TASCOM USER MANUAL

Version 3.15 for Linux – BW1, BW2, W1

Jørgen Bundgaard and Per Skaarup

Risø National Laboratory, Roskilde, Denmark
June 2004

TASCOM MANUAL VERSION 3.15 BW1-2, W1

INDEX AND QUICK REFERENCE GUIDE

8. Counting, scalers and ratemeters **page 32**

MON I AUX TIM	Monitor, detector, auxiliary, timer
F_MON F_ I F_AUX F_TIM	Read monitor, detector, etc while counting
PRSC	PRSC=0 preset time; PRSC=1 preset MON
PRES	PRES in seconds or in monitor counts
COUN	Count for preset PRES and read scalers
COST	Start counting and return control to TASCOM
COFL	If scalers still counting COFL=1
_ASK_CC	=1: Ask if Control(c) stop counting
RdS RdM RdF	Detector ratemeter Slow Medium Fast
RmS RmM RmF	Monitor ratemeter Slow Medium Fast
RaS RaM RaF	Auxiliary ratemeter Slow Medium Fast
RATE	RATE=1 ratemeters enabled all the time
WAIT s	Wait s seconds
TIME	Returns time in seconds
GET_TIME	Update YEAR, MONTH, DAY, etc.
TFRQ	Timer frequency TFRQ=1000 or 10
_CH_TFRQ	Automatic shift between TFRQ 1000 or 10

9. Data storage **page 34**

FINA	Filename of .dat data file
SEGN	Segment number of .dat data file
CLOSE_F	Close all open data file
_OW_DAT	=0: data files are not overwritten
LPRP LFIP LFIH	List PRint Params, FIle Params, FIle Header
DPRP DFIP DFIH	Define PRint Params, FIle Params, FIle Header
APRP AFIP AFIH	Append PRint Params, FIle Params, FIle Header
REAP	Remove all appended parameters
FICA	Lists buffers and file names
DFIT	Define file text (e.g. DFIT='Background scan')
MTOU	=1: All motor positions on file header
OUT	writes PRP/FIP variables to screen/file
AOUT	writes FIA variables to array data file
FIOU	FIOU=0 then no variables to file
PROU	PROU=0 then no variables to screen
AROU	AROU=0 then no variables to array data file
DATA_DIR	Define data subdirectory
REOPEN_F	Reopen a data file
FIAR	Filename of .dab or .dar array data file
SEAR	Segment number of .dab or .dar array data file
_OW_DAR	=0: array data files are not overwritten
FINL	Define logfile name (e.g. FINL='mylog')
LOGF LOGP	Log file/printer enable/disable
_LOGDIR	Path for logfile
LLOG DLOG ALOG	List, Define, or Append LOG buffer
LOG_PAR	Log buffer
LOG	Log information (e.g. LOG 'New Sample',M2)
PLOG	Log and print information (e.g. PLOG 'New Sample',M2)
COMM	Comment on logprinter
LOG_CLOS	Log when scan file is closed

1. Introduction.

This manual is a user's guide to the program TASCOM (Triple Axis Spectrometer COMmands). As the name indicates, the program was devised for a particular type of diffractometer, but the program's structure is so general that it has been adapted to many other diffraction instruments. The program is written in C but it is not expected that the user of this manual will need to alter the source code. Certain key aspects of the program's internal structure will, however, be discussed in this manual. This introduction describes the main features of TASCOM. The following sections deal in more detail with these features.

Command files. The most powerful feature of TASCOM is that it has its own simple programming language, with some resemblance to BASIC. This allows one to write customized command files for performing certain scans. These command files all have the extension .tas, e.g. myscan.tas. The command file is executed simply by typing the name (without the extension) and pressing the return or enter key. TASCOM is ready to execute commands when it displays the > prompt. Several such command files for standard scans are supplied with the program.

Symbol table. As well as these external command files, TASCOM has a number of predefined internal commands. These do not exist as .tas files, but are subroutines of the program TASCOM. The names of these commands are stored in a symbol table. The names of .tas command files are not stored in this table. If TASCOM cannot find a command in the symbol table, it begins searching on the hard disk for a .tas command file. TASCOM allows the user to define under which subdirectory the search occurs. This feature allows different users to keep their customized commands separate, avoiding confusion.

Variables. To designate such things as motors, angles, distances, counting times, measured intensities, the TASCOM language uses variables with names of eight letters or less. The variables are stored in the symbol table. Again, certain variables are predefined internally, while other customized variables can be defined by the user. Internal variables are, for example, the preset counting time PRES or the measured intensity I. There are a number of internal variables called status variables or flags, which determine whether TASCOM performs certain actions. For example if FIOU is 1, the program will send output to a datafile, otherwise it won't. Also, certain internal variables are associated with character strings, for instance the internal variable FINA represents the first characters of the filenames where TASCOM stores data. TASCOM supports array variables. Arrays are particular useful for storage of counts from position sensitive detectors and multichannel analyzers.

User variables can be defined at any time in terms of internal variables or other, already defined user variables. For example, USER=M1*M2, defines the variable USER and calculates its value from the present motor positions M1 and M2. In contrast to user-defined commands (.tas files), the names of user-defined variables are stored in the

symbol table, filling a previously empty slots there. Note that for a variable which is defined in terms of other TASCOM variables, the value of the variable is normally *not* automatically updated if one of the variables in terms of which it was defined changes.

Arithmetic and logic. As well as recognizing variables and commands, the TASCOM programming language can perform basic arithmetic, and recognizes a number of standard functions, e.g. SIN(X). Certain logical operations are also supported, e.g. IF, WHILE and DO. A description of the mathematics and logic recognized by TASCOM is given in section 3. A good understanding of this is essential to make the most of the programming capabilities of TASCOM.

Assignments. In common with BASIC, the = sign plays a special role in TASCOM. It assigns the variable on the left hand side to be equal to the number, variable or mathematical expression on the right. In the example USER=M1*SIN(M2), if the value of M1 is 10 and that of M2 is 30 then USER is assigned the value 5.0 (trigonometric functions are in degrees). Certain internal variables represent motors. When these are assigned a value, the motor moves to this value. For example M1=1000 moves motor M1 to position 1000. Any command that moves motors uses such assignments at some stage of its execution. Certain internal variables represent character strings, and are assigned by putting the string in single quotation marks, for example FINA='test'. In order to find out what value is currently assigned to a variable, for example to M1, one writes ?M1. TASCOM will return the current value of M1. Indirect variables may be assigned with the @-sign, for example MOTOR=@M1.

Scan variables. In a user-defined command file, the user may want to enter some values for certain variables which will be different from scan to scan, for example the position of motor M1 at the beginning of the scan. This is done by writing VAR: and then one or more variable names at the top of the command file. When the command file is executed, TASCOM will prompt for values of these variables. For example VAR:M1BE will prompt for the value of the (user-defined) variable M1BE. Further along in the command file, the assignment M1=M1BE moves M1 to its start position.

Scans. The way TASCOM performs scans is to move motors, count, store the values of certain variables (e.g. intensity, motor positions) on a file, and move to the next step of the scan. There are commands to describe which variables are stored on the file at each step, which ones are written to the screen, and which ones are used to make a simple plot of the scan on the screen when the scan is finished. If need be, some variables which do not change during the scan can be stored at the top of the data file, for reference.

Data storage. All the data files produced by TASCOM have extension .dat if they are from a scan. The special data array files that holds arrays from reading the MCA have extension .dab or .dar. Both types of data files have a name consisting of seven letters or less plus a segment number, which increases by one each scan. TASCOM has facilities for keeping different users' data files in different directories.

External access. TASCOM has a facility that allows other programs to communicate with it and in this way obtain various useful information, such as motor positions, values of scalers, data from partially completed measurements etc. This facility is called ODA, for "online data access". A software package is available separately that interfaces TASCOM to the World-Wide-Web, and allows the remote supervision over the Internet of experiments controlled by TASCOM.

The following shows the structure of a typical command file, actually a somewhat reduced version of the standard scan file sma.tas.

```
VAR: MOTOR,MBEG,MEND,NPOI,PRES          prompt for scan variables
AFIP MOTOR,I                            variables to be stored on file
APRP MOTOR,I,MON                        variables to be written to screen
DPLO MOTOR,I                            variables to be plotted after scan
SEGN=SEGN+1                             number
MSTP=(MEND-MBEG)/(NPOI-1)               step between scan points
! Now follows the loop                  a comment
WHLE J.LE.NPOI                          start of scan loop
  MOTOR=M1BE+M1ST*(J-1)                 move motor MOTOR
  COUN                                  count for a given preset
  OUT                                   output on screen and data file
NEXT                                    loop til next step
PLOT                                    plot of scan on line printer
```

Many more useful features of TASCOM are described in the following pages. A quick-reference list of standard commands, variables and important files is given in the front of this manual. The names in the quick reference list are written in **bold** at their first appearance in the following sections. There is a help facility in TASCOM which may be used as another quick-reference. A list of the subtopic names in the help facility is given by typing HELP.

2. Log-in, start, and exit.

The user log-in account is: Username: tascom
Password:

TASCOM runs at the PC under the Linux operating system. A quick reference guide to Linux and to the emacs text editor as well as a sketch of the directory tree for the TASCOM files are included in this manual as appendices. The user home directory is /usr/users/tascom/.

TASCOM is started by typing tascom. The characteristic prompt > indicates that TASCOM is running. If TASCOM does not start correctly, see 'Restart of TASCOM after error' in the appendix.

Before starting, TASCOM checks if another TASCOM is already running. If so, the following message is printed:

TASCOM probably already running.
If you start a new TASCOM, the running one may be killed.
Start new TASCOM anyway (y or n)?

If an old TASCOM was not exited correctly, this message may appear even if the old TASCOM is not running and in that case a new TASCOM may be started.

When TASCOM is started and if a startup file exists in the user command directory or in the system command directory, the user is asked if the startup file should run. The startup file is a .tas command file the name of which is given by assigning **SFIL**, e.g. SFIL='start'. Different users can thus have customized startup files which assign certain variables standard values or execute certain commands.

TASCOM can always be interrupted by typing **control-C**, which stops all action (running motors, counting etc.), and returns the control to the user in the prompt (>) mode. Control-C does not kill TASCOM.

The on-line printer echoing the TASCOM dialog can be turned on and off with the TASCOM command **DIALOG**. DIALOG=1 turns the printer on while DIALOG=0 turns the printer off. The TASCOM dialogue is saved in the file dialog.dat in the user directory if DIALOG is set to DIALOG= -1. If DIALOG=2, the dialogue is printed as well as saved on the file.

The command **EXIT** closes all files and returns to Linux. When TASCOM is left with the EXIT command, the present state of TASCOM is saved, i.e. variables and parameters are saved on files. For example, the binary version of the symbol table is saved, all motor positions are read from the ECB system to the motor control block file together with the motor set-up parameters, the user-defined specification of output to data files is saved, arrays are saved, etc. The variables and parameters are read from these files on starting TASCOM again. The present state of TASCOM may be saved at any time with the command **_SAVE_S**.

TASCOM may also by exited - brutally - by typing **control-Z** or by closing the window in which TASCOM is running. On such exits, the present state of TASCOM is *not* saved

and when TASCOM is started again, the start parameters are taken from the last saved versions of the symbol table, motor control block, etc.

At start up TASCOM decides which hardware is supported by trying to communicate with the peripherals generated into that particular TASCOM version. If the communication with the ECB system fails, the user is asked whether TASCOM should continue without supporting the ECB system or not. If the communication fails with any of the other units, e.g. the temperature controller, a 'no support' message is printed while TASCOM continues to start up. When TASCOM is running, calls to units that are not supported are replaced by dummy calls. Thus TASCOM may run with only part - or even none - of the normal peripherals and this may be useful in some situations. With the command **SUPPORT** the user can have a list on the screen of the hardware supported at any given time.

Even if a particular hardware unit is not included in the present TASCOM version, it may still be possible to communicate with the unit by using the **GDEV** command. See the appendix page 76 for details.

Normally TASCOM will be running with communication to the ECB system. If for some reason the communication fails, a time-out error will result. This is considered a fatal error, i.e. TASCOM immediately returns to prompt mode and an error message will be printed. It is then as a rule necessary to exit TASCOM, reset the ECB system and reenter TASCOM.

The number of motors supported by TASCOM depends on the installed hardware and on the symbol table. On starting TASCOM the motor positions for the motors in the ECB system are read from the ECB system. The ECB system is equipped with a battery power supply in case of power failure. Therefore, it will not loose the motor positions if the power is turned off. If for some unexpected reason TASCOM is stopped before the motor positions are updated in the motor control block file, then when the program is started again, TASCOM will read the positions from the ECB system to the motor control block. If there are any differences with the values previously in the motor control block file, the user will be notified. This will happen if, for example, the manual box has been used while TASCOM was not running.

At start up and when the command SUPPORT is given, TASCOM lists the assignment to the variables SYM_NAME, LAST_GEA, LAST_DIS and possibly LAST_MUX. The variables are useful at instrumentations with several mechanical set-ups each having different symbol tables and set-up files. SYM_NAME holds the name of the last loaded ASCII symbol table and is updated when an ASCII symbol table is loaded. LAST_GEA, LAST_DIS and LAST_MUX show the names of the gearing file, the display set-up file and the multiplexer file, respectively, provided they are updated when the files are loaded. The user should include these three variables in the files as shown in the examples in the appendix.

A read-only version of TASCOM may be invoked with the Linux command **sim_tascom**. At start-up read-only TASCOM reads the present state of the symbol table, motor control block, etc. and enters calculation mode (CALC=1, see page 45) which is indicated by the prompt C>. Read-only TASCOM never writes anything on files and does not communicate with the ECB system or other peripheral equipment. It may be run in a

window at the same time as the normal TASCOM runs in another window. It is usefull for checking command files by doing simulated scans, e.g. to check the values for the motor settings.

3. TASCOM syntax

Basic syntax. TASCOM does not distinguish between UPPERCASE and lowercase letters in variable names or command names but translate lowercase input to uppercase, e.g. EXIT, Exit, and exit result in the same command. TASCOM does not distinguish between integer and real input, but translates all input to real. Exceptions to this rule are variables defined as character strings. Such character strings may have a length of up to 256 characters. A real variable is output as real or integer depending on the format specified for the variable.

Variable and command names can be any combination of up to eight letters, digits, or underscore (_). They must not start with a digit and only certain system variables and commands should start with underscore. User symbols must not start with an underscore. Motor names may only have up to three letters or digits and must start with a letter or the string 2T. The at-sign (@) is used for indirect variables.

Mathematical expressions recognized by TASCOM can involve the following elements.:

+ - * / ^ % ( )
ABS EXP LOG LOG10 SQRT
SIN COS TAN ASIN ACOS ATAN ATAN2

ASI, ACO and ATA may be used instead of ASIN, ACOS, and ATAN, respectively, for compatibility with older TASCOM versions. In addition, integer and fraction of a real symbol is returned by the command **GET_INT**, eg GET_INT M1 returns integer part and fractional part of symbol M1 in the variables _INT and _MOD, respectively.

Assignments. As in BASIC, the = sign assigns the numerical value of the expression on its right hand side (which may involve other variables and mathematical functions) to the variable on the left hand side. For example, M7=M7+LOG (STEP).

String variables are entered enclosed in single quotation marks, e.g., FINA = 'mine'. The maximum length of the string is 256 characters.

When a previously undefined variable is assigned a value or a string, e.g. USER=2000 or I_AM='my_name', then the variable is defined as a new user symbol and fills an empty a slot in the symbol table. If TASCOM encounters an undefined variable in an expression (e.g. M1=USER/40 where USER does not already exist in the symbol table) an error message will result. The size of the symbol table is finite, so it may occasionally be necessary to erase user-defined variables from the table.

If a car return is typed immediately after the "=" sign in an assignment, then on the following line is printed: the symbol name, an "=" sign, and the present value of the symbol. The user may then edit this line. It is particular useful when making modifications in long string variables. This feature works only for assignments from the keyboard, not assignments in .tas command files.

Command strings. The purpose of command files is to avoid having to repeatedly type in long strings of commands. However, a series of commands and assignments can also

be executed simply by typing them on one line, after the > prompt. All commands and assignments must be separated by at least one blank character. In the following example, TASCOM moves motor M2, counts, moves motor M2 again and counts again:

M2=2000 COUN M2=4000 COUN

Line editing. One can use the **Arrow keys** to retrieve the last executed command strings, possibly edit them, and execute them again without having to type them in from scratch. **Ctrl(A)** moves the pointer to the start of the input line, **Ctrl(E)** to the end of the line, **Ctrl(U)** erases the input line from the pointer to the start of the line, and **Ctrl(K)** erases the input line from the pointer to the end of the line. The input line may have up to 256 characters. The start of the input line is the first character typed after the prompt (>), and the end of the line is a car return character typed by the user. Thus, in the Tascom window a Tascom input line may be displayed as more than one line.

Output of variables. The value of a variable can be output to the screen by a question mark '**?**' directly in front of the variable name. A list of variable values can be output in the same manner by separating the variable names with commas. A comment may also be written by putting it in single quotation marks. For example:

?'RECIPROCAL LATTICE POINT : ',X,Y

will return:

RECIPROCAL LATTICE POINT : X = 0.0000 Y =6.0000

Such comments are useful for later reference if a slave printer is recording everything that appears on the screen. A set of variables often required at once can be conveniently output by creating an appropriate command file.
An example is the command file:

```
?'MOTOR STATUS:'
?M1,LM1,LM2
?M2,LM2,UM2
```

which outputs

```
MOTOR STATUS:
M1=   5432  LM1=   -11000  UM1=   15000
M2=    657  LM2=   -10000  UM2=    1000
```

Format of variables. The format in which a variable is output may be specified by the user. A format specification is included in the symbol table for each symbol. User defined symbols are given the default format %-12.5g for real symbols and %12s for string symbols. With the command **_US_FORM** the user can define another format to be used for new real user symbols, e.g. _US_FORM='%-15.7g'. Following an assignment to _US_FORM, Tascom must be exited and reentered. The **FORMAT** command is used to show or change the format for symbols. For example,

FORMAT PROU

shows the format for symbol PROU, while

FORMAT PROU,'%5d'

sets the format for symbol PROU to %5d, i.e. integer with 5 digits precision. A detailed description of the argument to the FORMAT statement and the format specification in the symbol table is found in the appendix (page 78).

