINE/PAR P6 + NUMBER "Enter first input number:  " 22,1024
 WRITE/KEY INPUTI/I/7/1 {P6}        ! store contents of P6 in INPUTI(7)
 WRITE/KEY 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:  "
!
!  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 in the context of `COMPUTE/KEY`. The existing functions are listed in the following table (on the next page). 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.
The table on the following page contains all functions available.
On–line help for these functions is available via `HELP COMPUTE/KEY`.

As an example we want to check whether a table file name has been input with or without the file extension `.tbl` ; if not given we append the type inside the procedure:

```
!+
!  Example 16, MIDAS procedure exa16.prg
!+
DEFINE/LOCAL INDX/I/1/1 0
COMPUTE/KEY INDX = M$INDEX(P1,".tbl")          ! test, if type of a table
                                                     ! already given
IF INDX .EQ. 0 THEN
  WRITE/KEY IN_A {P1}.tbl                        ! if not, append type
ELSE
  WRITE/KEY IN_A {P1}                  ! if yes, no need to append type
ENDIF
```

The MIDAS command `@@ exa16 mytable` as well as `@@ exa16 mytable.tbl` will both fill the character keyword IN_A with the string 'mytable.tbl'.
The MIDAS procedure verify3 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).

Command	Description
M$ABS(*arg1*)	returns the absolute value of integer/real/dou ble *arg1* as integer/real/double
M$EXIST(*arg1*)	returns 1 or 0, depending on whether the file *arg1* exists or not
M$EXISTD(*arg1*,*arg2*)	returns 1 or 0, depending on whether the descriptor *arg2* of frame *arg1* exists or not
M$EXISTK(*arg1*)	returns 1 or 0, depending on whether keyword *arg1* exists or not
M$INDEX(*arg1*,*arg2*)	returns index of string *arg2* in string *arg1* as integer value (same as function INDEX of FORTRAN 77)
M$LEN(*arg1*)	returns length of character string *arg1* (omitting trailing blanks)
M$NINT(*arg1*)	returns nearest integer of real/double *ar g1*
M$SYMBOL(*arg1*)	returns the translation of DCL symbol (VMS) or environment variable (Unix) *arg1* 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$TSTNO(*arg1*)	returns 1 or 0 depending on whether string *arg1* is a number or not
M$TIME()	returns the current date and time as an ASCII string of 24 characters
M$SECS()	returns the current time as no. of seconds elapsed since 1st Jan. 1970 (as an integer)
M$LN(arg1)	returns natural logarithm of real/double *arg1*
M$LOG(arg1)	returns logarithm (base 10) of real/double ıt arg1
M$EXP(arg1)	returns exponential of real/double *arg1* (base e)
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 d egrees
M$ACOS(arg1)	returns arccosine of real/double *arg1* in degrees
M$ATAN(arg1)	returns arctangent of real/double *arg1* i n degrees
M$SQRT(arg1)	returns square root of real/double *arg1*

Table 3.2: Special Functions available for operations on keys

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, that is 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/PAR P1 ?  IMA "Enter input frame:  "
DEFINE/PAR P2 ?  IMA "Enter output frame:  "
WRITE/KEY IN_A {P1}
DEFINE/LOCAL MYRESULT/C/1/80 {P2}
RUN tricky.exe
PAUSE
!
INQUIRE/KEY 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/PAR P1 11 NUMBER "Enter input number:  "
WRITE/OUT "Parameter 1 = {P1}"
!
ENTRY 2
DEFINE/PAR P1 new C "Enter input:  "
WRITE/OUT "Parameter 1 = {P1}"
!
ENTRY third
DEFINE/PAR P1 spiral IMA "Enter input image:  "
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/PAR P1 000 C "Enter control flags for entries:  "
DEFINE/PAR P2 sombrero IMA "Enter image to work with:  "
!
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
!
CREA/IMA {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 anytime 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:

`cloud`	Model for absorption lines
`daophot`	Object detection and classification, the DAOPHOT-2 package
`echelle`	Reduction of echelle spectra
`exsas (*)`	Analysis of X-ray data from the ROSAT satellite
`iue (*)`	IUE-tape reader
`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
`long`	Reduction of long slit spectra
`optopus`	Package to prepare observations with the fibre Optopus facility at La Silla
`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

For example, the command `SET/CONTEXT invent` will activate the commands related to the INVENTORY photometric package.
`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:  "
!
WRITE/KEY IN_A {P1}
WRITE/KEY 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/IMA res=perrito in=lobo`.

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.

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`.
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 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 explicitely 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 (this command currently only implemented for image catalogs).

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/KEY`. The keyword CATAL is not modified.
However, if P1 contains the name of a catalog of the form *file.cat*, this catalog (which has to contain images) 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; 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       930115
!+
CREA/COM RK READ/KEY                                !define abbreviations
CREA/COM WK WRITE/KEY
CREA/COM RD READ/DESCR
CREA/COM WD WRITE/DESCR
CREATE/COMM XH CREATE/GUI HELP
CREA/COM SMOOTH/SPECIAL @@ mysmooth            !define a new command
!
CREATE/DEF CREATE/GRAPH ? 400,800              !size for graphic window
CREATE/DEF CREATE/DISP ? 600,600,400,400 !size+loc for display window
!
SET/MIDAS_SYS edit=vi user=user
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`. 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 us=EXPERT`.
The level NOVICE is the default level assigned to you when getting into MIDAS.
The following functions of the MIDAS system are affected by the user-level:

`HELP` facility:
Help text for NOVICE- and USER- level MIDAS users is limited to the screen size of the terminal, i.e. you have to hit `Return` to scroll through the help text, if it is longer than single screen size.
For EXPERT users the help text is typed out completely.