Indirect variables. A variable may be addressed indirectly by making an assignment to it with the indirect character @. For example,

MOTOR = @M1

assigns the variable MOTOR to M1. After the assignment, commands with MOTOR act on M1, e. g. MOTOR=100 sets motor M1 to position 100 and ?MOTOR reads the position of motor M1. In the same way

SCALER = @MON ?SCALER

reads and prints the monitor scaler.

Control statements. In TASCOM it is possible to do loops and nested scans by using the control statements. The allowed control statements are:

DO < command sequence > UNTIL < logical expression>
(e.g DO M1=M1+10 COUN UNTIL J.GE.1000)

WHILE < logical expression > < command sequence > NEXT
(e.g. WHILE J.LE.NPOI M1=M1+MSTP*J COUN NEXT)

IF < logical expression > < command sequence > ELSE < command sequence > END
(e.g. IF I.GE.IMAX M1=M1+MSTP COUN ?M1 ELSE ?I END)

IF < logical expression > < command sequence > END
(e.g. IF AUX.LE.1000 ?'AUX small' END)

WHLE may be used instead of WHILE and UNTL may be used instead of UNTIL for compatibility with older TASCOM versions.

Logical operators used in TASCOM are:

.LT. or **<**	less than
.LE. or **<=**	less than or equal to
.GT. or **>**	greater than
.GE. or **>=**	greater than or equal to
.EQ. or ==	equal to; may be used also to compare string variables
.NE. or |=	not equal to; may be used also to compare string variables
.AND.	logical and
.OR.	logical or
.NOT. or |	logical negation

In the above examples, a special loop counter called **J** is automatically incremented each time the loop is executed. J always starts at J=1 and cannot be assigned. TASCOM keeps track of J in nested loops. There is no limit as to how many nested loops one can have. Note that the semicolon separator must not occur in loops (see page 42).

In command files the control keyword (UNTIL, WHILE, or IF) and the following logical expressions must be on one line. Exceptions to that rule are expressions with .AND. or .OR.. Lines with such expressions may be split after .AND. or .OR., ie .AND. or .OR. must be the last token on the split line.

String variables. A number of functions are available for manipulating string varaibles as illustrated by the following examples:

Adding two or more strings:

```
FINA = FINA + 'xyz'
FINA = ’abc’ + FIAR + ’123’
```

Comparing strings:

```
IF (FINA == 'abcd') ....
IF (FINA |= FIAR + 'xyz') ...
```

STRingLENgth. Return the length L of the string FINA:

```
L = STRLEN (FINA)
```

STRing TO Real. Return the real value R of the string STRING:

```
STRING=’1234’
R = STRTOR (STRING)
```

Real TO STRing. Convert the real variable R to a string FINA with the format specified:

```
R=12.34
FINA = RTOSTR ('%6.3f', R)
```

STRing TO Array. Return the ASCII values of the characters in the string STRING to the array elements AB[0] to AB[3]:

```
STRING=’abcd’
AB[0...3] = STRTOA (STRING)
```

Array TO STRing. Return a string FINA with the characters those ASCII values are specified in array AB[]:

```
FINA = ATOSTR (AB[])
```

Keyboard function keys F1 to F10. The function keys F1 to F10 are programmable by the TASCOM variables F*i* where $1 \le i \le 10$. For example,

```
F4 = '?M1,M2'
```

places the command line ?M1,M2 in the F4 buffer and and pressing function key F4 on the keyboard will execute the command line. If the last character in a command line is a space, e. g.

```
F10 = '?M1,M2 '
```

pressing the function key will not execute the command line immediately but display the line on the screen waiting for the user to execute the command line by typing a carreturn, possibly after editing in the command line.

4. Arrays

Arrays in TASCOM are a collection of real variables. Each array variable can hold a number in the range $\pm 10^{\pm 38}$ with 8 decimal places of significance. Array variables can be either one-dimensional or two-dimensional. The one-dimensional arrays can have up to 65536 elements and the two-dimensional arrays can have up to 65536*65536 elements. However, the number of elements is limited by the available memory space.

Arrays can be system arrays or user arrays. System arrays are defined and dimensioned in the ASCII symbol table file and can not be defined, dimensioned or deleted by the user. User arrays must be defined and dimensioned by the user with the **DIM** command before any other reference is made to them.

Syntax.
Arrays may be referred to without specifying the subscripts,

```
A[] = B[] + C[]
```

where A[], B[], and C[] are arrays with the same dimensions.

Array elements may be referred to with subscripts, e.g.

```
D = A[1,56]
```

where D is a variable and A is a two dimensional array.

The subscripts may be variables or expressions as well, e.g.

```
D = A[1,56] + B[N,(M+A[1,56])]
```

A range of elements in an array may be selected, e.g. A[12...56]. The ellipsis (...) specifies that the elements A[12], A[13], ..., A[56] are to be referred to in an OUT command or in a print command, e.g. ?A[12...56]. The range specification can be used also in two-dimensional arrays A[5...10,N...M].

A user array must be declared and dimensioned before any other reference is made to the array. The **DIM** statement declares an array, e.g.

```
DIM A[64,64]
```

In the DIM statement the indices must be reals or variables, expressions are not allowed.

User arrays may be deleted from the symbol table by the command **DELA**, for example, DELA A[] deletes user array A[]. All user arrays may be deleted by the command **DELA_ALL**. The command **SHOW_A** prints a list of all arrays.

If the DELA or DELA_ALL commands are used in a command file and any of the arrays deleted are declared again later in the same command file, e.g with other sizes, a semicolon must be inserted between the delete commands and the DIM commands. Similarly, if a new declared array is refered to in the command file where it is declared, a semicolon must be inserted after the DIM statement. This ensures that the delete commands are executed and the array symbols removed before the DIM statements are compiled and that the new array symbol is known by TASCOM before it is refered to. About the semicolon separator, see page 42.

Operations on unsubscripted arrays.
Unsubscripted arrays can be used in the following arithmetic operations. A[], B[] and C[] are assumed to be arrays with the same dimensions:

A[] = B[] + C[]	addition
A[] = B[] - C[]	subtraction
A[] = B[] * C[]	multiplication
A[] = B[] / C[]	division

in the above operations each element in B[] is operated upon with the corresponding element in C[] (same subscript (S)) with the arithmetic operator (+ - * /) and each result is stored in the element of A[] with the same subscript (S).

Similarly in

A[] = TAN(B[])	tangent
A[] = ABS(B[])	absolute value
A[] = EXP(B[])	e raised to the power of element
A[] = LOG(B[])	natural logarithm
A[] = LOG10(B[])	logarithm base 10
A[] = SQRT(B[])	square root
A[] = -B[]	negation

the specified functions is executed on each element in B[] and each result is stored in the element of A[] with the same subscript.

Mixed type operations can be executed:

A[] = B[] (+ - * /) D

where A[] and B[] are assumed to be arrays with the same dimensions and D is a real variable. The specified operation, addition, subtraction, multiplication, or division is executed using D and each element of B[]. The result of each operation is stored in the element of A[] with the same subscript. For example,

A[1...(X+10)] = B[1...(X+10)]*SIN(30).

The **SUM** function calculates the sum of the elements in an array,e.g. D = SUM(A[]) calculates the sum of all elements of array A[]. The command **FICO** may be used to find array elements greater than a specified minimum. For example, FICO A[],10 lists all elements of array A[] greater than 10.

Operations on subscripted arrays.
Array elements can be used as real variables in all arithmetic operations and functions. For example, if A[] is an array and B and C are real variables

```
A[1,56] = B + C
B = A[1,56] * C
B = SIN(A[1,56])
```

Array elements may also be used in logical expressions like

IF A[1,56] > 1000 ....

Array elements can replace real variable in internal or external TASCOM commands, like in the following statement

AFIP M1, A[1,56]

If the subscripts used are real variables, then rounding and integer conversion takes place before the subscript is used.

Storage of arrays.

Each time an array is defined or declared by means of the DIM statement, memory is allocated to hold the array elements, and the array name and dimension is put on the symbol table. When TASCOM is exited, the symbol table is stored in a file. The values of all the array elements for each array are stored in a separate file with the same name as the array and the extension .arr in subdirectory /usr/users/tasfiles/array/ for system arrays and in /usr/users/tascom/array for user arrays. When TASCOM is again invoked, the symbol table along with all the arrays are read in again.

When a user array is deleted, the name of the array is removed from the symbol table and the memory that was occupied by the array is freed for other use. The array file with extension .arr in subdirectory /usr/users/tascom/array/ is also deleted.

5. Motor control

A motor can be moved by assigning the motor name a new value, e.g. **M1=10**. This assignment results in TASCOM performing a check of whether the new position is outside the lower and upper software limits of M1, **LM1** and **UM1**, set by the user. If so, an error message is returned, otherwise the motor is moved. The status variable **POTE** determines whether the test of compatibility with the limits is performed. Normally POTE=1 and the test is performed, but by setting POTE=0 the user can override software limits. If the status variable **OUPO**=1, a warning is written if the desired position is outside the limits.

By setting the motor fix parameter to 1, it is possible to lock a motor, so that it cannot be moved by a software command. This is done with the _SMPNA command, see later. If an attempt is made to move a fixed motor, a warning is written provided OUPO=1 and POTE=1. If POTE=0, the motor is moved even if the fix parameter is set.

Soft limits and the fixed motor feature are valid when motors are controlled by TASCOM commands, not when a motor is run from the manual box. With the manual box a fixed motor may be moved and a motor may be moved past a soft limit, but not past a hardware limit.

Motor movement can be aborted at any time by **control-C** or by pressing the ERROR-ON switch on the motor drive system.

Motor movement is aborted if the motor setting time-out is exceeded. This is specified by **_M_TIMO** in seconds. If _M_TIMO=0 or if the variable _M_TIMO does not exist, the motor setting time-out feature is disabled. The number on the motor(s) aborted is returned in the variable M_ABORT1 and M_ABORT2 for the first two motors aborted. The motor setting time-out feature is useful for axis with encoder.

TASCOM takes motor values from the ECB motor modules rather than from the symbol table or the motor control block, avoiding any possible confusion after a motor movement abortion, or use of the manual motor box.

It is often necessary to redefine the software value of a motor position, e.g. during lineup of the instrument. The command **REMO** does this, e.g.

REMO M1,1000

redefines the current position of M1 to 1000.

With the command **M_SAVE** the present motor positions are saved on a file in the user command directory, e.g.

M_SAVE ’fname’

saves the positions on file fname.tas. The motor positions are saved in ‘REMO’ form, i.e. each line in the file has the form

REMO motor-name, position.

The motor positions from the file fname.tas may be reloaded with the command fname that reads the file and REMOs the motors one by one.

With the command **L_SAVE** the present soft limit positions are saved on a file in the user command directory, e.g.

L_SAVE 'fname'

saves the positions on file fname.tas. The soft limit positions from the file fname.tas may be reloaded with the command fname.

Motor positions and soft limits may also be saved with the command **_SYMBOL** that saves all TASCOM variables on a file, see page 65.

The command **MOPO** outputs the current positions of all motors. The command **LIPO** outputs a list showing for each motor the soft limits, i.e. the value of L<motorname > and U<motorname>. In addition, LIPO indicates if the upper and lower hardware limit switches are open so that the motor is unable to move in the direction indicated. The symbol _ / _ is used to indicate an open hardware limit switch.

In some cases, it is efficient to move several motors simultaneously. This is done by first defining the final positions of all the motors to be moved using the command **NEWP**, for example:

NEWP M1,1000 NEWP M2,2000 NEWP M3,3000

Then using the command **MOVE**. No motor moves until MOVE is executed. Then all the motors defined by NEWP starts to move if they can; if some of the motors share a power supply as may be the case with Riso motor drives, these motors will run one at a time.

Normally, while motors are running, TASCOM is busy and other commands cannot be performed. However, the command **MSTA** allows one to start a motor running, and return to the TASCOM > prompt mode. This can be useful, for example while moving one motor a very long distance. The syntax is:

MSTA M1,10

The motor can then be stopped at any point using the command **MSTO**. The syntax is:

MSTO M1

Note that the backlash compensation is not working with the MSTA command since that internally in TASCOM requires two motor settings, one to the wanted position plus or minus the backlash compensation followed by one to the correct position. Only the first setting is done with the MSTA command.

Scan with one or two motors :

To scan with motor A and B use one of the two command files sma.tas (Scan Motor A), or smab.tas (Scan Motor A and B). These command files require as input: The motor name(s) as indirect variables, start motor position and end motor position, number of points in the scan, and preset time or count for the preset unit.

Example 1:

Scan motor OM from position 100 to 200 in 11 steps with preset 1:

SMA

```
MOTOR [@2T] = @OM
MBEG [-1.5] = 100
MEND [2.25] = 200
NPOI [11] =
PRES [100] = 1
```

The values in square brackets are the present assignments to the variables. If a car return is typed as the first and only character after equal sign, the value of the variable is unchanged.
The same scan may be specified in one line:

```
SMA @OM,100,200,11,1
```

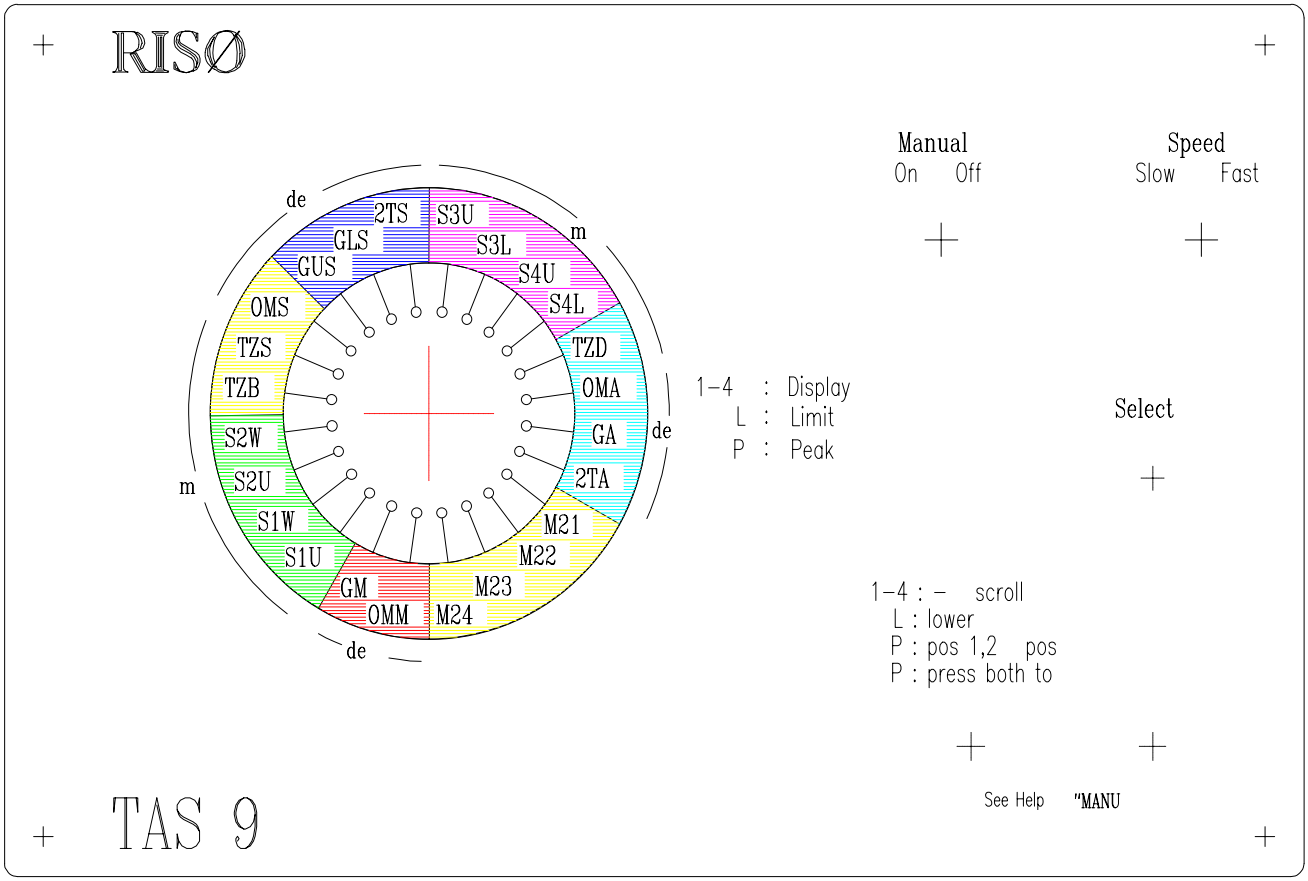
Example 2:
Scan motor 2TM from position -5 to 5 and motor OM from position 0.5 to 10.5 in 7 steps with preset 2.5:

```
SMAB @2TM,-5,5,@OM,0.5,10.5, 7,2.5
```

In the above commands the @-sign is optional. The VAR: command at the start of the files sma.tas and smab.tas does expect indirect variable names but if the @'s are missing, they are added by the VAR: command.

6. Manual control box and video monitor

The front panel of the manual control box is shown in the figure below. Motors are set up to manual mode with the Manual button or switch on the panel. The desired motor is then selected with the knob, and driven with the joystick. The Speed switch allows one to change between a low speed range, typically 1 - 40 steps/sec., and a high speed range, e.g. 350 - 3000 steps/sec. The minimum and maximum speeds in the two ranges are set with the _SMPNA command, see later, and choosen so that the motors are not running faster than they can and not running at a possible resonance speed. The position of the motor selected with the manual box is shown underlined at the first line on the video display.