`ERROR` reporting:
For NOVICE- and USER- level MIDAS users the full error text is displayed.
For EXPERT users only one-line error messages are displayed.

`CREATE/COMMAND` command:
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 ↑, ↓, ←, or →.

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 Appendix A 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
12345678901234567890123456789

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 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 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.;
- 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 subsection 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 plot output on a workstation running under X-window 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*.

Assignment and assignment change of the default device 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 regurlarly. Finally, if you run MIDAS from a standalone (non-graphic) terminal, an assignment to the NULL device has to be made. 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.

6.1.3 General Commands

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

Device Identification	AGL_type
AGFA postscript laser printers	pscript
Apple Laser Writer	pscript
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
Null device	null
Raster devices	raster
Tektronix 4010	tkg.t4010
Tektronix 4014	tkg.t4014
Tektronix 4100 series	tkg.t4100
QMS laser printer	tkg.qms
Versatec V-80	raster
X-Window	idi

Table 6.1: Supported Devices

and some general plot functions. These commands are:

`CREATE/GRAPHIC` – create 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

`PLOT/AXES` – plot a box with tickmarks, ticklabels and axes labels
`OVERPLOT/AXES` – overplot a box with tickmarks, ticklabels and axes labels

`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

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, line thickness, symbol type and size, 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 is 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 lenghts) 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 fonts`.

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

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 reinitialisation 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.

`PLOT/AXES` and `OVERPLOT/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 tickmark 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.

LABEL/GRAPHIC, OVERPLOT/LINE and OVERPLOT/SYMBOL

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.

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:

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

```
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.

6.1.4 Main 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 MIDAS will extend the existing plot information without erasing the results of previous plot commands.

In general all plot commands in this section 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. The main commands for plotting data structures are:

`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

\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

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

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

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. At this stage, names of plotfiles 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.

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 prespecified (with the ASSIGN/GRAP command) may lead to incorrect scales. In case the prespecified 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.0`). Below, follows a simple LATEXexample that shows how it works.
Example:

```
MIDAS 005> ASSIGN/GRAPH laser NOSPOOL
MIDAS 006> PLOT/ROW image [@100,@150:@150,@250] 20.0,20.0
MIDAS 007> OVERPLOT/TABLE table #1
MIDAS 008> COPY/GRAPH laser
MIDAS 009> $copy pscrplot.0 latexplot.ps
```

In your directory we now have a PostScript file latexplot.0, containing the complete plot information written by the commands 006 and 007. Now, 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=latexplot.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 shows a summary 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/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.
MIDAS supports peripheral displays like e.g. the Gould IP8000 (former DeAnza) series and XWindow displays. We describe here the IP8500 display specifically and a generic XWindow display.

6.2.1 IP8500 display

Each image memory or "channel" has independent scroll and zoom capabilities as well as an intensity transformation table which can be used like a colour look–up table and can also be used to change the output values that are fed to the look–up tables in the video output controller. This allows fast displays of log or histogram equalized images without having to reload the entire image.

One image channel is designated as the graphics (or overlay) channel. This also has its own zoom and scroll capabilities. In addition, the colour of the overlay can be selected. In MIDAS the last available channel is always used as the graphics channel. Also an alphanumeric memory is associated with the image display station.

The video output controller (VOC) selects which image memory is to be displayed on which colour channel as well as performing the task of overlaying the graphics plane. It also takes care of integrating the cursor and alphanumeric data into the video output. Finally, the VOC supports *Split Screen* mode where parts of 2 or 4 image channels are displayed together on the screen.

Using Image Memories

Several image memories are associated with each image display station. Thus it is possible to have several images loaded in the image display at the same time and to switch quickly from one channel to the other. Images can be loaded into any of the image memories. They are referenced by numbers 0, 1, 2, or 3. A typical command sequence would be:

```
LOAD/IMAGE galaxy 0                    ! load an image in channel 0
DISPLAY/CHANNEL 0                      ! display memory channel 0
```

The most important commands associated with handling the image memories are:

`LOAD/IMAGE` - *load an image into image memory*
`GET/IMAGE` - *read an image from the image memory*
`GET/CURSOR` - *read pixel values of displayed image*
`DISPLAY/CHANNEL` - *display the image in the selected channel*
`CLEAR/CHANNEL` - *erase contents of an image channel*
`SET/SPLIT` - *enable split screen*
`CLEAR/SPLIT` - *disable split screen*
`ZOOM/CHANNEL` - *zoom in integer steps 1 to 8*
`SCROLL/CHANNEL` - *scroll the image*
`BLINK` - *blink between two different image memories*
`SHOW/CHANNEL` - *show status of image channel*

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 pseudo–colour mode. Commands exist to load LUTs into the image display, to modify LUTs interactively and to read back LUTs from the image display. Interactive modification is done via the joystick or trackball device.

Some of the existing LUTs are:

```
backgr
color
heat
light
pastel
pseudo1, pseudo2
rainbow, rainbow1 ... rainbow4
random, random1 ... random4
smooth
staircase
stairs8
```

Use the command `TUTORIAL/LUT` to see what some of the available LUTs actually look like and how to modify the LUTs interactively.
The main commands available for handling 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*

Intensity Transformation Tables

The intensity transformation tables (ITTs) come between the image memories and the look–up tables. Using ITTs in the display mode allows special modifications to be applied to the displayed data without modifying the look–up tables or the data in the image memories. Interactive modification is done via the joystick or trackball device.
The main commands which control the ITT functions are:

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

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.

Using the Cursors

Each image display has two independent cursors available. In addition each cursor shape can be defined. The variety of possibilities available for various cursor forms and types defies a simple explanation here that would make much sense. The interested user is referred to the command `TUTORIAL/CURSOR` which gives a demonstration of the possible

cursor shapes.
The cursor(s) are controlled interactively by a special device e.g. tracker ball, joystick or mouse. Detailed information on how to operate the cursors can be found in Appendix C. In general there is a switch for each cursor to define its status ON/OFF. When a cursor is active (ON) its position can be read by pressing the Enter key. To exit an interactive cursor command one normally has to switch the cursors off and press the Enter key.
The commands associated with cursor control operations are:

SET/CURSOR - *enable selected cursor*
LOAD/CURSOR - *load a programmable cursor*
GET/CURSOR - *read cursor positions*
CLEAR/CURSOR - *disable cursors*

Graphics

Each user has a graphics (or overlay) channel associated with the image display. The currently available commands associated with the use of the overlay channel are listed below:

SET/OVERLAY - *enable the overlay of graphics on top of the image*
CLEAR/OVERLAY - *disable the overlay memory*
LOAD/OVERLAY - *load a LUT for the graphic channel (for experts only!!)*
SCROLL/OVERLAY - *scroll the graphics channel with the image channel*
ZOOM/OVERLAY - *zoom the overlay with the image channel*
DRAW/... - *draw a geometric shape like CIRCLE, RECTANGLE, etc. in the overlay plane*
SET/CHANNEL - *designate a channel as image or graphics*

Turning off the overlay/graphics via CLEAR/OVERLAY only disables the overlaying of the graphics. To really get rid of what is in the overlay channel you must use CLEAR/CHANNEL OVERLAY.
Furthermore, due to the internal hardware of the IP8500, disabling the overlay will also turn off the visibility of the cursors!

Alphanumerics

The alphanumerics memory is divided up into 22 lines of 80 characters. (see Appendix C). The alphanumeric characters that are available are alphabetical upper case and numbers plus several special characters. Options exist to choose the colour and priority of the alphanumeric display.
Commands associated with alphanumeric display are:

LABEL/DISPLAY - *load a string into the alphanumeric memory*

`CLEAR/ALPHA` - *clear the alphanumeric display*

There is also an option in the `LABEL/DISPLAY` command to use the overlay channel for text (with higher resolution) instead of the alphanumeric memory.

True Colour or RGB Operations

The `IP8500` allows pictures to be displayed in *true* colour using three image memories (channel 0, 1 and 2) simultaneously for the red, green and blue images needed to make up a true colour image. Channel 0, 1 and 2 may also be referred to as Red, Green and Blue. To start using the RGB mode of the display, execute the command:

```
SET/DISPLAY RGB
```

The `CUTS` used for mapping the image into the range of image memory need to be set individually for each of the images to be used. Next, each channel must be loaded individually with the appropriate image. This is done as follows:

```
LOAD/IMA rpict R                    ! load the red picture in channel 0
LOAD/IMA gpict G                  ! load the green picture in channel 1
LOAD/IMA bpict B                    ! load the blue picture in channel 2
```

It is now possible to use several of the other commands that control the image display, but they may perform in slightly different ways than when the system is used for pseudo–colour displays. The following comments are intended to give some guidelines.

`ZOOM` — This command will zoom all three channels together.

`SCROLL` — This scrolls only one channel at a time. The choice is governed by the input parameter which can be `R`, `G`, or `B`.

`SET/SPLIT` — This will show the three channels in their native colours.

`GET/CURSOR` — Generally cursor operations will not perform the operation you expect in RGB mode. Nevertheless, cursor values can be extracted. They will come from the last accessed image channel.

`LOAD/LUT` — This command should be avoided since LUTs are handled in a very special way in RGB mode.

To exit from the RGB mode, execute the command `SET/DISPLAY PSEUDO`.

6.2.2 XWindow display

With the term `XWindow` display we refer to a bitmapped screen supporting the XWindow 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.
Another important difference results from the way XWindows works: when an X application program terminates, all the connected windows and data structures disappear. Therefore, MIDAS starts up an independent server process, the *IDIserver*, which owns all X11 related data structures. The 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).
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. It may have one or several image channels associated with it. 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 the peripheral image displays, but exist as data structures in main memory. Also an overlay channel and an alphanumerics memory are emulated for each image display. 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 at any time. With the command `ASSIGN/DISPLAY` you switch from one display to the next.
Each image channel has independent scroll (also emulated in software) but no zoom capabilities. There are special commands like `GET/CURSOR` and `VIEW/IMAGE` which provide zoom in a special *zoom window*. Also available is an intensity transformation table but only one per image display and not one per image memory since the ITTs are emulated by convolving the ITT values with the current LUT.

Using Image Memories

Several image memories may be associated with each image display. However, at the moment the blinking between different image memories is so slow that is it not very useful to create image displays with many image channels. Images can be loaded into any of the image memories. They 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/CHANNEL 0                      ! display memory channel 0
```

The most important commands associated with handling the image memories are:

`CREATE/DISPLAY` - *create an image display with image channels(s)*
`VIEW/IMAGE` - *explore an image ...*
`LOAD/IMAGE` - *load an image into image memory*
`GET/IMAGE` - *read an image from the image memory*
`GET/CURSOR P5=w` - *read pixel values of displayed image using a zoom window*
`DISPLAY/CHANNEL` - *display the image in the selected channel*
`CLEAR/CHANNEL` - *erase data in an image channel*
`SCROLL/CHANNEL` - *scroll the image*
`SHOW/CHANNEL` - *show status of image channel*

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 pseudo–colour mode. In contrast to the `IP8500` display, the size of the LUT is not constant but depends upon how many colours are already used by other X applications running already. Commands exist to load LUTs into the image display, to modify the LUTs interactively and to read back LUTs from the image display.
Interactive modification is done via the arrow keys on the keyboard.

Some of the existing LUTs are:

```
backgr
color
heat
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 to modify the LUTs interactively.
The main commands available for handling 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*

Intensity Transformation Tables

The intensity transformation tables (ITTs) come between the image memories and the LUT and are emulated in the Xwindow environment by convolving the ITT values with the current LUT. Interactive modification is done via the arrow keys.
The main commands which control the ITT functions are:

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

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.

Using the Cursors

Each image display has two independent cursors available. One cursor is controlled interactively via the mouse, the other via the arrow keys. In addition, a region of interest

(ROI) is provided. The ROI is moved via the mouse and its size is adjusted via the arrow keys.

The cursor position can be read by pressing the ENTER button which is the leftmost button on the mouse or by hitting the RETURN key. To exit an interactive cursor command, press the EXIT button which is the button on the mouse to the right of the ENTER button, i.e. the middle button on a 3–button mouse and the right button on a 2–button mouse. Currently, a 1–button mouse is not supported in MIDAS. The commands associated with cursor control operations are:

```
GET/CURSOR - read cursor positions
CLEAR/CURSOR - disable cursors
```

Graphics

A graphics (or overlay) channel is provided with the image display. However, it is not a physical image channel like for the IP8500 display but emulated in software. Only drawing functions are supported (i.e. you cannot load an image into the overlay channel). The currently available commands associated with the use of the overlay channel are listed below:

SET/OVERLAY - *enable the overlay of graphics on top of the image*
CLEAR/OVERLAY - *disable the overlay*
DRAW/... - *draw a geometric shape like CIRCLE, RECTANGLE, etc. in the overlay plane*

Alphanumerics

The alphanumerics memory is divided up into 3 lines, the number of characters depends upon the size of the image display. Three different Font sizes are supported for the alphanumeric characters. Commands associated with the alphanumeric memory are:

LABEL/DISPLAY - *load a string into the alphanumeric memory*
CLEAR/ALPHA - *clear the alphanumeric display*

There is also an option in the LABEL/DISPLAY command to use the overlay channel for text instead of the alphanumeric memory.

6.2.3 Image Hardcopy

A hardcopy of a frame shown on the image display or stored on disk can be made only if the site has appropriate devices (see Appendix C which gives the detailed description of

the available options).

Currently grayscale and colour pictures can only be created on Laser printers which support `PostScript`. Typical command sequences for 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 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).

Chapter 7

Data Exchange Format

This chapter describes how to exchange data between MIDAS and other systems. MIDAS supports FITS and IHAP data formats for input while only FITS is available for output. The general format for exchanging data with other centers is the "FITS" (**F**lexible **I**mage **T**ransport **S**ystem) format. This tape format is described below. The second input format supported by MIDAS is the so–called "IHAP" format which was developed for use on the old HP image processing system. Observational data from La Silla may be written in this format.

It is important to recognise that MIDAS only officially supports the FITS format for data exchange. The IHAP format is provided for compatibility with the old ESO image processing systems and the data acquisition systems at La Silla. Other formats are not supported officially and no help will be provided to access such data.

Note

The internal layout of the MIDAS data files 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 on tape for a longer time.

For standard text files such as programs procedures and ASCII data files, operating systems utilities can be used. 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.

7.1 Exchange Formats

This section describes the formats supported by MIDAS for exchange of data. Normally data exchange is done using magnetic tapes, however, MIDAS also supports conversions between external and internal format directly on disk.

7.1.1 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. It is defined in Wells et al. [1], Greisen et al. [2], Grosbøl et al. [3] and Harten et al. [4] where a detailed description can be found. The FITS format is recommended by the IAU for exchange of digital information between astronomical institutes.

A FITS file contains a sequence of logical units 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. Values of parameters are decoded using standard FORTRAN–77 rules. They describe in detail the data following the header records. After the last header/data unit in the file additional records may exist.

The basic FITS paper 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 (Grosbøl et al. [3]). The allowed range of blocking factors is explicitly defined for a given media. For normal 1/2-inch 9-track magnetic tapes, factors between 1 and 10 were allowed giving a maximum physical block length of 28800 bytes. Each file terminates with a tapemark, and the last file on tape terminates with a double tapemark `i.e.` end of information.

7.1.2 MIDAS Implementation of FITS

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

- The maximum number of axes in images is 16
- Only 16 characters are decoded for string variables in the header
- Maximum 512 columns are decoded for tables
- Information in headers without associated data is not stored

7.2 IHAP Format

The IHAP format is defined in the IHAP manual (see section VIII in the March 1985 edition). It is the internal format of the IHAP system. Further, it is used for data acquisition at the La Silla observatory.

7.2.1 MIDAS Implementation of IHAP

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

7.3 How to Read/Write Tapes

This section describes the procedures to read and write magnetic tapes in MIDAS. Before accessing a tape with the `INTAPE` or `OUTTAPE` commands, the appropriate tape has to be mounted on a tape drive (Note: insert a write-ring when write on a tape). Instructions on where tape stations are located and how they are operated can be found in Appendix C.

In general the tape read/write commands will first allocate the tape unit and then position the tape at the first file (in order to make absolute positioning) before they try to access data on the tape. Depending on the local tape device implementation, the tape may be rewound after the access.

Note

The default options on the ***`OUTTAPE`*** *command will write from 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.*

7.3.1 Reading in Data Tapes

After having mounted the tape the `INTAPE`-command in MIDAS will read it:

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

where `files_list` is a list of absolute position numbers of files on the 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 maximum four characters on the names of files created. The `device` is either the logical name of the tape device (*e.g.* tape1) or a prefix for writing on disk. If the first four characters of the name are `tape`, it is assumed to be a physical device. When a filename prefix is given, the extension '`.mt`' is used. The correspondence between logical tape names *e.g.* `tape0, tape1, ...` and physical units available at your site can be found in Appendix D. The `flag` is a list of three one character flags which specifies the amount of information listed and storage format of the data (see Appendix A for full description). It is easier to read read many files from disk 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.

7.3.2 Writing Out Data Tapes

To write out data from MIDAS in FITS format the `OUTTAPE` command should be used:

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

where the `cat[,list]` is the catalogue of files to be written with an optional list of numbers. It can be defaulted by giving either '`*`' or '`?`' in which case all files in catalogues set by the `SET/xCAT` command are written out. The `device` is either the logical name of the tape device (*e.g.* `tape1`) or a prefix for writing on disk. The disk file should not be `tape` because it then would be assumed to be a physical device. Disk files will have the extension '`.mt`' by default. It is possible to write a specific file out to disk by giving the full file name with extension as the `cat[,list]` entry and its output name with extension as `device`. The correspondence between logical tape names *e.g.* `tape0, tape1,...` and physical units available at your site can be found in Appendix D. The `flag` is an optional list of three one-character flags specifying the append mode, the amount of information listed and if the `LHCUTS` descriptor in the file should be used for scaling (see Appendix A for full description). The `dens,block` parameter can specify the tape density (*e.g.* 1600 or 6250) and a physical blocking factor in the range 1-10. For cartridge tape devices, the tape density is ignored. The `type` flag is used to specify the type of FITS format to write where '`B`' indicate basic FITS *i.e.* with integer format only. The default is '`O`' for original including floating point representation.

Only files in a catalogue can be written out. In order to make a catalogue with your files use the `CREATE/xCAT` and `SET/xCAT` commands.

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

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 $x = (x_1, \ldots, x_n) \in I\!R^n$ are the independent variables and $a \in A \subset I\!R^p$ are the p parameters lying in the domain A. If A is not the whole space $I\!R^p$, 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

$$g(a) \ = \ 2 \ J(a)^T \ r(a) \qquad \textit{and}$$

$$H(a) \ = \ 2 \ (J(a)^T \ J(a) \ + \ B(a))$$

where $r(a)$ is the residuals vector

$$r(a) \ = \ (r^{(1)}(a), \ldots, r^{(m)}(a)) \ \ ,$$

$J(a)$ 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 Appendix A).

`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

$$\|a^{(solution)} - a^{(found)}\| \leq prec\ (1 + \|a^{(found)}\|) \quad ,$$

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 Appendix A 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)

$$\begin{aligned} POLY(x;a,b,c,\ldots) &= a + bx + cx^2 + \cdots \\ POLY(x,y;a,b,c,\ldots) &= a + bx + cy + \cdots \end{aligned}$$

8.8.2 Logarithmic and Exponential Function

$$\begin{aligned} LOG(x;a,b,c) &= a\ln(b+cx) \\ EXP(x;a,b,c) &= a\exp(b+cx) \end{aligned}$$

```
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

$$\begin{aligned} SIN(x;a,b,c) &= a\sin(b+cx) \\ TAN(x;a,b,c) &= a\tan(b+cx) \end{aligned}$$