The box has the following modes:
C-mode used to change manual selected octal motors on P2324.
L-mode where one can set software limits.
P-mode where one can input limits for a standard motor scan and execute the scan.
(1-4) mode determines which four motors or scalers are dispayed on the video monitor.
These modes are selected by toggling through the options C L P 1 2 3 4 on the LED indicator using the Select button.

Motors connected to the P2324a octal motor drive may be run from the manual box by selecting the octal drive with the knob on the box. This selects the one motor already enabled on the octal drive out of the eight motors that may be connected to the octal drive. To enable another of the eight motors, select the C-mode with the select button and use the + or - button to scroll through the eight motors connected to the octal drive until the wanted motor is shown on the monitor.

In L-mode, the user moves the manually selected motor to the lower (upper) limit and presses the button 'lower' ('upper') to set this limit.

In P-mode, pressing button 'pos 1,2' prints and saves the actual position of the motor selected with the box. First time the button is pressed the position is saved in the variable **PPO1**, next time in **PPO2**.

Also in P-mode, a simple standard scan may be defined and executed. Move the motor to the scan start position and press button 'pos 1,2,peak'. The intensity is counted for 2 sec. Then move to the scan end point and press button 'pos 1,2,peak'. Again the intensity is counted for 2 sec. Then TASCOM moves the motor to the midpoint , count for 2 sec., and prints the result.

The variables PPO1 and PPO2 contain the most recent positions chosen in the P-mode. The midpoint is **PPOS** and the measured intensity is **PINT**. The measured intensity PINT can come from one of the three scalers, as determined by **PRDE**, so that if PRDE=1, 2, or 3 then MON, I, or AUX is measured, respectively. This procedure is defined in the command files cfkey1.tas and cfkey2.tas.

To execute the standard scan in P-mode, press both buttons at the same time. This will run the command file cfkey7.tas in the system command directory. The cfkey7.tas file defines a scan from PPO1 to PPO2 and is listed in the appendix. The user can copy cfkey7.tas to his command directory and edit the file, for example to set the number of points in the scan, or the counting time. To copy the file, cd to your command directory and type:

cp /usr/users/tasfiles/command/sys/cfkey7.tas .

The last .(dot) in the cp command refers to the current directory, i.e. your command directory. Please do not make changes in cfkey7.tas in the system command directory.

Video monitor. Associated with the manual control box is a video monitor with a four-line display. To set up the display, choose the line of the display (1-4) using the Select button and then toggle forwards (backwards) through the list of motor names, scalers and ratemeters using the button 'scroll+' ('scroll-'). These variable names are defined in the

display parameter file, e.g bw2_dis.tas for bw2, see an example in Appendix C. Note that the ratemeters, e.g. RmS, are not TASCOM variables as such, but only appear on the screen. Note that when the Manual switch is in the On position, the motor chosen on knob always appears underlined at the first line of the display.

Alternatively, the display may be set up from Tascom with the command **DIS_UNIT**. The syntax is

DIS_UNIT line1, line2, line3, line4

where the arguments are the display unit numbers from the display parameter file. For example, using the display parameter file in Appendix C, the command

DIS_UNIT 5, 6, 17, 11

results in the timer TIM beeing displayed on line #1 on the monitor, scaler I on line #2, ratemeter RdM on line #3, and motor number 5 on line #4.

At the top of the video display, a box diagram indicates which motors are running (filled boxes), and which motor is selected on the manual box. In addition, up to three running motors may be shown on the three bottom lines of the display, overwriting - while they are running - the units otherwise selected on these lines. If this feature is wanted, set the symbol **DISP_RM** = 1. If DISP_RM = 0 or undefined, running motors do not overwrite the lower lines on the display.

It is possible to move motors by normal motor assignments at the TASCOM terminal even when the manual box is in manual mode. This is done by assigning **DMAN**=0 which disables the manual box. Control is restored to the manual box by assigning DMAN=1. In the standard files which are executed when the manual box is in P mode, the assignment DMAN=0 is made at the beginning of the scan and DMAN=1 at the end.

The action TASCOM takes when one of the switches on the manual box is pressed may be programmed by the user. However, reprogramming the switches might make the above description obsolete and should be done by the system manager only. How the switches are programmed is described in the appendix (page 70).

Hardware motor limit switches. The motors stop when a hardware limit switch is reached, while the TASCOM software limits are bypassed when motors are in manual control. If a hardware limit is reached while moving a motor, this is indicated by an arrow sign next to the motor name on the display. An up-arrow indicates that upper limit switch is activated, a down-arrow indicates that lower limit switch is activated. Up-arrow and down-arrow at the same time normally indicates the presens of the ERROR signal from the group of limit switches connected in series, i.e. at least one of these switches is activated.

If an up OR down arrow is shown, then the motor may be backed away from the limit switch, in the direction opposite to that of the movement which caused the limit switch to be activated. Also, such limit switches affect only one motor each.

Limit switches of the ERROR type prevent movement of any motor in any direction. How, then, to move off an ERROR limit? For this one can use the pushbuttons "Bypass error" and "Error-on" on one of the limit switch modules in the motor drive racks.

Holding down "Bypass error" allows one to move the offending motor off the limit switch. It also allows one to move the motor in the opposite direction: Further into the limit! Thus the pushbutton "Bypass error" should be used only with the utmost care and the motor movement should be observed visually.

By pressing and holding down the pushbutton "Error-on", any ongoing motor movement can be stopped, e.g., in an emergency.

7. Motor parameters

The motors implemented in a given spectrometer set-up are specified in the ASCII symbol table file for that set-up. Also, for each set-up there are:

1) A multiplexer set-up file specifying the P2324a octal motor multiplexer set-up (not used at BW1 and W1 at present).
2) A gearing file specifying motor parameters such as speed, accelerations, etc. for the ECB controlled motors.
3) A display set-up file for the ECB video display.

The following is a list of ASCII symbol table files, gearing files, multiplexer set-up files and display set-up files for BW1, BW2, and W1:

Spectrometer	Symbol table	Gearing	Display	Multiplexer
BW1	bw1_sym.sor	bw1_gea.tas	bw1_dis.tas	
BW2	bw2_sym.sor	bw2_gea.tas	bw2_dis.tas	bw2_mux.tas
W1	w1_sym.sor	w1_gea.tas	w1_dis.tas	

Symbol table files are found in directory /usr/users/tasfiles/symfil/ in the system managers log-in account and are loaded with commands like _LO_S'bw1_sym', that loads the ASCII symbol table file for the BW1 spectrometer. Note that loading an ASCII symbol table file resets TASCOM variables to default values, see page 64.

Multiplexer files, gearing files and display set-up files are .tas command files and are found in directory /usr/users/tasfiles/command/sys/. They are loaded by typing their names without the extensions.

Motor parameters such as the acceleration and speed of the motor can be set with the Set Motor Parameter commands **_SMPNA** or **_SMPNU**.

The values of the parameters can be read to the variable VDIG with the Get Motor Parameter commands **_GMPNA** or **_GMPNU**.

The syntax is:
_SMPNA motor-name , 'parameter-name' , value
_SMPNU motor-number , 'parameter-name' , value

_GMPNA motor-name , 'parameter-name'
_GMPNU motor-number , 'parameter-name'

parameter-name is one of the following:

START	Speed in steps/sec at start of acceleration in fast speed range
MAX	Maximum speed in steps/sec in fast speed range
AUTO	Speed in steps/sec in auto mode and slow speed range
MAN	Speed in steps/sec in manual mode and slow speed range
ACC	Acceleration/deceleration time
RANGE	Speed range. Fast=1, Slow=0.
ENC	=0 for no encoder, encoder number if axis has encoder
CONT	=0 if no control signals used
ROT	Direction of rotation
RACK	Motor drive rack number
DELAY	Start delay time
TOL	Position tolerance
DEGREE	Step per degree
DIG	Degree per encoder digit
BACK	Backlash
ULIM	Upper software limit (equal to U<motor-name >)
LLIM	Lower software limit (equal to L<motor-name >)
OFFSET	Offset
FIX	Motor fixed if =1
NUMBER	Only with _GMPNA; returns motor number

For example
SMPNA CHI, 'START' , 200
sets the start speed for motor CHI to 200 steps/sec.

A list of the motor parameters for all motors can be obtained by writing **LMPAR** to get all the parameters, **LMPAR1** for the first group of parameters, **LMPAR2** for the second group of parameters, and **LMPAR3** for the third group of parameters. These commands may have a motor name as argument, e.g.
LMPAR M1
lists all motor parameters for motor M1.

Most motors have resonance ranges and will loose steps if they run at speeds in these ranges and of course all motors have a maximum speed above which they do not move at all. Thus it is often a delicate task to change the motor parameters and it should not be done whitout consulting the system manager.

The gearing of the motors can be set by the user, so that for example a goniometer position is given directly in degrees, rather than in motor steps. For example,

_SMPNA M1, 'DEGREE' , 123.4

sets a conversion factor of 123.4 steps/degree for motor M1. On the video display and in TASCOM such motor positions will appear as decimal numbers.

Loading the step per degree conversion for a motor with the commands _SMPNA or _SPMNU also sets the output format FORMAT for that motor according to the following table:

Conversion factor s steps/degree	Format
s = 1	%12d
1 < s ≤ 10	%12.1f
10 < s ≤ 100	%12.2f
100 < s ≤ 1000	%12.2f
etc....	

The format may be changed by the user with a FORMAT command following the _SMPNA command, e.g. to get more decimals in the output.

Another parameter which it is occasionally necessary to change is the backlash parameter. When travelling in the negative direction, motors are normally made to overshoot their destination by a fixed backlash, in order that final positions are always approached in the positive direction, to eliminate the effects of mechanical backlash. The parameter may be set for e. g. motor 2TM as

_SMPNA 2TM , 'BACK' , 0.25

The backlash can also be negative (approach in negative direction).

The motor parameters are saved on the motor control block file (a binary file) when TASCOM is exited, and are reloaded from the file when TASCOM is started. A standard set of motor parameters are available on the gearing files for the different spectrometer set-up, e.g. bw1_gea.tas for the BW1 spectrometer loaded with the command BW1_GEA. An example on a gearing file is included in this manual as appendix A. Note that parameter values changed with the _SMPNA or _SMPNU commands are lost when the parameters from the gearing file are loaded. This is due to the fact that the gearing file calls mpar1.tas and mpar2.tas that contains a number of _SMPNU commands that overwrites ealier _SMPNA or _SMPNU commands.

With the command **SMPAR** the present motor parameters from the first two groups of motor parameters are saved in ASCII form on a file in the user command directory (the first one specified if more than one user directory), e.g.

SMPAR 'fname'

saves the parameters on file fname.tas. The motor parameters may then be reloaded by typing the name of the file.

The motor parameters are set to default values with the command **_INIMCB**. The parameters are also set to default values when a new symbol table is loaded with the command _LO_S, see the appendix page 64. The default values are defined internally in

TASCOM and are normally not the same as the values in e.g. bw1_gea.tas. Thus following a _LO_S command, the gearing file should be loaded.

Multiplexed motors. In instrumentations using the P2324 Octal motor drive that can multiplex eight motors from one ECB motor control output, the parameter **MULT_MOT** must be set to MULT_MOT=1. If multiplexing is not used, MULT_MOT=0. After the value of MULT_MOT has been changed, TASCOM must be exited and restarted before the new value is applied. A list of which motors are multiplexed may be obtained with the command **LPORT**. The motor multiplexing set-up file is described in the appendix, page 74.

8. Counting, scalers and ratemeters

The ECB timer/scaler module has a timer and three inputs for scaler readings (counts). In TASCOM, these units have the names

0) **TIM**
1) **MON**
2) **I**
3) **AUX**

(timer, monitor intensity, detector intensity, auxiliary intensity). On completion of a counting started with, e.g. the COUN command (see below), these variables are read from the ECB timer/scaler module to the symbol table. When asking for one of them, e. g. ?MON, or when they are included in one of the output buffers, their values are taken from the symbol table.

It is possible to measure the detector intensity for a fixed time in the timer or for a fixed number of counts at one of the scalers. The status variable **PRSC** defines the counting mode:

PRSC=0 Preset is time (ECB clock)
PRSC= *i* Preset is taken from scaler number *i*.

So PRSC=1 takes preset from the monitor. The variable **PRES** defines the preset counting time, either as real time in seconds (PRSC=0) or in monitor counts (PRSC=1).

It is possible to read the timer and scalers while they are counting rather than wait for a scan to complete. The variables **F_TIM**, **F_MON**, **F_I**, and **F_AUX** may be used to read the timer and scalers directly from the ECB timer/scaler module, e.g. ?F_TIM reads the timer.

Some systems have four additional scalers. These scalers are named **I4**, **I5**, **I6**, and **I7**. They can not be used as preset scalers but are started with the commands that start the other scalers and stopped when the other three scalers are stopped. They may be read while counting with the commands **F_I4**, **F_I5**, **F_I6**, and **F_I7**.

The **COUN** command counts for the given preset, and reads the scalers. The command **COST** makes it possible to start counting and immediately return to command mode so that other commands may be executed while counting. The counting continues until preset is reached or control-C is typed. The variable **COFL** may be used to check the state of the scalers; it returns COFL=1 if the scalers are counting and COFL=0 if the scalers are stopped.

Counting may be stopped at any time by typing control-C. If the variable **_ASK_CC** is _ASK_CC=1, the user is asked whether the counting is to be restarted, continue, or aborted. If _ASK_CC=0, the counting is immediately aborted when control-C is typed.

The scaler variables measure a total intensity, but it is often useful to see the intensity per second on the video display. TASCOM can convert counts into a rate internally, and shows these on the video display. The names of these ratemeters, which can be included on the display set-up file, are:

RdS, **RdM**, **RdF**, **RmS, RmM**, **RmF**, **RaS**, **RaM**, **RaF**.

These are not TASCOM variables, but only appear on the video display. The ratemeter shows the intensity per second at the detector, monitor or auxiliary (d, m or a), by measuring the counts (I, MON or AUX) per second. The averaging time of this measurement can be 4 sec (Slow, e.g. RdS), 1 sec (Medium, e.g. RdM), or 0.25 sec (Fast, e.g. RdF).

The rate meter can be enabled all the time by assigning **RATE**=1 which means that the scalers runs all the time, i.e. when preset is reached the scalers stop while they are read and are restarted immediately after. If RATE=0, the ratemeter is only enabled during counting, i.e. the scalers are not restarted after they are read-out. If RATE=-1, the ratemeter is only enabled during counting but when TASCOM returns to prompt mode, e.g. when a .tas command file is finished, RATE is set to RATE=1.

The command **WAIT s** makes TASCOM wait for a specified number of seconds, s, e.g.
WAIT 10.23
waits for 10.23 seconds.

The symbol **TIME** holds the time in seconds since 1 January 1970.

The command **GET_TIME** updates the TASCOM variables YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND from the system clock.

The variable **TFRQ** determines the frequency of the internal timer in the ECB system. Normally TFRQ=1000, but if PRES is set to a value greater than 4 hours, the assignment TFRQ=10 should be made. If the variable **_CH_TFRQ** is defined and _CH_TFRQ >1, the system sets the timer frequency depending on the preset value of the timer. If, for example, _CH_TFRQ=1234, then setting the timer preset to a value less than 1234, sets TFRQ=1000, while setting the timer preset to a value greater than 1234, sets TFRQ=10. Note that the accuracy of the timer is one timer clock pulse. If e.g. TFRQ=10 and the timer is used as preset timer with a preset of 1 sec., the uncertainty of the preset is 10%. Therefore, with a low preset always use TFRQ=1000.

9. Data storage

Data from TASCOM scans is stored on files with the extension .dat. The files contain only ASCII characters. In .dat files, motor positions, intensities etc. can be stored at every step of the scan. The data from each step in the scan appears in the file on one line.

The filename for .dat files is constructed from a name defined by the character variable **FINA** (e.g. FINA='test') and a segment number, defined by the variable **SEGN**. The name plus the segment number must not exceed 8 characters which is the maximum length of DOS filenames. If it does exceed 8 characters, FINA is shortened, e.g. if FINA='abcdef' and SEGN=1234, the filename will be abcd1234.dat. If name plus segment number is less than 8 characters, zeros are filled in between the name and the segment number, e.g if FINA='ab' and SEGN=12, the filename will be ab000012.dat. All TASCOM command files that store data should contain the assignment SEGN=SEGN+1, since this is used by TASCOM to determine whether a new file should be opened. The new file is actually opened when the first OUT call (see below) is made after changing segment number. Open data files are closed when a new one is opened, with the command **CLOSE_F** or when the TASCOM prompt (>) is output. Data files are also closed when PLOT is called.

The variable **_OW_DAT** decides what happens if a file with the name constructed from FINA and SEGN already exists. If _OW_DAT = 1, the file is opened thereby overwriting the existing data file that is lost. If _OW_DAT = 0, SEGN is incremented until an unused data file name is found and a file of that name is opened.

There are four output buffers in TASCOM defining which variables are output at each step, ie at each OUT command. Output to the terminal is controlled by the print parameter buffer, PRP, output to the data file is controlled by the file parameter buffer, FIP, and output to the file header at the beginning of each data file is controlled by the file head parameter buffer, FIH. The fourth buffer, FIA, defines output to array data files, see later.

The variables currently in the output buffers may be listed on the terminal with the commands **LPRP**, **LFIP** and **LFIH**.

Variable names are placed in the output buffers with the commands **DPRP**, **DFIP,** or **DFIH.** The syntax is, for example:

DPRP M1,I,M2,USER,FINA

Note that also string variables, e.g. FINA, may be output. Undefined variables specified in an output buffer will be defined as user variables with the value 0. The DPRP, DFIP and DFIH commands automatically clear the output buffers before filling them.

New variable names may be appended to an output buffer with the commands **APRP**, **AFIP**, or **AFIH** commands, for example:

AFIH MON,AUX, A[12]

All appended parameters can be removed from the buffers by the **REAP** command. The command removes appended variable names in all the output buffers but not variable names placed in the buffers with the DPRP, DFIP, or DFIH commands.

On exiting from TASCOM, the current parameters in these buffers are stored on file; the buffers are restored from the file when TASCOM is started.

The command **FICA** lists some of the buffers and file names, and some other parameters.

With an assignment to **DFIT** it is possible to define a text string that is stored at the beginning of all data files. For example:

DFIT = 'This is a backround scan'.

In addition to the variables specified in the file header buffer, all motor positions are output in the file header if the variable **MTOU** = 1.

The command **OUT** writes the variables the user has defined in the output buffers to the screen and the data file. In scan files, OUT must be executed at every step in the scan loop. The effect of OUT can be switched on and off using the status variables **FIOU** and **PROU**. If FIOU=1, data is stored on file, if FIOU=0, no data is stored on file. Likewise, if PROU=1, there is output to the screen, if PROU=0, there is no output to the screen.

Subdirectories are used to store output data files, so that, for example, different users can keep their data separate. The string variable **DATA_DIR** contains the current user subdirectory where the data files are stored. The path specified in DATA_DIR will be appended to the /usr/users/tascom/data subdirectory. For example if DATA_DIR='alpha/beta/gamma' then data is stored in the subdirectory /usr/users/tascom/data/alpha/beta/gamma. Note that the directory must be created by the UNIX/Linux command mkdir before writing data to it.

The following is an example on a data file:

```
#fdt     data/mydata/tes00012.dat 23/11 2001   14:18:59
#txt     Background scan
#cmd     SMA,2TA,-7,11,7,2
#com     Plot variables:
#plv     2TA      I
#com     Plot parameters:
#plp     YMIN  YMAX  XMIN  XMAX  BACK  LIMY  NBAC  PLTY  ERR_BAR
#pls        0     0 -0.05  1.05  0.15     0     0     0        0
#com     File head parameters:
#fih            M1   A_DAT[12]        TIM     FINA
#fhp          0.00         480    100.000      tes
#com     File parameters:
#fip         2T         OM      2TA        I
        -49.670    264.640    -7.00       12
        -49.670    264.640    -4.00      111
        -49.670    264.640    -1.00      637
        -49.670    264.640     2.00     1020
        -49.670    264.640     5.00      877
        -49.670    264.640     8.00       98
        -49.670    264.640    11.00        5
```

Note the prefix on the lines in the data file. These prefix are used by calculation programs to distinguish between comments (prefix #com), the file text (#txt), the last keyboard command (#cmd), the plot variables from the DPLO command (#plv), some parameters

useful for later plot of the data (#pls), the date and time line (#fdt), the file header parameters (#fih) and their value (#fhp), and the file parameters (#fip).

The line starting with the #cmd prefix may be changed with the **SAVE_LIN** command. A SAVE_LIN command in a .tas command file will result in the following command line in the command file being saved for output in the data file after the #cmd prefix. Note that the line is saved at the time the commands are compiled while output to the data file is when the commands are executed. Therefore, if several SAVE_LIN commands are used in the same command file, semicolons must be inserted to separate the SAVE_LIN commands.

The format width of the variables in the output buffers should not be less than the number of characters in the variable name plus one, plus - for array elements - the number of characters in the indices. E.g. in the example above, the format for A_DAT should be %10d; if the width is less, e.g. %7d, there is not enough room on the #fih line for the whole array element name A_DAT[12].

Usually, when a scan is finished, a new scan is started and a new data file is opened. Sometimes, however, it is desirable to continue a scan and add more data to the already closed data file. This is possible by setting the variable **REOPEN_F** =1. A new data file is normally opened by the OUT command following an assignment of a new value to the variable SEGN, e.g. by the command line SEGN=SEGN+1 in the standard scan command file sma.tas, but if REOPEN_F=1, the assignment to SEGN will leave SEGN unchanged. Instead, at the next OUT command, the old data file will be opened for append of more data and at the same time REOPEN_F is set to REOPEN_F=0 so that following assignment to SEGN will result in new data files being opened.

Output of arrays. An array or a fraction of an array may be output to the screen or to a .dat file using the normal OUT command. In this case all specified elements of the array are output on the same (may be very long line) as the other parameters in the respective parameter buffers. For example,

DFIP OM, A[6...15] or

DPRP M1, AUX, A[]

In the latter case all elements of array A are output.

Large arrays may also be stored in a separate data array file in ASCII form. Data array files have extension .dar. The filename is constructed as described above for data files, but uses the variables **FIAR** (e.g. FIAR='test') and a segment number, **SEAR**. The arrays to output are defined in the file array parameter buffer, FIA, which are operated upon with the commands **DFIA**, define file array parameters, **AFIA**, append file array parameters and **LFIA**, list file arrray parameters. For example,

DFIA A[]

specifies output of array A[] to a data array files on each OUT command.

As with .dat data files, a variable **_OW_DAR** decides whether or not .dar files may be overwritten.

The output of arrays to data array files on each OUT command can be switched on and off using the status variables **AROU**. If AROU=1 arrays are output, if AROU=0 output is switched off.

Alternatively, the command **AOUT** may be used to output arrays specified in the FIA buffer to data array files. AOUT outputs the array data when AOUT is executed (not at the OUT command) and will produce output even if AROU=0.

Arrays are stored as ASCII files depending on the variable **FIBI** as follows:

FIBI value	Array type	File extension
-1	ASCII with format '%d'	dar
0	ASCII with format as specified for the array	dar

FIBI=-1 gives the smallest ASCII data array files since the elements are separated by one space only, but the elements are not grouped nicely in columns.

Log file and log printer. TASCOM allows the user to generate a log either on a printer connected the computer or on a file or both. The log printer is enabled with the assignment **LOGP** = 1 while LOGP = 0 disables the log printer. The log file is enabled with the assignment **LOGF** = 1 while LOGF = 0 disables the log file. LOGF = 1 and LOGP = 1 are also used to print a header on the log. The extension of the logfile is .log and its name is defined by the character variable **FINL**, e.g. FINL='my_log'. The logfile is to be found in the data subdirectory defined by the user. If **_LOGDIR** is defined and nonzero, the logfile is in /usr/users/tascom/data.

The log parameters are defined in a log buffer. To list what is currently in the log buffer, the command **LLOG** is used, to define which parameters are in the buffer, the command **DLOG** is used, and to append parameters to the buffer, the command **ALOG** is used. The syntax is, for example:

DLOG M1, I, M2, USER, FINA

String variables like FINA may be included in the buffer. All appended parameters are removed from the buffer by the REAP command. The parameters in the log buffer are logged each time the command **LOG_PAR** is given.

The log parameters are also logged after each scan, i.e. when the scan data file is closed, provided the variable **LOG_CLOS** is LOG_CLOS=1. Then, in addition to the parameters in the log buffer, also some scan parameters are logged: time, filename, scan variable, and some scan information. The scan variable that is logged is the first one defined in the DPLO statement.

The command **LOG**, e.g.

LOG AUX, TCOM_DIR, 'text'

will log the specified parameter(s) and text string(s) at the time the command is executed. All types of TASCOM variables may be logged with the LOG command.

The command **PLOG** is similar to the LOG command, but will in addition print the specified parameters in the Tascom window.

Comments may be added to the log with the command **COMM**. The command will enter a mode where lines typed on the terminal are printed on the log. During the command mode, the TASCOM prompt appears as <. Return to normal TASCOM mode is done when a * is typed as the first character on a line.

The REMO command also gives a message on the log.

A new header is logged after **LPAG** lines of log output.

The length of the log lines may be adjusted with the variable **LLINE**. 80 ≤ LLINE ≤ 200. If LLINE is not defined, the line length is set to 120 characters.

If a log printer is used with Linux on a PC, the PC serial port to which it is connected must be specified in the variable **PORT_LOG**. For example,

PORT_LOG = ’cua2’

10. TASCOM plot and Statistical Calculation

When a scan is finished, TASCOM can produce a simple plot and data analysis of the scan data by using the command **PLOT**. This command is normally included at the end of scan command files. The resulting plot is a simple ASCII plot that is output on the screen as well as on the printer (provided the printer is on). To generate the plot, TASCOM uses the data in the .dat file and the files plot.plt and plot.cmd.

The variable **PLOU** enables/disables the plot and data analysis so that if:

PLOU = 0	no plot and no data analysis
PLOU = 1	plot but no data analysis
PLOU = 2	no plot, data analysis only
PLOU = 3 or -1	both plot and data analysis

The command **DPLO** determines the x and y coordinates of the plot, for example:
DPLO M1, I
The status variable **LIMY** defines whether TASCOM should perform automatic scaling of the y-axis (LIMY=0), possibly with baseline 0 (LIMY=2). If LIMY=1, the limits for the y axis are defined by the variables **YMIN** and **YMAX**.

In addition to plotting the data in the simple plot mode, TASCOM performs some simple data analysis, on the assumption that the user is measuring a peak. The algorithms assume a constant background. The user can define which part of the scan is to be considered as background using the variable **NBAC**. For example if NBAC=2, the first two and the last two points of the scan are considered as background. Alternatively, the background may be defined with the variable **BACK**. If both NBAC=0 and BACK=0, no background correction is made. After the scan is completed, several variables contain information about the peak. **IMAX** is the maximum intensity attained during the scan. **CEMA** is the center of mass of the scan. **MIDP** is an estimate of the midpoint of the peak, as derived from the slopes of the peak. **FWHM** is an estimate of the full width at half maximum obtained from the slopes of the peak. **BGD** is the avarage background per point. **PEAK_A**, **PEAK_B** and **PEAK_H** are peak area, peak background and peak height, respectively.

These variables are printed out depending on the setting of PLOU. The data analysis may also be performed and printed with the command **DTAN**.

The data from the analysis can be used to simplify the task of typing in subsequent scans, for example, if a scan of motor M1 has just been performed, M1=MIDP will set the motor M1 to the midpoint of the peak and

SMA M1,MIDP-500,MIDP+500,11,1

will scan around the midpoint.

In addition to the simple plot a more detailed plot may be generated using the separate PLOT-program. For details see appendix P, page 88.

Statistical calculation.
When a scan is finished TASCOM can perform simple statistical analysis by calculation of the mean value and standard deviation on the measured data. The command **STAT**

calls the statistical analysis program and the calculated data are printed on the screen with one line for each variable specified.

The variables for which statistics are calculated are held in the STP buffer. The variables currently in the buffer may be listed on the terminal with the command **LSTP.** Variable names are placed in the buffer with the command **DSTP.** The syntax is, for example:

DSTP M1,I,M2

The DSTP command automatically clear the buffer before filling it. New variable names may be appended to the buffer with the command **ASTP.** All appended parameters can be removed from the buffer by the REAP command.

Note that variables in the STP buffer must also be included in the FIP buffer, e.g. with the DFIP command. The statistical calculation is done on the data in the data output file so in order to do the calculation on a variable, the variable must be in the data output file.

11. Command files

Sequences of TASCOM commands can be entered in command files and executed by simply writing the name without the extension of the command file. The name must be in lower case consisting of letters, numbers, or the underscore character, the first character being a letter or underscore. The extension must be .tas in lower case.

A set of command files for frequently-used scans is provided with the program, and are stored in subrirectories of /usr/users/tasfiles/command in log-in account tasfiles. These system command files can be inspected by the user, e.g. with the _TYPE command, but not changed. The string variable **SCOM_DIR** contains the name of the subdirectories where TASCOM will search for system command files. More than one subdirectory may be specified in SCOM_DIR separated by semicolons. In addition to what is specified in SCOM_DIR, TASCOM will always look in /usr/users/tasfiles/command/sys where some essential system files are kept.

Command files created by the users are in the user log-in account tascom and are stored in subdirectories of /usr/users/tascom/command. Each user should create a personal subdirectory in order to keep different users' command files separate. The string variable **TCOM_DIR** contains the name of the subdirectories where TASCOM will search for user command files. More than one subdirectory may be specified in TCOM_DIR separated by semicolons.

For example, if

SCOM_DIR = 'subdir1; subdir2/mydir'
TCOM_DIR = 'subdir3; subdir4/mydir'

then TASCOM will first search

/usr/users/tascom/command/subdir3,	then if the command file is not found
/usr/users/tascom/command/subdir4/mydir,	then
/usr/users/tasfiles/command/sys	then
/usr/users/tasfiles/command/subdir1	and finally
/usr/users/tasfiles/command/subdir2/mydir	

The subdirectories subdir1, subdir2, subdir3, subdir4, and mydir must be created in Linux with the mkdir command before assigning SCOM_DIR or TCOM_DIR to them.

The command **DIRTAS 'file-name'** lists all instances of the files file-name.tas in /usr/users/tasfiles/command/sys and in the subdirectories specified by SCOM_DIR and TCOM_DIR.

Scan variables. The **VAR:** statement is used in the first non comment line of a command file to prompt the user for values of certain variables. The variables are typically such scan parameters as start and end positions of scan, number of points and preset. The syntax is, for example:

VAR:MOTOR,M1BE,M1EN,NPOI,PRES

The VAR: statement must be the first non comment line of the command file. If the name of the command file is sma.tas, then by writing SMA alone, TASCOM will prompt for the variables one at a time, e.g.

MOTOR [@OMM] =

The present assignment to the variable is shown in square bracket after the variable name and is unchanged if a car-return is typed immediately after the = sign. Instead of having TASCOM to prompt for the parameters one by one, the user can write all (or the first few) parameters on one line, e.g.

SMA M1,100,200,11,1

This has the advantage that the command string can be retrieved and repeated by using the arrow keys, and editing the variable values if necessary. Note that mathematical expressions can appear as variable values *only* if they are written on the same line as the command file name, e.g:

SMA M1,M1-100,M1+100,11,1

If the VAR: line contains indirect variables as for example MOTOR in the motor control command file sma.tas, the only legal assignment to such indirect variables in VAR: lines are other variable names. Such an assignment would normally start with the @-sign. However, in the case of assignment to indirect variables in VAR: statements, the @-sign may be omitted. This syntax has been adopted in order to make it easy to use the standard scan files, and to avoid the situations where an unexpected motor starts moving if the user by mistake specifies a value for the motor rather than the name.

While the VAR: statement may take input from the keyboard or from another command file, the similar command **ASK_VAR** always takes input from the keyboard. It may be placed anywhere in a command file and asks the user to input one or more variables from the keyboard. The syntax is:

ASK_VAR 'string',VAR1,VAR2,...,VAR9

Up to 10 arguments are legal to ASK_VAR. The first argument may be a string that is printed on the screen when the ASK_VAR command is executed. If the two last characters in the string are \n, they are converted to newline. The string argument is optional. The following arguments must be TASCOM real variables or string variables. Arrays are not allowed. The user is prompted for the variables one by one. The user input to a string variable must be a string. The user input to a real variable must be a number or a real variable; expressions are not allowed. For example:

ASK_VAR 'Input following variables:\n', FINA, FIAR, H, K, L

The semicolon separator. When TASCOM receives a command, or a list of commands, e.g. from a command file, all the commands are translated into internal code that are stored in a node table. This compiled version of the commands in the node table is then executed. When a semicolon **;** is entered in a list of commands, the commands before the semicolon are compiled and executed. Then the commands following the semicolon are compiled and executed. If a fatal error occurs during the execution of the code before the semicolon, execution of the whole command file is aborted.

The semicolon should be used to avoid problems when invoking a series of large command files, to avoid overflow in the internal node table of TASCOM. The semicolon should then be placed after each reference to a command file. From Tascom version 3.08 the size of the node table has been increased from 32k words to 2M words.

A semicolon must never occur within DO-UNTIL, WHILE-NEXT, or IF-ELSE-END control statements as the whole body of the control statement must be compiled before the execution can start.

Interrupting command files. A running command file may be interrupted by control-C, in which case TASCOM returns to prompt (>) mode. A command file will also return to prompt mode if the command **PROMPT** is executed in the command file. With the command **BEFOR_CC**, the user may specify a command file that will be run before control-C returns, e.g. if

BEFOR_CC = 'xxx'

the file xxx.tas runs after control-c is typed but before the prompt mode is entered. The file specified in BEFOR_CC is also run if the command file is terminated with the PROMPT command or because of a fatal error, e.g. activation of a hardware limit switch while a motor is running. BEFOR_CC is useful if parameters in the beginning of a command file temporary are assigned some values, and are reassigned to the old values before the command file finished. If such a command file is interrupted by control-c or by a fatal error or left with the PROMPT command, the old values may be reassigned in xxx.tas.

Comments. In a .tas command file comments can be written by prefacing them with an exclamation mark **!**. TASCOM will ignore what follows on the line at compilation. By using the **?**, it is possible to output text and variable values to the screen from a command file, for example:

?' Motor ',M1

will output:

Motor M1 = 10.

Typing files. The command **TYPE** allows one to list command file or other files, e.g.

TYPE 'myfile'

types the file myfile.tas on the screen. If the name of the file is specified without a path, ie without a / character in the specification, the file is expected to be located in one of the TASCOM command file directories. In case of a TASCOM command file, the file may be specified with or without the .tas extension and the command directories are searched for the file in the order described above. The first file found that matches the specification is typed. If a path is specified, the file may be located in any directory and is typed if the user has permission to read the file. For example

TYPE '/usr/users/tasfiles/help/me.hlp ' or

TYPE 'bin/owltest'

The latter example will type the file /usr/users/tascom/bin/owltest.

The command **_TYPE** may be used similarly but types only files from the system command directories (i.e. from /usr/users/tasfiles/command/sys and directories specified by SCOM_DIR), for example

_TYPE 'sma' or
_TYPE 't100123.tas'.