8.8.4 Sinc and Sinc Square

$$SINC(x;a,b,c) = a\sin(b+cx)/(b+cx)$$

$$SINCS(x;a,b,c) = a\,sinc^2(b+cx)$$

15–January–1988

8.8.5 Distributions

$$
\begin{aligned}
GAUSS(x;a,b,c) &= a\exp\left[-\ln 2\left(\tfrac{2(x-b)}{c}\right)^2\right] \\
GAUSS(x,y;a,b,c,d,e,f) &= a\exp\left[-\ln 2\left(\tfrac{(x-b)^2}{d^2}+\tfrac{(y-c)^2}{e^2}-\tfrac{2f(x-b)(y-c)}{d\,e}\right)\right] \\
GAUSSA(x;a,b,c) &= \tfrac{a}{\sqrt{(2\pi)}c}\exp\left[-\tfrac{1}{2}\left(\tfrac{(x-b)}{c}\right)^2\right] \\
IGAUSS(x;a,b,c) &= a\int_{-\infty}^{x}\exp\left[-\ln 2\left(\tfrac{2(u-b)}{c}\right)^2\right]du \\
IGAUSSA(x;a,b,c) &= \tfrac{a}{\sqrt{(2\pi)}c}\int_{-\infty}^{x}\exp\left[-\tfrac{1}{2}\left(\tfrac{(u-b)}{c}\right)^2\right]du \\
CAUCHY(x;a,b,c) &= a\left[1+\left(\tfrac{2(x-b)}{c}\right)^2\right]^{-1} \\
LORENTZ(x;a,b,c,d) &= a\left[1+\left(\tfrac{2(x-b)}{c}\right)^2\right]^{-d} \\
POISSON(x;a,b,c) &= \tfrac{ab^x\exp(-b)}{\Gamma(x+1)} \\
LAPLACE(x;a,b,c) &= a\exp\left[-\ln 2\left(\tfrac{2|x-b|}{c}\right)\right] \\
TRIANG(x;a,b,c) &= a\left(1-\tfrac{|x-b|}{c}\right)
\end{aligned}
$$

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 Wright [1]. Bard [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.

[1] Gill P.E., Murray W and Wright M.H. . Practical Optimization. Academic Press. London. 1981.
[2] Bard Y. . Non--linear Parameter Estimation. Academic Press.

London. 1974.
[3] Gill P.E., Murray W. . Algorithms for the solution of non-linear least squares problems. SIAM J. of Num. An., vol 15, pp 977-992, 1978
[4] Gill P.E., Murray W. and Pitfield . The implementation of two revised algorithms for unconstrained optimization. Rep. NAC 11. Nat. Phys. 1972. Lab., Teddington. England.

Appendix A

Detailed Command Description

Appendix B

Acknowledgements

B.1 General

It is of course never possible to adequately acknowledge the many small, but extremely useful, comments which the Image Processing Group 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 which will be made available to MIDAS sites through ESO on special request.

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.

Significant contributions were added in the application area. A new version of the graphical user interface XSpectra has been implemented by Cristian Levin. 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. 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. The main installation contains a number of UNIX workstations and Servers (most of which are SUN/SPARC compatible). The logical names of the workstations are **ws***n* where *n* is a running number. The Servers have the prefix **ns** or **mc** depending on their main function as either file servers or main computers. 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 User room 213. 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 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 of MIDAS. After having started MIDAS with `inmidas` enter a double digit number (00,..,99) as a MIDAS unit if you want to use image and graphic displays.

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.

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 213 (see

table C.1).

Hardcopies of plots are made using the `ASSIGN/PLOT` and the `PLOT` commands (see Chapter 6). The first parameter of the `ASSIGN/PLOT` command specifies the plotting queue/device to be used. The `help` command in MIDAS can be used to get an upto date list of output devices. The following table C.1 gives the default queues and devices:

MIDAS Logical Name	UNIX Queue	Device
LASER	ps2usr1	HP LaserJet III
COLOUR	pc2usr0	Tektronix Phaser PX
SLIDE	sl2usr0	Chromascript
PENPLOT	hp2usr0	HP plotter 7550A, A4
LPRINT	lp2usr0	Lineprinter

Table C.1: Printer and Plot Queues

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. Although most 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.

C.1.4 Film Hardcopy

This Section describes the Film hardcopy devices available at ESO, Garching. Systems for recording 35mm Colour Slides and 70mm B/W negative film are 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 format specified in the table below.

UNIX queue	Device	Film	File format
sl2usr0	Agfa	35mm Colour	PostScript
fr0mmf0	Celco	70mm B/W	FITS

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. For B/W film, the `outtape` command may be used to create a FITS file on disk which then can be send to the queue.

Processing of Film Hardcopies

Depending on the number of entries, images in the B/W and Colour queues 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 half inch magnetic tape drivers are available for either the UNIX or the VMS systems. Located in the User Room 213, there are four drives connected to UNIX systems and one drive for the VAX/VMS system. Cartridge tape drives (e.g. QIC, 8mm and DDS/DAT) are also being installed for general usage in the User Room 213.

They can be accessed from any UNIX workstation on the network. The step by step procedure is as follows:

- Login on a UNIX workstation.
- Run MIDAS with **inmidas**.
- Mount your tape on a drive connected (see below). If you want to write on the tape remember to put a write ring.
- 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! During reading the tape drive will itself sense the tape density used. Thus, the generic tape name can always be given when reading.
- Dismount your tape from the drive so that others can use it.
- Logout from MIDAS with **bye** and from the workstation.

The logical device names currently defined for magnetic tape drives on UNIX systems with the following generic notes:

1. An asterisk (*) in front of the MIDAS name indicates that the drive is located in the computer room and the access is restricted.
2. 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.
3. The temporary *register* accounts used by visitors belong to domain `eso`.
4. New tapes are from factory write enable.
5. Tapes can be obtained from Rinze de Roos, room 220/1, 2nd floor.

6. 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)

1/2 inch MAGTAPES:						
Midas name	**Host**	**Device**	**Tape**	**Density**	**Location**	**Domain**
*tape0	ns0	/dev/nrst17	1/2 inch	6250 dpi	Room 000	arc
*tape0h	ns0	/dev/nrst17	1/2 inch	6250 dpi	Room 000	arc
*tape0m	ns0	/dev/nrst9	1/2 inch	1600 dpi	Room 000	arc
tape1	ws1	/dev/nrst18	1/2 inch	6250 dpi	User room 2nd fl.	eso
tape1h	ws1	/dev/nrst18	1/2 inch	6250 dpi	User room 2nd fl.	eso
tape1m	ws1	/dev/nrst10	1/2 inch	1600 dpi	User room 2nd fl.	eso
tape2	ws1	/dev/nrst19	1/2 inch	6250 dpi	User room 2nd fl.	eso
tape2h	ws1	/dev/nrst19	1/2 inch	6250 dpi	User room 2nd fl.	eso
tape2m	ws1	/dev/nrst11	1/2 inch	1600 dpi	User room 2nd fl.	eso
tape3	st0	/dev/nrst17	1/2 inch	6250 dpi	ECF room 4th fl.	ecf
tape3h	st0	/dev/nrst17	1/2 inch	6250 dpi	ECF room 4th fl.	ecf
tape3m	st0	/dev/nrst9	1/2 inch	1600 dpi	ECF room 4th fl.	ecf
tape4	ws0	/dev/nrst19	1/2 inch	6250 dpi	User room 2nd fl.	eso
tape4h	ws0	/dev/nrst19	1/2 inch	6250 dpi	User room 2nd fl.	eso
tape4m	ws0	/dev/nrst11	1/2 inch	1600 dpi	User room 2nd fl.	eso
tape5	ws8	/dev/nrst18	1/2 inch	6250 dpi	SkyLight room 4th fl.	eso
tape5h	ws8	/dev/nrst18	1/2 inch	6250 dpi	SkyLight room 4th fl.	eso
tape5m	ws8	/dev/nrst10	1/2 inch	1600 dpi	SkyLight room 4th fl.	eso

REMARKS:

- For each 1/2 tape drive there is three names but actually only two densities, e.g. tape2 (default: 6250 dpi), tape2h (high density 6250 dpi) and tape2m (medium density 1600 dpi).
- 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 MIDAS name as taken from the above table and used with command "INTAPE/FITS" then actually sets 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.

8mm TAPES ("EXABYTE"):						
Midas name	**Host**	**Device**	**Tape**	**Capacity**	**Location**	**Domain**
tape8mm0	ns2	/dev/nrst1	8mm	5.0 Gb	IPG room 4th fl.	ipg
*tape8mm1	ns0	/dev/nrst3	8mm	2.2 Gb	Room 000	arc
tape8mm2	st0	/dev/nrst3	8mm	2.2 Gb	ECF room 4th fl.	ecf
tape8mm3	ws8	/dev/nrst4	8mm	2.2 Gb	SkyLight room 4th fl	eso
tape8mm3	ws8	/dev/nrst1	8mm	5.0 Gb	SkyLight room 4th fl.	eso
tape8mm4	ws4	/dev/nrst0	8mm	5.0 Gb	User room 2nd fl.	eso
tape8mm5	ws5	/dev/nrst1	8mm	2.2 Gb	User room 2nd fl.	eso
*tape8mm6	mc6	/dev/nrst1	8mm	5.0 Gb	Room 000	eso
*tape8mm7	mc7	/dev/nrst1	8mm	5.0 Gb	Room 000	eso

REMARKS:

- 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.

QIC TAPES:						
Midas name	**Host**	**Device**	**Tape**	**Capacity**	**Location**	**Domain**
*tapect0	ns0	/dev/nrst8	QIC-24	60Mb	Room 000	arc
tapect1	ws0	/dev/nrst8	QIC-24	60Mb	User room 2nd fl.	eso
tapect2	ns2	/dev/nrst0	QIC-150	150Mb	IPG room 4th fl.	ipg
tapect3	ns3	/dev/nrst0	QIC-150	150Mb	ECF room 4th fl.	ecf
tapect4	ws2	/dev/nrst8	QIC-24	60Mb	User room 2nd fl.	eso
tapect5	ws3	/dev/nrst8	QIC-24	60Mb	User room 2nd fl.	eso
*tapect6	mc6	/dev/nrst0	QIC-150	150Mb	Room 000	eso
*tapect7	mc7	/dev/nrst0	QIC-150	150Mb	Room 000	eso
tapect8	ns2	/dev/nrst3	QIC-24	60Mb	IPG room 4th fl.	ipg
*tapect9	ns1	/dev/nrst0	QIC-150	150Mb	Room 000	eso

REMARKS:

- 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.

DIGITAL AUDIO TAPES (DAT):						
Midas name	**Host**	**Device**	**Tape**	**Capacity**	**Location**	**Domain**
tapedat0	ns2	/dev/nrst2	DDS	1.2Gb	IPG room 4th fl.	ipg
tapedat1	ws4	/dev/nrst1	DDS	1.2Gb	User room 2nd fl.	eso
tapedat2	ws5	/dev/nrst0	DDS	1.2Gb	User room 2nd fl.	eso
tapedat3	ws8	/dev/nrst3	DDS	1.2Gb	SkyLight room 4th fl.	eso

REMARKS:

- DDS drives use Sony 60/90/120 min. DAT tapes.
- DAT tapes are write enable when the swith in front of the tape closes the hole.