Startup command file. It may be convenient for the user to give certain variable values or execute certain commands every time TASCOM is entered. These should be included in a .tas command file, and the variable **SFIL** gives the name of the file (e.g. SFIL='myst'). The file myst.tas must be in the subdirectory defined by the user with TCOM_DIR or in /usr/users/tasfiles/command/sys. When TASCOM is started and if a startup file myst.tas exists, the user is asked:

Do you want to run myst.tas [Y/N] N:

The default answer is normally N for no. If the variable **SFIL_DEF** is SFIL_DEF=1, the default answer is Y for yes.

Name checking. A TASCOM command file should not have the same name as one of the symbols in the symbol table. For example, a command file with the name out.tas will never be executed because when the command OUT is given, TASCOM will find the symbol OUT in the symbol table and therefore never look for a command file of that name. The command **_SCHECK** may be used to check if there exists command files in the system directory and in the user directory with the same names as symbols in the symbol table. _SCHECK lists the names of such files and if any of those command files are to be used, they will have to be renamed.

12. Error handling and various features

Considerable effort has been invested in making sure TASCOM will not crash when an error occurs in compilation or execution of commands. A number of error messages indicate to the user the source of problems. For example, TASCOM will indicate an undefined variable in a command file by the name of the variable, the line in which it appears, and the name of the command file. When a run-time error is encounted while TASCOM is executing a command file, the name of the command file and the line in the file is listed together with the type of error. Also, by nested command files, a trace-back through the files is listed. If **TRACE_B**=0, no trace-back is done. As well as error messages, the user can choose to have certain warnings output to the screen by setting the status variable **WARN**=1.

Some types of error are termed fatal, and abort a command file completely, even if the file is separated into parts by semicolons. Among such fatal errors are control-C typed by the user and overflow of the node table. Error messages may be printed on black background if symbol **COLOUR** is COLOUR=1, or with red characters if COLOUR=2. Normal background and characters are used if COLOUR=0. Error messages may be followed by a beep on the computer loudspeaker if symbol **BEEP** is BEEP= 1 or 2. If BEEP=2, a beep is given also at each prompt. The command **DO_BEEP** gives a beep. This may be used to produce beeps, e.g. at the end of a measurement.

The symbol table has a finite length, and if too many user symbols have been created, TASCOM will report that no more symbols can be created. In such cases, the user can erase user-defined variables, see the appendix page 65. The command FICA gives a list of some important variables and the amount of free space in the symbol table.

Calculation mode. It is often convenient to test certain diffractometer geometries without moving motors. For example to see whether a certain point in reciprocal space is compatible with all motor limits. TASCOM has a calculation mode, which can be invoked by setting the status variable **CALC**=1. In calculation mode, the TASCOM prompt appears as **C>.** All commands that normally move motors or set temperature or otherwise communicate with spectrometer hardware are ineffective in calculation mode, but symbol table values are updated. On exiting calculation mode by setting CALC=0, the original symbol table values are restored.

Unix/Linux commands. It is possible to execute Unix/Linux commands from TASCOM with the TASCOM command **UNIX_COM**. The UNIX_COM command may take Tascom variables or strings as arguments so that the Unix/Linux command is composed by all the arguments added together. The arguments may be separated by commas or added by plus-signs. To ease the use of UNIX_COM, the following four read-only Tascom variables may be included as arguments:

PATH_DAT	/usr/users/tascom/data/<value of DATA_DIR>/	
PATH_TAS	/usr/users/tascom/	
FILE_DAT	latest data file name	e.g. mine0022.dat
FILE_ARR	latest array data file name	e.g. mine0022.dar

For example, the following two commands show two ways of doing the same:

UNIX_COM 'cat ', PATH_DAT, FILE_DAT
UNIX_COM 'cat /usr/users/tascom/data/', DATA_DIR+ '/'+ FILE_DAT

Both commands will list the latest data file in the Tascom window.

If the flag **UNIX_FLG** is set to UNIX_FLG=1 and UNIX_COM is typed on the keyboard, the Unix/Linux command is printed in the Tascom window before it is send to the Unix/Linux shell. If UNIX_FLG=2, the Unix/Linux command is printed in the Tascom window even if UNIX_COM is included in a .tas command file.

If the flag **UNIX_CAL** is set to UNIX_CAL=0 or is undefined, the UNIX_COM command will only execute when CALC=0. If UNIX_CAL=1, the UNIX_COM command will execute also in calculation mode (CALC=1) and in sim_tascom.

OWL command. The OWL command can call an external program, transfer data from TASCOM to the external program and transfer data back from the external program to TASCOM. The external program must be located in subdirectory /usr/users/tascom/bin/. The syntax is:

OWL 'name-of external-program $arg1 $arg2 … $argn' ,x1,x2,…xm

where arg1, arg2,…, argn are tascom real variables the value of which are to be transferred to the external program, while x1,x2,…,xm are the TASCOM real variables that receive the data from the external program. Maximum number of arguments are 14.

Alternatively, the data from the external program may be transferred to a TASCOM array. The syntax is:

OWL 'name-of external-program $a1 $a2 … $an' ,A[]

The maximum size of the array A[] is 1024.

If the flag **OWL_FLG** is set to OWL_FLG=1 and the OWL command is typed on the keyboard, the string that the OWL command sends to the external program is printed in the Tascom window before it is send. If OWL_FLG=2, the string is printed in the Tascom window even if the OWL command is included in a .tas command file.

If the flag **OWL_CAL** is set to OWL_CAL=0 or is undefined, the OWL command will only execute when CALC=0. If OWL_CAL=1, the OWL command will execute also in calculation mode (CALC=1) and in sim_tascom.

Example:
The external program is called owltest and is the following Linux perl script that is located in /usr/users/tascom/bin

```
#!/usr/bin/perl
$newl="\n";
$a=$ARGV[0];
$b=$ARGV[1];
print $newl;
print $a*$b;
print $newl;
```

```
print $a/$b;
print $newl;
```

The script reads two arguments a and b and returns a*b and a/b. Called form Linux:
owltest 10 4
it returns 40 and 2.5

From Tascom the command owl is used like:
>a=10 b=4 x1=0 x2=0
>OWL 'owltest $a $b',x1,x2
After the owl command, x1 and x2 are set to the two values returned from owltest; x1 is set to 40 and x2 to 2.5

On-line Data Access. If the TASCOM web server is installed on the PC, TASCOM may be accessed from the Internet using a standard World-Wide-Web browser. This is achieved by pointing the browser to the URL (address)

http://hastasbw1.desy.de /

where "hastasbw1.desy.de " should be replaced with the hostname of the PC on which TASCOM runs. The use of this facility is largely self-explanatory. This feature is controlled by the variable **ODA_ON**. If for some reason remote access should not be allowed, it may be disabled by setting ODA_ON=0. It may be re-enabled with ODA_ON=1.

The ODA status window shows a number of Tascom variables. These are specified in the file (for BW1) bw1_oda.tas in /usr/users/tasfiles/command/sys. The file may be edited to get other Tascom variables. The file is loaded by typing the file name, e.g., bw1_oda, and the edited set-up will be used at the next update of the ODA status window.

Help feature. There is a **HELP** command in TASCOM which contains the quick-reference list and other subtopics. The list of subtopics is obtained by typing HELP. A given subtopic SUBT is output by writing HELP 'SUBT'. All subtopics exist as files with the extension .hlp in the directory /usr/users/tasfiles/help.

13. Temperature control

Risoe Digital Temperature Controller A1931a is used to control and read the temperature. The TASCOM commands for A1931a has the form **TC'command'**, where the command string is one of the legal temperature commands listed in the Quick Reference guide for the temperature controller. (Note however, that not all of the commands in the Quick Reference guide can be used from TASCOM.) If the command string starts with ?, a value is returned from the temperature control to TASCOM parameter VDIG, e.g. the command TC'?TEMP1' will return the temperature of thermometer 1 to VDIG. Some temperature commands do not return a value but a text string, e.g. the command TC'?VCC1' returns 'ON' or 'OFF'. In such cases VDIG is set to VDIG=0 if the string returned starts with the letters 'OF' or 'N'. If the string starts with any other letters, VDIG is set to VDIG=1.

A variable may also be specified outside the ' ', e. g. TC'SET1=',10 is identical to TC'SET1=10'. This may be used to do temperature scans with a TASCOM variable.

The variables TEMP*i* and SET*i* where $1 \leq i \leq 8$ may be used directly as TASCOM variables, i. e. without the TC command. Thus the commands TC'SET3=20' and SET3=20 will give the same result. The commands TC'?TEMP1' and ?TEMP1 both read temperature sensor number 1, but the first command stores the result in VDIG while the last command stores the result in TEMP1.

Set-up parameters for the temperature control are held in files named t<number >.tas in directory /usr/users/tasfiles/command/tfil and are sent to the temperature control with the command **TFIL=number**, e.g. typing TFIL=1234 will send the parameters in file t1234.tas to the temperature control.
Calibration parameters for the temperature control are held in files named c<number >.tas in directory /usr/users/tasfiles/command/cfil and are send to the temperature control with the command **CFIL=number**.

The command **PIDD** dumps PID parameters from the temperature control to a file from where they may later be reloaded, possibly after being edited. The reload command is **PFIL**. For example,
PIDD 1,'name'
loads the PID parameters from temperature channel #1 to the file name.pid in the users directory. The command
PFIL='name'
loads PID parameters from the file name.pid to the temperature control.

14. ON/OFF control.

On some systems an ECB relay module is included for control of ON/OFF functions. The module has 16 relay contacts that may be shifted with the command **SREL** and read with the command **RREL**.

The syntax is:
SREL number, onoff
where number is the relay number. 1 ≤ number ≤16
and onoff is =1 for relay on, =0 for relay off

RREL number
where number is the relay number. 1 ≤ number ≤ 16
RREL returns 1/0 if relay is on/off to the TASCOM variable **REL**.

Note:
Do not use relay number 9. Relay 9 is reserved for use on some spectrometers to control of the analyser shielding.

15. HP power supply control.

The Hewlett Packard DC Supply, series 66(4,5,6,7)xA may be controlled and read from TASCOM. The HP power supply must be set to GPIB address 1 or 2. The TASCOM commands has the form **HPS1'command'** or **HPS2'command'** for GPIB address 1 or 2, respectively. The command string in ' ' is one of the commands listed in the HP manual. A variable may be specified outside the ' ', for example the two commans HPS1'VOLT',4 and HPS1'VOLT 4' both sets the output voltage to 4. If the command asks for a return message from the HP, the return message is stored in array **HPA[]**. If the array does not exist, it must be defined by the user with

DIM HPA[80]

The return message is printed on the screen if the command is from the keyboard.

The routines controlling the HP is only included in TASCOM if the flag **HPS_FLAG** is set to HPS_FLAG=1. After a change in HPS_FLAG, TASCOM must be exited and restarted before the change is actually made.

Example: Ask for the HP Supply #1 volt setting.

```
HPS1 'VOLT?'
1.2500E+0          Return message printed if command was from keyboard
                   Array HPA[] may have a return message like
                   HPA[0]: 49   ASCII value of character 1
                   HPA[1]: 46   ASCII value of character .
                   HPA[2]: 50   ASCII value of character 2
                   HPA[3]: 53   ASCII value of character 5
                   HPA[4]: 48   ASCII value of character 0
                   HPA[5]: 48   ASCII value of character 0
                   HPA[6]: 69   ASCII value of character E
                   HPA[7]: 43   ASCII value of character +
                   HPA[8]: 48   ASCII value of character 0
                   HPA[9]: 10   ASCII value of character <newline>
                   So the return message from the device was 1.2500E+0
```

16. Multichannel analyser

The multi channel analyser is composed of the two NIM modules Canberra ADC-8075 and the Histogramming memory P2126a. Input pulses are fed to the ADC-IN connector on the ADC, converted by the ADC, and stored in the histogramming memory if the MCA is enabled.

The MCA can be controlled by the following commands:
DMCA disable the MCA
EMCA enable the MCA without clearing it
MCST enable and clear the MCA
CMCA clear the MCA
RMCA read the MCA

However, the MCA counting is normally started and stopped from the ECB timer/scalers. In that case the timer/scalers on/off-signal is connected to the GATE-IN connector on the ADC and the COINC/ANTI switch set in the COINC position. The sequence of commands that controls the MCA is then: Enable and clear MCA with the command MCST and start timer/scalers. When preset time or preset count is reached, the timer/scalers stops and the MCA are read with the command RMCA. RATE must be set to RATE=0; otherwise the scalers are restarted immediately after they are read thereby starting the MCA again before the MCA is read.

The command RMCA reads the MCA starting from the first channel. The data from the MCA is read to array **DD[]** (for Detector Data). DD[] is a user array which must be defined and dimensioned by the user before reading the MCA. For example,

DIM DD[4096]

defines DD as a one dimensional array that can hold 4096 MCA channels, while

DIM DD[10,10]

defines DD as a two dimensional array that can hold 10*10 MCA channels. If it is wanted to change the size of DD, the existing DD must be deleted with the command

DELA DD[]

before a new DD of the wanted size can be defined with the DIM command.

The total intensity, i.e. the sum of the counts from channel 0 to the maximum channel read, may be calculated as SUM(DD[]). The command **FICO** may be used to find array elements with counts greater than a specified minimum. For example,

FICO DD[],10

lists all channels with counts greater than 10.

The data read or part of the data is printed on the terminal with the OUT command if PROU=1 . DD[] or part of DD[], e.g. DD[20...30] for print out of the channels from 20 to 30 for a one dimensional array, must be included in the print buffer, e.g. with the command APRP DD[] or APRP DD[20...30].

If FIOU=1, the data is stored on the normal data output file with extension .dat each time the OUT command is executed. DD[] or part of DD[], e.g. DD[20...30] for the channels

from 20 to 30, must be included in the file buffer, e.g. with the command AFIP DD[] or AFIP DD[20...30]. All channels are stored on one, probably rather long line on the file.

If AROU =1, the data is stored on the array data output file. DD[] or part of DD[], e.g. DD[20...30] for the channels from 20 to 30, must be included in the array file buffer, e.g. with the command AFIA DD[] or AFIA DD[20...30]. The name of the array data output file is composed of the character variable FIAR (e.g. FIAR='mca') and the segment number SEAR which is automatically increased by one each time the OUT command is executed. The array is stored in ASCII form with extension .dar.

It is possible to do arithmetic operations on the data in DD[] and/or transfer the data or part of the data to other arrays and use these arrays in the output buffers, see the description of array operations in chapter 4.

The status variable **DD_DISP** determines whether the MCA data is transferred to the computer memory during the counting in order to produce a live display. If DD_DISP=1, the MCA is read approximately once a second when the counting is started with the COUN command. If DD_DISP=0, the MCA is not read during the counting. If DD_DISP=1, a live display may be shown on a screen window by using the PLOT-program, see appendix P page 88.

Another way of having the MCA data transferred to the computer memory is by using the **SHOW_DD** command that works even if the ECB scalers are not available. If called with no argument, SHOW_DD transfers the MCA data to memory as fast as possible until the user stops the reading with control-C. If called with an argument, the argument decides the number of MCA readings performed, e.g. SHOW_DD 10 reads the MCA10 times and then returns to the TASCOM prompt.

Set-up of the detector can be modified by changing the MCA parameters. The command **LMCA** lists the MCA parameters and their values. The command **_SETMCA** is used to modify MCA parameters. The syntax is

_SETMCA 'parameter' , value

For example,

_SETMCA 'MAP',2

sets MAP = 2. The normal set-up is MAP = 2 and MODE = M_X, where M_X = 1. All other parameters are set to zero.

17. Communication with ON_LINE.

The Hasylab spectrometer control program is called ON_LINE and is running on the PCs hasbw1, hasbw2 and hasw1 at BW1, BW2 and W1, respectively. Communication between TASCOM and ON_LINE takes place via tcp/ip.

Before the communication starts, ON_LINE must be set to server mode which is the standard network interface to ON_LINE. This is done with the command

SERVER 7777 /verbose

typed on the PC where ON_LINE runs.

The TASCOM commands for ON_LINE has the form **SPCM'command_string'** where the command string is one of the ON_LINE commands, see the ON_LINE manual. For example,

SPCM'energy()=8000'

will execute the ON_LINE command energy()=8000. A variable may be specified outside the ' ', e. g.

SPCM'energy()=',8000

is identical to SPCM'energy()=8000'. This may be used to do ON_LINE scans with a TASCOM variable.

ON_LINE returns either a value or the string 'done' for commands that completed successfully, 'error' otherwise. If a string is returned, it is stored in the TASCOM variable VSTR. If a value is returned, it is stored in the TASCOM variable VDIG. For example,

SPCM '*=energy()'

returns the value of energy in VDIG.

The first SPCM command send from TASCOM opens the communication with ON_LINE. The communication is terminated with the command

SPCM 'bye'

Two TASCOM variables _SP_HOST and _SP_PORT defines the PC and the port for the TASCOM communication. For example, for BW1:
_SP_HOST = hasbw1
_SP_PORT = 7777
They are read-only variables and should normally never be changed.

18. ADC control.

On BW1 an ECB analog-to digital module is included. The module has 16 analog input channels that are read with the command

ADC number

where number is the channel number, 1 ≤ number ≤ 16.
The ADC value read is stored in the variable VDIG and printed on the screen if the command was from the screen. The value is between -10 Volt and +10 Volt.

19. BW2 Floor

The floor on BW2 may be adjusted up or down with three motors M1, M2, and M3. Two Wyler level meters is used to control that the floor is horisontal. To maintain the horisontal orientation, the three floor motors must be moved simultaneously. If the slope of the floor exceeds 0.3 degree, the movement of the floor is stopped. Attempt to move only one or two of the motors result in an error message. The floor motors can not be moved with the manual box.