C.2 Operating Systems

The main operating system used for image processing in ESO is UNIX. On the SUN workstation the X11 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 the secretaries.

C.2.1 Login Procedures

To login to a computer system you have to hit the return key of the terminal you want to use. Terminals are connected either directly or through a PBX device which is used for terminal switching. In the latter case the PBX will ask you to **Select Destination** which should be the last three characters of the host-name *e.g.* **ns1/mc6/mc7** for the central UNIX Servers and **ts0/ts1** for general purpose TCP/IP terminal servers. After this you will get the welcome message of the computer written on your terminal with a request to give your login identification and password. In case you don't have an account on the systems you should contact the System Manager (room 220/2).

C.2.2 Differencies between VAX/VMS - UNIX

This section gives some hints on UNIX commands for people who are more familiar with the VAX/VMS operating system. There are a number of differencies between UNIX and VAX/VMS systems. Some of the very basic ones are listed in Table C.2.

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.

Action/item	UNIX	VAX/VMS	Remarks
Directories	top/next/etc	[top.next.etc]	/ is UNIX root directory
File names	name.ext	name.ext;ver	UNIX is case dependent and has no version no
Change directory	cd	set def	
Directory list	ls	dir	
Current directory	pwd	show def	
Online help	man	help	
Text editor	vi	edit	**emacs** can be used on all systems
Logout	exit	logout	
Start MIDAS	inmidas	inmidas	
Select MIDAS version and options	$setmidas	setmidas	select the version of MIDAS
Restart MIDAS	gomidas	gomidas	

Table C.2: Differencies between UNIX - VAX/VMS

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 **92NOV**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

If you are going to install MIDAS on your system you have received the release tape which is written in either VAX/VMS BACKUP or tar format with the label **92NOV**. This version of MIDAS can run under DEC/VMS version 4.7 (or higher) or UNIX operating systems. The monitor and low level interfaces are coded in C which means the portable MIDAS requires both a FORTRAN-77 and a C compiler for the installation. The installation of MIDAS has been certified with the public domain GNU C compiler which can be obtained from the Free Software Foundation.

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.

Two areas may require special attention during the first installation. They relate to the tape I/O routines and image display interfaces. Concerning the latter, MIDAS conforms to the standard IDI interfaces. IDI's are provided only for DeAnza IP8500 systems (connected to VAX/VMS systems) and X-Window 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 **92NOV** is the 8th official release of portable MIDAS. This release contains the main body of MIDAS commands, however, some procedures and packages have not yet been fully verified. A number of new applications have been added to the version. Given below is a list of the known deficiencies, changes and improvements compared with the previous release:

Deficiencies:

1. The CCD context is not yet available.
2. The command `UNION/TABLE` is not implemented

Improvements:

1. The Table File system has been upgraded significantly by allowing single table entries to contain arrays of values.
2. The FITS reader/writer (IN/OUTTAPE) now supports the new Binary Table and Image extensions. OUTTAPE will now write tables in Binary format by default.
3. The PostScript output files from plot and display hardcopy commands are now written in Encapsulated PostScript File (EPSF) format.
4. The Time Series Analysis context has been implemented.
5. A programme for scheduling Photometric observations has been added as the first component of a Photometry context.
6. The IRSPEC context has been revised and improved.
7. A number of Graphical User Interfaces (GUIs) are now available in this release on a trial basis. They include XHelp, XEchelle, XSpectra and XFilter. The three latter ones are based on the MIT X11 widget set, whereas XHelp is using OSF/Motif. The ESO standard for GUIs has not been finalised yet, which means that the current set of GUIs will be significantly modified in the next release to conform to this standard.

   It should be possible to install XHelp on all systems having the OSF/Motif software. The MIT X11 based GUIs have only been tested on Sun Sparc systems and may give installation problems on other systems.
8. The command VIEW/IMAGE, which we recommend for first viewing of your data, has been enhanced significantly.

For a 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. Only these parts are included in the package you receive. Users are referred to the on-line help facility which contains all the latest updates. Please note that due to typographical problems, the underscore in the printed helpfiles has been printed as an inverted comma.

Please replace the old pages in your MIDAS User's Manual with the new ones. A list of the parts to be replaced is given below :

- Volume A, Titlepage
- Volume A, Chapter 3 to be replaced.
- Volume A, Chapter 5 to be replaced.
- Volume A, Chapter 6 to be replaced.
- Volume A, Appendix A to be replaced.
- Volume A, Appendix B to be replaced.
- Volume A, Appendix C to be replaced.
- Volume A, Appendix D to be replaced.
- Volume A, Appendix E to be replaced.
- Volume B, Titlepage
- Volume B, Chapter 6 to be replaced.
- Volume B, Chapter 7 to be replaced.
- Volume B, Chapter 8 to be replaced.
- Volume B, Chapter 14 to be added.
- Volume B, Chapter 15 to be added.
- Volume B, Appendix A to be replaced.
- Volume B, Appendix E to be replaced.
- Volume B, Appendix G to be replaced.
- Volume B, Appendix H to be replaced.
- Volume B, Appendix J to be added.

D.5 Use of NAG Library

The NAG mathematical library is still used in several 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

A set of dummy routines are provided for sites that do not have a NAG library. This new implementation gives them the possibility of using the FIT package (in this case, only the Newton-Raphson method will be supported).