The Wyler level meter readings may be read with ?**FLEV1** and ?**FLEV2**. In addition, following one of these commands and provided the string variables **FLEV1STR** and **FLEV2STR** are defined, FLEV1STR and FLEV2STR will contain the full string returned from the level meters. Such a string may look like

980 A –0.003 O

where 980 is the number of the reading, A stands for Absolute (alternatively R for relative), -0.003 is the reading, and O is the unit.

The PC serial output port used by the level meters is defined in the variables _W1_PORT and _W2_PORT, for example

_W1_PORT = '/dev/cua4'
_W2_PORT = '/dev/cua5'

They are normally read-only variables and are only changed if the ports on the PC are changed.

20. HP Data Acquisition/Switch unit.

Communication with Hewlett Packard 34970A Data Acquisition/Switch Unit (HPD)

HPD_FLAG. Set HPD_FLAG=1 to enable control of HPD. After the HPD_FLAG has been changed, EXIT and reenter Tascom.

HPD_TIME. Command to set the HPD parameters date and time to the system computers date and time.

HPD 'HP-command'. Send commands to HPD. (HPD GPIB address must be 12). HP-command is one of the commands listed in the HPD manual. If the command asks for a return message from the HPD, the message is stored in array **HPDA[]**. If the first part of the message is a number, the number is stored in VDIG as well. If the array HPDA[] does not exist, it must be defined by the user with

DIM HPDA[80]

HPD_RFM. If HPD_RFM=1, data stored in the HPD memory is read when Tascom is idle. The data read is stored on a file as well as in the two dimensional array HPCH[]. If the array HPCH[] does not exist, it must be defined by the user with

DIM HPCH[48,5]

HPD_FILE. File name for data read when HPD_RFM=1. For example

HPD_FILE='mine'

creates a file mine.hp for the data in the users data directory.

HPCHnnn. String variable for assigning a name to channel nnn where nnn is a three digit channel number between 101 and 122 for the first group of channels and between 201 and 222 for the second group. For example,

HPCH105='AREA'

assigns the name area to channel 105. The name is output to the file together with the other channel data. If the assigned name is the name of a Tascom symbol, the symbol is updated with the measured value from the channel at each read-out.

Example #1: Set trigger.

```
HPD 'TRIG:COUNT 400'
        or
VALUE=400
HPD 'TRIG:COUNT ',VALUE
```

Example #2: Ask trigger setting.

```
HPD 'TRIG:COUNT?'
+4.00000000E+02  ! Return message printed if command was from keyboard
        ! VDIG is set to VDIG = 400
        ! Array HPDA[] may have a return message like:
```

```
!   HPDA[0]:  43   ASCII value of character +
!   HPDA[1]:  52   ASCII value of character 4
!   HPDA[2]:  46   ASCII value of character .
!   HPDA[3]:  48   ASCII value of character 0
!   HPDA[4]:  48   ASCII value of character 0
!   HPDA[5]:  48   ASCII value of character 0
!   HPDA[6]:  48   ASCII value of character 0
!   HPDA[7]:  48   ASCII value of character 0
!   HPDA[8]:  48   ASCII value of character 0
!   HPDA[9]:  48   ASCII value of character 0
!   HPDA[10]: 48   ASCII value of character 0
!   HPDA[11]: 69   ASCII value of character E
!   HPDA[12]: 43   ASCII value of character +
!   HPDA[13]: 48   ASCII value of character 0
!   HPDA[14]: 50   ASCII value of character 2
!   HPDA[15]: 10   ASCII value of character <newline>
```

Example #3:

```
AREA=0                ! Make Tascom symbol AREA if it does not exist.
HPCH105 = 'AREA'      ! Assign the name AREA to the data from channel 105
HPD_RFM=1             ! Set Tascom to read HPD when HPD has data ready.
 .
 .
?AREA                 ! When HPD has been read, print data from channel 105
```

21.File data input.

Data lines in a file may be read to a Tascom array provided that the lines contains only numbers. The file must be located in the user's command directory and is opened with

FIL_OPEN 'name-of-file'.

When the file is open, the command **FIL_READ** reads one line from the file so that the first FIL_READ command reads the first data line from the file, the next FIL_READ command reads the next data line, etc. until end of file. Only data lines are read; empty lines or lines not starting with a number are skipped. The data lines must contain only data tokens (numbers) separated by separation characters. The separation character is specified in the string variable **FIL_SEP**. After each FIL_READ command, the data tokens are converted to real values and put in array **FIL_PAR[]**. The variable FIL_LINE is set to the number of lines read or to -1 if no file is open. The number of data tokens to be read from each line is specified in the variable **FIL_NDAT**. The file is closed by ctrl(c), by an attempt to read past end of file or when a new FIL_OPEN command is executed.

Example:

A data file by the name of euler.inp may look like:

```
title: rt test
file name: euler.inp
   0.200   1e+1   0.200    10.649    -5.325
   0.200   1e+1   0.200    13.052    -6.526
   0.000   1e+1   0.400    15.082    -7.541
   0.200   1e+1   0.400    16.875    -8.437
   0.200   1e+1   0.400    18.499    -9.249
```

Each data line has five data tokens separated by spaces. The following Tascom commands may be used:

```
FIL_SEP =' '
FIL_NDAT=5
FIL_OPEN 'euler.inp'
FIL_READ
?FIL_LINE
```

The FIL_READ command will skip the two first lines that are not data lines and read the first data line. The array FIL_PAR[] will contain

```
FIL_PAR[0] : 0.2
FIL_PAR[1] : 10
FIL_PAR[2] : 0.2
FIL_PAR[3] : 10.649
FIL_PAR[4] : -5.325
```

and FIL_LINE is set to 1. The next FIL_READ command will read the next data line from the file and so on until all lines are read. In case of error, the value the array element is set to the value 2e38.

Appendix A

```
!           Gearing file:   rtg1_gea.tas
!          ------------------------------
! This file is used to initialise the motor parameters. The data
! in the file is read by TASCOM with the command RTG1_GEA. The file
! require the presence of the command files _mpar1.tas and _mpar2.tas.
!
! Explanation of motor parameters:
! CONT   : Controls specified by the bits in the CONT
!          bit1 set: P2048 control signal used, e.g. for aircushions.
!          bit2 set: Analyzer shielding up/down with this motor.
!          bit4 set: Define floor motor on BW2
!          bit5 set: Manual control disabled
! ACC    : Acceleration/deceleration time.
!           P1648: 2, 1, or .5 sec., P2048: 5, 2, 1, or .5 sec.
! START  : Speed at start of acceleration.
!           P1648: 100-500 steps/sec, P2048: 10-4000 steps/sec.
! MAX    : Max speed in fast speed range.
!           P1648: 100-3000 steps/sec, P2048: 100-20000 steps/sec.
! AUTO   : Speed in slow speed range at auto. 1-500 steps/sec.
! MAN    : Max speed in slow speed range at manaual. 1-500 steps/sec.
! DELAY  : Delay before and after a motor setting. 0-2500 millisec.
!           For motor drive P1604 minimum 50 for the enable signal.
! ROT    : P2048 only. If= 1: normal (positive) direction of rotation.
!                      If=-1: negative direction of rotation; only if
!                      the DIR switch on the module is in neutral.
!
! ENC    : If=0: no encoder for this motor,  if>1: encoder number.
! RACK   : Power unit number 1 2 3 ...etc. Motors sharing power unit
!           can not run simultaneously.
! RANGE  : =0: slow auto speed range, =1: fast auto speed range
! TOL    : Tolerans. If calculated number of steps is less than
!            tolerance, no motor setting.
! DEGREE : If no encoder: number of motor steps per degree.
!          If encoder: number of encoder digits per degree.
! DIG    : If no encoder: =0
!          If encoder: Motor steps per encoder digits.
! BACK   : Backlash. A number of degree that the motor should
!           overshoot when moving into position from negative
!            (back>0) or positive (back<0) direction.
!
! LLIM   : Lower software limit.
! ULIM   : Upper software limit.
! OFFSET : For axis with encoders: Offset from encoder output; changed
!           with REMO command; used when reading the motor position.
!          For axis without encoders: Differences after REMO commands;
!           for information only.
! FIX    : If=1, motor is fixed at the present position.
!
!
```

```
! LOAD FIRST GROUP OF MOTOR PARAMETERS BY CALLING  _mpar1.tas:
!
? 'LOADING PRIMARY MOTOR PARAMETERS...';
!
DMAN = 0                ! DISABLE MANUAL CONTROL WHILE LOADING
!
!    MOTOR ,START,   MAX, AUTO, MAN, DELAY, ACC, RANGE, ENC, CONT, ROT
!-------------------------------------------------------------------------
_MPAR1   1 ,  10 ,   200,   30,  50, 600  ,  5 ,   1  ,  1 ,   1 ,  1
_MPAR1   2 , 500 ,  2200,  500,  50,  50  ,  1 ,   1  ,  0 ,   0 ,  1
_MPAR1   3 ,  10 ,  1000,  500,  50,  50  ,  1 ,   1  ,  0 ,   0 ,  1
_MPAR1   4 , 500 ,  2000,  500,  50,  50  ,  5 ,   1  ,  0 ,   0 ,  1
_MPAR1   5 , 500 ,  2000,  500,  50,  50  ,  2 ,   1  ,  0 ,   0 , -1
_MPAR1   6 , 100 ,   600,   50,  40, 600  , .5 ,   1  ,  2 ,   0 ,  1
_MPAR1   7 , 100 , 20000,  300,  40,  50  ,  1 ,   1  ,  0 ,   0 , -1
_MPAR1   8 , 330 ,  1000,  330,  40,  50  ,  2 ,   1  ,  0 ,   0 ,  1

;

! LOAD SECOND GROUP OF MOTOR PARAMETERS BY CALLING _mpar2.tas:
!
? 'LOADING SECONDARY MOTOR PARAMETERS...';
!
!      MOTOR , RACK, DELAY,   TOL,  DEGREE,     DIG,   BACK
!-------------------------------------------------------------
_MPAR2     1 ,   1 ,  600 ,     1,    0.02,     100,   2.66
_MPAR2     2 ,   1 ,   50 , 0.001,    1000,       0,    .15
_MPAR2     3 ,   2 ,   50 ,  .045,     100,       0,    .52
_MPAR2     4 ,   2 ,   50 ,  .001,    1000,       0,    .15
_MPAR2     5 ,   3 ,   50 ,  .001,    1000,       0,    .15
_MPAR2     6 ,   4 , 1000 ,   .01,     100,    2.88,   4.0
_MPAR2     7 ,   4 ,   50 ,   .01,    1000,       0,   0.05
_MPAR2     8 ,   4 ,   50 ,   .01,     100,       0,   1.5
!
DMAN = 1                ! ENABLE MANUAL CONTROL AGAIN
LAST_GEA = 'rtg1_gea'    ! NAME OF THIS FILE
```

Appendix B

```
!           Motor parameter files:   _mpar1.tas and _mpar2.tas
!          ---------------------------------------------------
! _mpar1.tas:
!***********
! This file loads the first group of 10 motor parameters into the
! motor control block and into the motor.
NUMBER=0
VAR: NUMBER,START,MAX,AUTO,MAN,ACC,RANGE,ENC,CONT,ROT
!
_SMPNU NUMBER,'START',START     ! START SPEED
_SMPNU NUMBER,'MAX',MAX         ! MAXIMUM SPEED
_SMPNU NUMBER,'AUTO',AUTO       ! SLOW SPEED IN AUTO MODE
_SMPNU NUMBER,'MAN',MAN         ! SLOW SPEED IN MANUAL MODE
_SMPNU NUMBER,'ACC',ACC         ! ACCELLERATION TIME
_SMPNU NUMBER,'DELAY',DELAY     ! START DELAY TIME
_SMPNU NUMBER,'RANGE',RANGE     ! SPEED RANGE.
_SMPNU NUMBER,'ENC',ENC         ! ENCODER NUMBER. 0 MEANS NO ENCODER
_SETPAR MOTOR,'CONT',CONT       ! MOTOR HAS A CONTROL SIGNAL. MUST
                                ! BE SET BEFORE DELAY BECAUSE DELAY FOR
                                ! CONTROL SIGNALS IS TASCOM INTERNAL.
_SMPNU NUMBER,'ROT',ROT         ! DIRECTION OF ROTATION


! _mpar2b.tas:
!************
! This file loads the second group of 6 motor parameters into the motor
! control block and into the motor.
!
VAR: NUMBER,RACK,DELAY,TOL,DEGREE,DIG,BACK
!
_SMPNU NUMBER,'RACK',RACK       ! MOTOR DRIVER POWER RACK NUMBER
_SMPNU NUMBER,'DELAY',DELAY     ! START DELAY TIME
_SMPNU NUMBER,'TOL',TOL         ! POSTION TOLERANCE
_SMPNU NUMBER,'DEGREE',DEGREE   ! STEPS PER DEGREE
_SMPNU NUMBER,'DIG',DIG         ! DEGREES PER ENCODER DIGIT
_SMPNU NUMBER,'BACK',BACK       ! BACKLASH NUMBER OF DEGREE
```

Appendix C

```
!      Display parameter specification file:  rtg1_dis.tas
!     ---------------------------------------------------
! The file specifies for the ECB-system display which units should be
!  displayed, the order in which they are displayed, and for each unit
!  a name that is displayed. Also the total number of units to be
!  displayed is specified. The file is read and the parameters send to
!  the ECB-system with the TASCOM command RTG1_DIS.
!
! The arguments to the _SETDIS call are:
!  First arg:   Display unit. Number in the range 1-48.
!  Second arg:  Name of display unit. A 3 character string in ' '.
!               If the name for a motor or encoder is
!               given as ***, the name is taken from the symbol table
!  Third arg:   The type. TIMER, SCALER, RATEMETER, MOTOR or ENCODER
!  Fourth arg:  Device number in the following range:
!                 Scaler/timer: timer= 0, scalers/ratemeters= 1-3
!                 Motors= 01-total number of motors in the experiment
!                 Encoders= 01-03, without offset= 41-43
!
? 'LOADING DISPLAY PARAMETERS...'
!
_SETDIS 01  , 'RmS' ,   'RATEMETER',    01
_SETDIS 02  , 'RmM' ,   'RATEMETER',    01
_SETDIS 03  , 'RmF' ,   'RATEMETER',    01
_SETDIS 04  , 'MON' ,   'SCALER',       01
_SETDIS 05  , 'TIM' ,   'TIMER',        00
_SETDIS 06  , 'I  ' ,   'SCALER',       02
_SETDIS 07  , '***' ,   'ENCODER',      01
_SETDIS 08  , '***' ,   'MOTOR',        02
_SETDIS 09  , '***' ,   'ENCODER',      02
_SETDIS 10  , '***' ,   'MOTOR',        04
_SETDIS 11  , '***' ,   'MOTOR',        05
_SETDIS 12  , '***' ,   'MOTOR',        06
_SETDIS 13  , 'M1 ' ,   'MOTOR',        07
_SETDIS 14  , 'M2 ' ,   'MOTOR',        08
_SETDIS 15  , 'AUX' ,   'SCALER',       03
_SETDIS 16  , 'RdS' ,   'RATEMETER',    02
_SETDIS 17  , 'RdM' ,   'RATEMETER',    02
_SETDIS 18  , 'RdF' ,   'RATEMETER',    02
!
_DISUN 18               ! No. of units to display
LAST_DIS = 'rtg1_dis'    ! NAME OF THIS FILE
```

Appendix D

The TASCOM symbol table

The symbol table has a binary and an ASCII version. TASCOM creates the binary version from the ASCII version when the ASCII version is loaded with the command _LO_S. Each symbol in the symbol table is defined by its name and five numbers and a string:

SYMBOL-NAME = SCOPE TYPE NUM1 NUM2 VALUE FORMAT

SYMBOL-NAME:
Up to 8 letters, digits, or underscore starting with a letter. For motors max 3 letters or digits starting with a letter or 2T. System symbols may start with _(underscore).
SCOPE:
An integer specifying the scope of the symbol. Scopes are defined internally in TASCOM. Some of the many defined scopes are:

1	Simple system symbol
2	User symbol
3	Read-only symbol
200	Procedure call
201	Procedure call with a string as argument

TYPE:
An integer specifying the type of the symbol. Defined types are:

1	Real
2 or 3	String
4	Variable pointer (indirect symbol)
5	Array

NUM1:
An integer. For arrays the first index. For ECB motors =0, for VME motors =1, for Trinamic motors = 2.
NUM2:
An integer specifying a number, e. g. motor number for motor symbols, scaler number for scaler symbols, procedure number for procedure calls. For two dimensional arrays the second index, for one dimensional arrays =0.
VALUE:
If TYPE=1, a real specifying the value of the symbol. If TYPE=2, a string of max 16 characters. For motor symbols (M1, LM1, etc.) the value used by TASCOM is held in a special motor control block for each motor and the value in the symbol table is not used. For procedure calls the value is the number of arguments to the call, or 0, if number-of-arguments checking is not wanted.
FORMAT:
A string specifying the format used at output of the symbol

Anything on a line following ! is a comment.

The ASCII version of the symbol table file that has extension .sor, is found under the /usr/users/tasfiles/symfil directory in the system managers log-in account and can be loaded from within TASCOM. A new ASCII symbol table file must be loaded when new system symbols are wanted or when symbol names are changed, e.g. changes of motor names. This should only be done by the system manager and only after the symbol values are saved, see below.

A new symbol table file is loaded with one of the TASCOM commands **_LO_S** or **_LO_SOU** that loads the source ASCII version of the symbol table and creates a new binary version, e.g., the command

_LO_S 'bw1_sym'

loads the ASCII symbol table file bw1_sym.sor. When the new symbol table file is successfully loaded, it is recommended to save the binary version of the symbol table with the command _SAVE_S or by EXIT. It is the binary version that is saved when TASCOM is exited and reloaded when TASCOM is started. If for some unexpected reason TASCOM is stopped before the new binary symbol table is saved, e.g. by a power failure, then when TASCOM is restarted the old binary symbol table will be loaded, probably causing some confusion.

The name of the ASCII symbol table file is stored in the variable SYM_NAME when the ASCII symbol table file is loaded with _LO_S. Since SYM_NAME is a read-only variable, it is always possible with the command ?SYM_NAME to see what symbol table is in use.

The actions performed by the two versions of the load commands for the ASCII symbol table file are the following:

_LO_S'name': Loads the ASCII symbol table file and creates a new binary symbol table. Assigns the values from the ASCII symbol table file to the symbols and resets motor parameters to TASCOM default values (not the same as in, e.g. bw1_gea.tas). Following a _LO_S command, motor parameters and display set-up must be reloaded with the appropriate commands, e.g. for the BW1 spectrometer BW1_GEA, and BW1_DIS, respectively. Symbols like SEGN, FINA, and other must be redefined, and the position of all motors with encoders must be checked and possibly changed with the REMO command. The _LO_S command is used when major changes are made or, e.g. if the value of a read-only symbol is changed.

_LO_SOU'name': Appends new symbols from the ASCII symbol table file to the existing binary symbol table. The values and symbols in the existing binary symbol table including user defined symbols are not changed. _LO_SOU is used when the only change to a symbol table is the addition of new symbols.

If an attempt is made to load an incorrect ASCII symbol table file, an error message is shown on the screen and the old binary symbol table is reloaded so that TASCOM is restored to the state before the load command. Then the user may correct the error and try a new load command.

A new user symbol is created by assigning a value or a string to a name that is not already in the symbol table. This may eventually fill-up all empty slots in the symbol table so no

new user symbol can be created. The command FICA lists the number of symbols in the symbol table and the number of empty slots. The command **SHOW_US** lists all user symbols, the command **SHOW_SS** lists all system symbols, and the command **SHOW_A** lists all arrays. The user is notified each time a new user symbol is created if the variable **REPO_US** is REPO_US=1.

User symbols and user arrays may be deleted. The command **DELS_ALL** deletes all user symbols in the binary symbol table while the command **DELA_ALL** deletes all user arrays. The command **DELS** deletes a user symbol, e.g. DELS MINE deletes the user symbol MINE from the symbol table. The command **DELA** deletes a user array, e.g. DELA A[] deletes user array A[] from the symbol table.

The command **_SYMBOL 'file-name'** writes all symbol values and formats to the file file-name.tas in the user's command directory. The file is an ASCII file that may be edited by the user. The file may later be read by TASCOM and all symbols thereby set to the values and formats saved in the file. If user symbols in the file does not exist in TASCOM, they are created with the value and format from the file. The _SYMBOL command may be used to save symbol values and formats before a new ASCII symbol table file is loaded with the _LO_S command which sets all system symbols to the values in the ASCII symbol table file and deletes all user symbols.

As an example using the BW2 set-up files, the following sequence of commands save all symbols (including motor positions) in a file in the user command directory, load a new symbol table file, restore motor set-up, restore the display set-up, and restore all symbols:

>_SYMBOL'my_symb'	Save symbol values in my_symb.tas
>_LO_S 'bw2_sym'	Load new symbol table file
>BW2_MUX	Load motor multiplexing
>BW2_GEA	Load motor parameters
>BW2_DIS	Load display set-op
>TCOM_DIR='name_of_user_directory'	Set user directory
>MY_SYMB	Load symbol values from my_symb.tas
>_SAVE_S	Save new binary version of symbol table

Note that the ASCII symbol table file name is bw2_sym.sor is in lowercase letters. Therefore, the command to load the ASCII symbol table file is either_LO_S 'bw2_sym' or _lo_s 'bw2_sym'. Tascom is case insensitive but a Unix/Linux file name is not.

Accessing symbol parameters.

As mentioned above each symbol in the symbol table is defined by its name and five numbers and a string:

SYMBOL-NAME = SCOPE TYPE NUM1 NUM2 VALUE FORMAT

The parameters scope, type, num1, and num2 as well as the index to the symbol (see below) may be read by TASCOM functions.

Index. Internally in TASCOM each symbol is assigned an index, which is a value between 1 and the maximum number of symbols that the symbol table can have. It is possible to get the index to a symbol with the command **G_INDEX(symbol)**, eg ?G_INDEX(FINA) returns the index to symbol FINA. If a symbol is undefined, 0 is returned. Thus the command may be used to check whether a certain symbol is defined or not. If the symbol is an indirect symbol, the index of the symbol to which the indirect symbol points is returned, for example, if for the indirect symbol MOTOR

MOTOR = @OMM

so that MOTOR points to OMM, the commands

?G_INDEX(MOTOR)
?G_INDEX(OMM)

both return the index to symbol OMM.

For an indirect symbol the command **S_INDEX(i-symbol)**, where i-symbol is an indirect symbol, can be used to point to a certain symbol. For example

S_INDEX(MOTOR) = G_INDEX(OMM)

is identical to

MOTOR = @OMM

A possible use of the G_INDEX and S_INDEX commands is is to temporarely save a pointer used by an indirect symbol, assign a new pointer, and later re-assign the saved pointer. For example the program sekvens

```
XX = G_INDEX(MOTOR)
S_INDEX(MOTOR) = G_INDEX(OMM)
…
?MOTOR
…
S_INDEX(MOTOR) = XX
```

first saves the pointer of the indirect symbol MOTOR in the variable XX, then assigns a new pointer, OMM, to MOTOR, then executes one or more commands with the new pointer, and finally re-assigns the original pointer to MOTOR.

Scope. The scope for a symbol is a value greater than zero. It is possible to get the scope for a symbol with the command **G_SCOPE(symbol)**, eg ?G_SCOPE(FINA) returns the scope for symbol FINA. If a symbol is undefined, 0 is returned. If the symbol is an indirect symbol, the scope of the symbol to which the indirect symbol points is returned.

The G_SCOPE command may be used to check the scope of indirect symbols. For example, the scope of a motor symbol is 10, so if the indirect symbol MOTOR is pointing to a motor symbol, the command G_SCOPE(MOTOR) should return 10. If anything else is returned, the symbol pointed to by MOTOR is not a motor symbol.

Type. The type for a symbol is a value greater than zero. It is possible to get the type for a symbol with the command **G_TYPE(symbol)**, eg ?G_TYPE(FINA) returns the type for symbol FINA. If a symbol is undefined, 0 is returned. If the symbol is an indirect symbol, the type of the symbol to which the indirect symbol points is returned. The G_TYPE command may, for example, be used to check if a symbol is an array symbol; if so, the G_TYPE command should return 5, which is the type for array symbols.

Num1 and Num2. Num1 and num2 for a symbol are values greater than or equal to zero. The values may be returned with the commands **G_NUM1(symbol)** and **G_NUM2(symbol)**, eg ?G_NUM1(FINA) returns the value of num1 for the symbol FINA. If a symbol is undefined, -1 is returned. If the symbol is an indirect symbol, num1 or num2 for the symbol to which the indirect symbol points is returned. The G_NUM1 and G_NUM2 commands may, for example, be used to check the size of an array symbol. For a one dimensional array, num2 is 0 and num1 the number of array elements. For a two dimensional array, the size is num1*num2.

The following is an examlpe on the use of the symbol parameter functions. It checks whether symbol DD is defined as an array symbol and if so, returnes the size of the array.

```
IF G_INDEX(DD) == 0
 ?'undefined symbol DD'
 PROMPT
END
IF G_TYPE(DD) |= 5
 ?'DD is not an array symbol'
 PROMPT
END
IF (G_NUM2(DD) == 0) .AND. (G_NUM1(DD) |= 0)
 ?'DD is a 1 dim array. Number of elements:',G_NUM1(DD)
END
IF (G_NUM2(DD) |= 0) .AND. (G_NUM1(DD) |= 0)
 ?'DD is a 2 dim array. Number of elements:',G_NUM1(DD),',',G_NUM2(DD)
END
```

Symbol table example.

The following is an example on a somewhat reduced ASCII symbol table for an instrumentation with four motors.

```
! SYMBOL TABLE
!
! SYM_NAME is set to the name of the file when it is loaded
SYM_NAME=    3 2 0 0       ' '   '%s'
!
! MOTOR CONTROL
M1      =   10 1 0 1        0.  '%12g'  ! MOTORS HAVE SCOPE 10
M2      =   10 1 0 2        0.  '%12g'
M3      =   10 1 0 3        0.  '%12g'
M4      =   10 1 0 4        0.  '%12g'

! LOWER SOFTWARE LIMITS                ! LOWER LIMITS HAVE SCOPE 11
! UPPER SOFTWARE LIMITS                ! UPPER LIMITS HAVE SCOPE 12
! The names for software limits and the symbol _motor-name are
! generated automatically by TASCOM when the symbol table is loaded
! with the _LO_S command.
! For a motor with the name M1 the symbols M1, _M1, LM1 and UM1 exist.

CALC    =   14 1 0 0        0.  '%d'    ! = 1:CALCULATION MODE ON
POTE    =    1 1 0 0        1.  '%d'    ! = 1: SOFTWARE LIMITS TESTED
OUPO    =    1 1 0 0        1.  '%d'    ! OUPO = 1 OUTPUT LIMIT

! COUNTING: SCALERS, TIMER AND RATEMETERS
TIM     =   33 1 0 0        0.  '%10.3f'! TIMER
MON     =   30 1 0 1        0.  '%9d'   ! MONITOR
I       =   30 1 0 2        0.  '%9d'   ! DETECTOR
AUX     =   30 1 0 3        0.  '%9d'   ! AUXILIARY
PRSC    =    1 1 0 0        0.  '%d'    ! SELECT PRESET UNIT
PRES    =    1 1 0 0        1.  '%12g'  ! PRESET IN SECONDS OR COUNTS
RATE    =   36 1 0 0        1.  '%d'    ! RATE = 1: RATEMETERS ENABLED
COFL    =   32 1 0 0        0.  '%d'    ! COFL = 1: SCALERS COUNTING
TIME    =   35 1 0 0        0.  '%12g'  ! TIME IN SECONDS
TFRQ    =   37 1 0 0       10.  '%12g'  ! TIMER FREQUENCY (1000 or 10)

! DATA STORAGE
FINA    =   20 2 0 0    'mine'  '%12s'  ! FILE NAME OF .DAT FILE
SEGN    =   21 1 0 0        0.  '%12g'  ! DATA FILE NUMBER (OPEN FILE)
DFIT    =    1 3 0 0        0.  '%12s'  ! FILE TEXT (DFIT = 'Scan')
FIOU    =    1 1 0 0        1.  '%d'    ! = 0: NO DATA TO FILE
PROU    =    1 1 0 0        1.  '%d'    ! = 0: NO DATA TO SCREEN
DDEV    =    1 2 0 0       ' '  '%s'  ! DEVICE (EG DDEV = 'B:')
DATA_DIR=    1 2 0 0       ' '  '%s'  ! DATA SUBDIR.

! COMMAND FILES
TCOM_DIR=    1 2 0 0       ' '  '%s'  ! .TAS FILE SUBDIRECTORY
SFIL    =    1 2 0 0       ' '  '%s'  ! STARTUP FILE NAME
```

```
! TASCOM procedure calls.
! Procedure calls have scope 200 or 201. The procedure number is the
! fourth argument after the = sign. The fifth argument, the 'value',
! specifies the number of arguments to the procedure call.  If the
! value is 0, the number of arguments is not checked. A value of 10
! specifies one of the output parameter buffers, e.g. DPRP, APRP.

PRINT   =   200 0 0 10      0.  '%12g'  ! Executes the '?' commands
EXIT    =   200 0 0 100     0.  '%12g'  ! Exit from tascom
_LO_S   =   200 0 0 102     0.  '%12g'  ! Load ascii symbol table
_LO_SOU =   200 0 0 118     0.  '%12g'  ! Load ascii symbol table
_SAVE_S =   200 0 0 115     0.  '%12g'  ! Save binary symbol table
_SMPNA  =   200 0 0 106     0.  '%12g'  ! Set motor params by name
_SMPNU  =   200 0 0 119     0.  '%12g'  ! Set motor params by number
_SETDIS =   200 0 0 107     0.  '%12g'  ! Set display parameters
_DISUN  =   200 0 0 108     0.  '%12g'  ! Set No. of display params
HELP    =   200 0 0 109     0.  '%12g'  ! Type a .HLP file

! Motor control
REMO    =   200 0 0 20      2.  '%12g'  ! Redefine motor position
MOPO    =   200 0 0 21      0.  '%12g'  ! List of motor positions
NEWP    =   200 0 0 22      2.  '%12g'  ! Define new motor positions
MOVE    =   200 0 0 23      0.  '%12g'  ! Move to position defined by
MSTA    =   200 0 0 24      2.  '%12g'  ! Starts motor running
MSTO    =   200 0 0 25      1.  '%12g'  ! Stops motor started by MSTA
LIPO    =   200 0 0 27      0.  '%12g'  ! Output a list of soft limits
LMPAR1  =   200 0 0 29      0.  '%12g'  ! List primary motor params
LMPAR2  =   200 0 0 30      0.  '%12g'  ! List secondary motor params

! Counting scalers and ratemeters
COUN    =   200 0 0 50      0.  '%12g'  ! Count for preset
COST    =   200 0 0 52      0.  '%12g'  ! Start count, return to >
COFL    =   200 0 0 53      0.  '%12g'  ! Test scalers. =1 if counting

! Data storage
TYPE    =   201 0 0 62      0.  '%12g'  ! Type .tas file
OUT     =   200 0 0 63      0.  '%12g'  ! Writes PRP/FIP variables
LPRP    =   200 0 0 64      0.  '%12g'  ! List PRint Params
LFIH    =   200 0 0 65      0.  '%12g'  ! List FIle Header
LFIP    =   200 0 0 66      0.  '%12g'  ! List FIle Params
DPRP    =   200 0 0 67     10.  '%12g'  ! Define PRint Params
DFIH    =   200 0 0 68     10.  '%12g'  ! Define FIle Header
DFIP    =   200 0 0 69     10.  '%12g'  ! Define FIle Params
APRP    =   200 0 0 70     10.  '%12g'  ! Append PRint Params
AFIH    =   200 0 0 71     10.  '%12g'  ! Append FIle Header
AFIP    =   200 0 0 72     10.  '%12g'  ! Append FIle Params
REAP    =   200 0 0 76      0.  '%12g'  ! Remove all appended params
FICA    =   200 0 0 78      0.  '%12g'  ! Output file characteristics

! Wait
WAIT    =   200 0 0 190     1.  '%12g'  ! Wait n seconds

! END OF SYMBOL TABLE
```

Appendix E

Programming the switches on the manual box.

The functions of the switches on the manual box are programmable by the TASCOM variables FKEY*i* where $1 \leq i \leq 7$. By default each of the variables calls a command file cfkey*i*.tas in the system command file directory /usr/users/tasfiles/command/sys/.

The connection between the switches, FKEY*i*, and cfkey*i* is listed below. Refer to the sketch of the manual box in the chapter describing the box.

FKEY*i*	cfkey*i*	Switch on box	
FKEY1	cfkey1	Pos1,2 peak	P-mode, first time the switch is activated
FKEY2	cfkey2	Pos1,2 peak	P-mode, second time the switch is activated
FKEY3	cfkey3	Pos1,2	P-mode, first time the switch is activated
FKEY4	cfkey4	Pos1,2	P-mode, second time the switch is activated
FKEY5	cfkey5	upper	L-mode
FKEY6	cfkey6	lower	L-mode
FKEY7	cfkey7	both	P-mode, both switches activated simultaneously

The command files cfkey*i*.tas that are located in the system command file directory /usr/users/tasfiles/command/sys/ should be changed by the system manager only. If the user wants to change the function of one of the switches, he should copy the command file for the switch to the user command directory and make the changes there. When TASCOM looks for command files, it looks first in the user command directories defined by TCOM_DIR. If the file is not found there, TASCOM looks in the system command directory. Thus if files exist with the same name in several directories, the file found first will be used by TASCOM.

Another way of changing the function of a switch is to make an assignment to FKEY*i*. For example, FKEY1='COUN ?I' will start a count for the given preset followed by a print of the scaler each time the key is pressed. Such an assignment, however, is lost when TASCOM is exit since starting TASCOM sets FKEY*i* = 'cfkey*i*'.

To further ilustrate the function of the manual box operations, follows listings of some of the cfkey*i* the command files.

```
! cfkey3.tas: COMMAND FILE FOR FUNCTION KEY 3
GMOT MOTOR
PPO1=MOTOR
_GMPNA MOTOR,'NUMBER'
PPO1_NUM=VDIG
?'MOTOR PPO1:  ',MOTOR

! cfkey4.tas: COMMAND FILE FOR FUNCTION KEY 4
GMOT MOTOR
PPO2=MOTOR
_GMPNA MOTOR,'NUMBER'
PPO2_NUM=VDIG
?'MOTOR PPO2:  ',MOTOR

! cfkey7.tas: COMMAND FILE FOR FUNCTION KEY 7
OLDPRES=PRES
OLDPRSC=PRSC
PRSC=0
PRES=2
DMAN=0
IF (PPO1_NUM == PPO2_NUM)
  SMA MOTOR,PPO1,PPO2,7,PRES
ELSE
  ?'PPO1 and PPO2 refer to different motors'
END
PRES=OLDPRES
PRSC=OLDPRSC
DMAN=1
```

In cfkey3.tas the command **GMOT** sets variable MOTOR to point to the motor selected on the manual box. The function is equvalent to doing an indirect assignment like MOTOR=@<name of manually selected motor >. The position of the motor is saved in variable PPO1, the motor number in PPO1_NUM, and the position printed on the screen. cfkey4.tas is similar.

In cfkey7.tas the preset values are temporarily saved and the manual motor box is disabled to allow the motor to be run from the computer. Then if the positions in PPO1 and PPO2 refer to the same motor, a scan is done from PPO1 to PPO2. Finally the manual box is enabled again and the old preset restored.

```
! cfkey5: COMMAND FILE FOR FUNCTION KEY 5
GMOT MOTOR
_SMPNA MOTOR,'ULIM',MOTOR
?'New upper limit for',MOTOR

! cfkey6: COMMAND FILE FOR FUNCTION KEY 6
GMOT MOTOR
_SMPNA MOTOR,'LLIM',MOTOR
?'New lower limit for',MOTOR
```

In cfkey5 and cfkey6 the command GMOT sets variable MOTOR to point to the motor selected on the manual box. The Set Motor Parameter call sets the limit to the present motor position.

Appendix F

How to change motor names.

The names of motors are defined in the ASCII symbol table. Motor names may have a maximum of *three* characters (letters or numbers). Change of motor names should be done by the system manager only.

Consider changing a motor name from M1 to CHI: The source ASCII version of the symbol table found under the /usr/users/tasfiles/symfil directory is edited to replace M1 with CHI. The new symbol table is loaded with the TASCOM command _LO_S which loads the source ASCII symbol table file and creates a new binary version. If e.g. the name of the ASCII symbol table file is bw1_sym.sor, the load command is _LO_S 'bw1_sym'. When the new symbol table is succesfully loaded, save the binary version of the symbol table with the command _SAVE_S. When TASCOM is started, the binary version is loaded and when TASCOM exits, the binary version is saved.

When _LO_S is used to load the symbol table file, all parameters are set to default values. Therefore, following a _LO_S command, the motor and display set-up files must normally be reloaded. The motor multiplexing set-up file, eg bw2_mot.tas, must be loaded as the first one if multiplexing is used. Then the motor parameter file, e.g. bw2_gea.tas. Finally the display set-up file, e.g. bw2_dis.tas. Also some parameters, e.g. the user directories, file names, SEGN, and others might have to be set.

The motor names might also have to be changed in the display set-up file found in directory /usr/users/tasfiles/command/sys. This determines what name appears on the four-line video display. An example of a display set-up file is given in appendix C. If the motor names in the display set-up file are defined as ***, see appendix C, the motors will be given the same names as defined in the symbol table and it is not necessary to edit the file after a motor name change. However, the new names must be defined on the display by loading the display set-up file.

A sequence of commands that loads a new symbol table and restore the motor parameters and the display might look as follows (for BW2):

>_SYMBOL'my_symb'	Save all symbol values in my_symb.tas
>_LO_S 'bw2_sym'	Load new symbol table file
>BW2_MUX	Load motor multiplexing
>BW2_GEA	Load motor parameters
>BW2_DIS	Load display set-op
>TCOM_DIR='name_of_user_directory'	Set user directory
>MY_SYMB	Load symbol values from my_symb.tas
>_SAVE_S	Save new binary version of symbol table

Appendix G

How to reload the ECB-C program from the PC.

In case of error in the ECB system, e.g. the ECB display is locked or not reacting on the scrollbox, and if it does not help to restart the ECB system as described in 'Restart of TASCOM after errors', the ECB program might have to be reloaded from the PC.

Proceed as follows:
Restart the ECB system by power-off - power-on. Thereafter, for the next 10 seconds the display shows:

'STARTING ECB'.

During this period press the UP or DOWN button at the lower right corner on the manual motor box. When the 10 seconds have elapsed, the display changes and shows:

'ECB PROGRAM NOT STARTED, LOAD NEW ECB PROGRAM' or
'ECB PROGRAM NOT VALID, LOAD NEW ECB PROGRAM'

Load the ECB program from the workstation by giving the command LOAD_ECB.

When this program has been succesfully run and the ECB program thereby reloaded, the ECB display should change to show either a blank screen or some ealier display set-up.

Start TASCOM with the Linux command tascom.

Restore the motor parameters and the display parameters by loading the gearing file, e.g. bw1_gea.tas, and the display set-up file, e.g. bw1_dis.tas.

Check the motor positions and restore them if necessary. Exit and reenter TASCOM in order to save the present state of TASCOM.

Appendix H

Set-up of motor multiplexing with P2324

The P2324 Octal motor drive can multiplex eight motors from one ECB motor control port. Since the P2048 ECB motor control module has four control ports, one P2048 may be connected to four P2324s and thereby control up to 32 motors. There are some restrictions when using P2324, in particular that the eight motors connected to one P2324 cannot run at the same time and this must be set up in the motor gearing file, e.g. rtg1_gea.tas for the X-Ray spectrometer and described in appendix A. Each time one of the multiplexed motors is going to run, the motor parameters for the motor that was last running are saved in the ECB memory while the parameters for the motor that is going to run are loaded from the ECB memory to P2048. Thus TASCOM must know the multiplexing scheme in detail.

When multiplexing is used, the parameter **MULT_MOT** must be set to MULT_MOT=1. If multiplexing is not used, MULT_MOT=0. After a new value has been assigned to MULT_MOT, TASCOM must be exited and restarted before the new value is valid. A print of which motors are multiplexed may be obtained with the list port command **LPORT**.

The motor multiplexing is set up with one of the set port commands **_SPORTNA** or **_SPORTNU**. The syntax is:

_SPORTNA Motor_name, Port_number, Line_number
_SPORTNU Motor_number, Port_number, Line_number

Port_number is the number on the output port on the ECB motor control module P2048. If the ECB system has six P2048 modules with four ports each, $1 \leq$ Port_number ≤ 24.

Line_number is either 0 if the motor is not multiplexed, or the number on the output line on the Octal motor drive if the motor is multiplexed. Thus $0 \leq$ Line_number ≤ 8.

The _SPORTNU and _SPORRTNA commands must be executed with the motor numbers in consecutive order starting with motor 1. It is therefore most practical to use a .tas command file for the multiplexing set-up. An example is given below:

```
! SET UP OF THE MULTIPLEXER CONFIGURATION. FILE NAME: rtg1_mux.tas
!-----------------------------------------------------------------
!
!
? 'LOADING MOTOR PORT CONFIGURATION...'
;
!
!MOTOR NUMBER , PORT , LINE
!
_SPORTNU   1  ,  1   ,  0
_SPORTNU   2  ,  2   ,  0
_SPORTNU   3  ,  3   ,  0
_SPORTNU   4  ,  4   ,  0
_SPORTNU   5  ,  5   ,  0
_SPORTNU   6  ,  6   ,  0
_SPORTNU   7  ,  7   ,  1
_SPORTNU   8  ,  7   ,  2
_SPORTNU   9  ,  7   ,  3
_SPORTNU  10  ,  7   ,  4
_SPORTNU  11  ,  7   ,  5
_SPORTNU  12  ,  8   ,  1
_SPORTNU  13  ,  8   ,  2
_SPORTNU  14  ,  8   ,  3
_SPORTNU  15  ,  8   ,  4

LAST_MUX = 'rtg1_mux'    ! NAME OF THIS FILE
```

This example is for an instrumentation with two P2048 ECB motor modules, i.e. eight ports, six P1604 Motor drives, and two P2324 Octal motor drives. The commands are given with the motor numbers in consecutive order from motor 1 to 15. The non-multiplexed motors must come first, motors 1 to 6. Then five motors controlled by port seven and multiplexed through the first five drive output lines on the first P2324. Finally four motors controlled by port eight and multiplexed through the first four drive output lines on the second P2324.

Appendix J

General GPIB device command.

The general GPIB device command GDEV may be used to send commands to devices on the GPIB. The command may be used to communicate with devices that are not otherwise supported by TASCOM. It will work with most devices but not all.

The device is specified by the GPIB address in TASCOM symbol _GADDR. The terminator character in the message to the device is set in symbol _GWTERM and the terminator character in the return message from the device in symbol _GRTERM. If _GRTERM = 0, no return message is expected. The return message is stored in array GD[]. If the array does not exist already, it must be defined by

DIM GD[80]

The communication with the device takes place when the GDEV command is executed. The command may be specified with or without a string argument. If GDEV is followed by a string argument, the string is send to the device followed by the terminator character. If GDEV is not followed by an argument, the characters in the array GD[] is send to the device. See the examples below.

Example 1
Send message to a device at GPIB address 2. The message string is given as an argument to the GDEV command and must be terminated by a newline character. A return message is expected and is terminated by car return.

```
_GADDR = 2
_GWTERM = 10          ! Terminator character linefeed (ASCII value 10) on write
_GRTERM = 13          ! Terminator character return (ASCII value 13) on read
GDEV '?V1'            ! Send ?V1 followed by newline to the device
                      ! Array GD[] may have a return message like
                          GD[0]: 49  ASCII value of character 1
                          GD[1]: 51  ASCII value of character 3
                          GD[2]: 46  ASCII value of character .
                          GD[3]: 48 ASCII value of character 0
                          GD[4]: 32  ASCII value of character (space)
                          GD[5]: 86  ASCII value of character V
                      i.e. the return string was 13.0V followed by car return
```

Example 2
Send message to a device at GPIB address 2. The message string is specified in the array GD[] and must be terminated by a newline character. An ASCII = 5 is used to terinate the array string but is not send to the device. A return message is expected and is terminated by car return.

```
_GADDR = 2
_GWTERM = 5          ! Terminator character for array string
_GRTERM = 13         ! Terminator character return (ASCII value 13) on read
GD[0] = 63           ! ASCII value for ?
GD[1] = 86           ! ASCII value for V
GD[2] = 49           ! ASCII value for 1
GD[3] = 10           ! ASCII value for newline
GD[4] = 5            ! ASCII value 5, array string terminator
GDEV                 ! Send ?V1 followed by newline to the device
                     ! Note that the _GWTERM character is not send
                     ! Array GD[] may have a return message like
                          GD[0]: 49  ASCII value of character 1
                          GD[1]: 51  ASCII value of character 3
                          GD[2]: 46  ASCII value of character .
                          GD[3]: 48 ASCII value of character 0
                          GD[4]: 32  ASCII value of character (space)
                          GD[5]: 86  ASCII value of character V
                     i.e. the return string was 13.0V followed by car return
```

Example 3
Send message to a device at GPIB address 2. The message string must be terminated by a newline character. No return message is expected.

```
_GADDR = 2
_GWTERM = 10         ! Terminator character linefeed (ASCII value 10) on write
_GRTERM = 0          ! No return message from the addressed device
GDEV 'START'         ! Send START followed by newline to the device
                     ! No return message in array GD[]
```

Note: The following GPIB device addresses are in use:

1	Prema DVM
2	HP power supply
3	Histogarmming memory for MCA
5	ECB system
8	Temperature controller
11	XG control (FYS-RTG1)
12	HP Data Acquisition/Switch Unit (BW1)
14	Reserved

Appendix K

FORMAT specification of variables

The format in which a variable is output may be specified by the user. A format specification is included in the symbol table for each symbol, see appendix D. The command **FORMAT** may be used to show or change the format for symbols, e.g. FROMAT PROU shows the format for symbol PROU, while FORMAT PROU,'%5d' sets the format for PROU to %5d, i.e. integer with 5 digits precision. Formats specified with the FORMAT command are overwritten by the formats from the symbol table when a new symbol table is loaded with the _LO_S command.

Each format conversion specification in the symbol table or as argument in the FORMAT command has the following syntax:

% [flags] [width] [.prec] type-character

The % (percent sign) and the type-character must always be part of the format conversion specification while the flags, width, and .prec are optional. The format specification follows the standard for the C-language. The parts of the specification controls the following:

[flags]: The flag characters are -, +, #, 0, and blank. They can appear in any order and combination.

blank — If the first character of a signed conversion is not a sign, a blank is prefixed to the result. If both the blank and + options appear, then the blank option is ignored

\# — The value is converted to an alternative form. For d and s conversions, the option has no effect. For e, E, f, g, and G conversions, the result always contains a decimal point, even if no digits follow the point. For g and G conversions, trailing zeros are not removed from the result as they usually are.

0 — For d, e, E, f, g, and G conversions, leading zeros (following any indication of sign or base) are used to pad to the field width, no space padding is performed. If the 0 (zero) and - options appear, the 0 (zero) option is ignored. For d conversion, if a precision is specified, the 0 (zero) option is ignored.

\+ — The result of a signed conversion always begins with a + or - .

\- — The result of the conversion is left aligned within the field

[width]: An optional decimal digit string that specifies the minimum field width. If the converted value has fewer characters than the field width, the field is padded on the left to

the length specified by the field width. If the left-adjustment option flag (-) is specified, the field is padded on the right.

[.prec]: An optional precision. The precision is a . (dot) followed by a decimal digit string. If no precision is given, it is treated as 0 (zero). The precision specifies:

- the minimum number of digits to appear for the **d** conversion.
- the number of digits to appear after the decimal point for the **e** and **f** conversions.
- the maximum number of significant digits for the **g** conversion.
- the maximum number of bytes to be printed from a string in the **s** conversion.

type-character: A character that indicates the type of conversion to be applied, as follows:

d Converts the output to a signed decimal notation in the style [-]*dddd*. The precision specifies the minimum number of digits toappear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a 0 (zero) value with a precision of 0 (zero) is a null string. Specifying a field width with a 0 (zero) as a leading character causes the field width value to be padded with leading zeros.

e, E Converts the output to the exponential form [-]*d.dd*e[+/-]*dd*. There is one digit before the decimal point, and the number of digits after the point is equal to the precision specification. If no precision is specified, then six digits are output. If the precision is 0 (zero), then no decimal point appears. The E conversion character produces a number with E instead of e before the exponent. The exponent always contains at least two digits. However, if the value to be printed requires an exponent greater than two digits, additional exponent digits are printed as necessary.

f Converts the output to decimal notation in the format [-] *ddd.ddd*. The number of digits after the decimal point is equal to the precision specification. If no precision is specified, then six digits are output. If the precision is 0 (zero), then no decimal
point appears.

g, G Converts the output in the style of the f or e conversion characters (or E in the case of the G conversion), with the precision specifying the number of significant digits. Trailing zeros are removed from the result. A decimal point appears only if it is followed by a digit. The style used depends on the value converted. Style g results only if the exponent resulting from the conversion is less than -4, or if it is greater than or equal to the precision.

s Outputs a string, and bytes from the string are printed until the end of the string is encountered or the number of bytes indicated by the precision is reached. If no precision is specified, all characters in the string are printed.

If the result of a conversion is wider than the field width, the field is expanded to contain the converted result. No truncation occurs. However, a small precision may cause truncation on the right.

The type-character **s** must be used with all string variables, e.g. FINA, and is the only legal type-character with string variables. The type-characters **d**, **e**, **E**, **f**, **g**, and **G** are used with all numerical variables, and are the only legal type-characters with numerical variables.

EXAMPLES:

1. Output of a string variable: FINA = 'MINE'

```
>?FINA
                  format:  %s         format: %10s
                  = MINE              =      MINE
```

2. Output of a real variable: KUK = 5.5

```
>?KUK
                  format: %10.3g     format: %10.3f     format: %10d
                  =         5.5      =      5.500       =           6
```

2. Output of a real variable: KUK = .0000123

```
>?KUK
      flag        format: %10.3g     format: %10.3f     format: %10d
                  =   1.23e-05       =      0.000       =           0
      +           =  +1.23e-05       =     +0.000       =          +0
      +0          = +01.23e-05       = +00000.000       = +000000000
      +-          = +1.23e-05        = +0.000           = +0
```

2. Output of a real variable: KUK = -123456

```
>?KUK
      flag        format: %10.3g     format: %10.3f     format: %10d
                  =  -1.23e+05       = -123456.000      =    -123456
      0           = -01.23e+05       = -123456.000      = -000123456
      +-          = -1.23e-05        = -123456.000      = -123456
```

Appendix L

Restart of TASCOM after errors.

In case of error that does not allow TASCOM to restart normally, use one of the following procedures to get TASCOM running again. See also appendix G (page 73) on how to reload the ECB program.

1) Restart the ECB system.
Exit TASCOM with the EXIT command if possible.
Switch off the ECB system.
Check that the GPIB cable is connected correctly.
Switch on the ECB system.
Start TASCOM. If TASCOM does not start correctly, use the following procedure 2).

2) Restart without the ECB system.
Switch off the ECB system or disconnect the GPIB cable between the computer and the ECB system.
Start TASCOM without the ECB system.
Reset the motor parameters in the motor control block with the command _INIMCB.
Exit TASCOM with the EXIT command.
Switch-on/connect the ECB system.
Start TASCOM. If motor multiplexing is used, load the multiplexer configuration set up file. Load the motor parameter from the motor gearing file. If the name of the gearing file is rtg1_gea.tas, the command to load it is RTG1_GEA.
If TASCOM does not start correctly, use the following procedure 3).

3) Restart without the ECB system; load symbol table.
Switch off the ECB system or disconnect the GPIB cable between the computer and the ECB system.
Start TASCOM without the ECB system.
Load the ASCII symbol table with the _LO_S command, e.g. _LO_S'rtg1_sym'
Exit TASCOM with the EXIT command thereby saving a binary version of the symbol table just loaded. Note that this sets all symbols to default values.
Switch-on/connect the ECB system.
Start TASCOM. If motor multiplexing is used, load the multiplexer configuration set up file. Load the motor parameter from the motor gearing file. If the name of the gearing file is rtg1_gea.tas the command to load it is RTG1_GEA. Load the display parameter specification file.
Check TASCOM symbols like FINA, SEGN, DATA_DIR, TCOM_DIR, etc., and check the motor positions.
If procedure 3) does not work, use the following procedure 4).

4) Restart without the ECB system and without symbol table.

Switch off the ECB system or disconnect the GPIB cable between the computer and the ECB system.

The binary version of the symbol table is located in subdirectory .../symfil and has the name symfil.bin. Rename this file, e.g to symfil.bbb, so that TASCOM can not find it.

Start TASCOM. TASCOM reports that it can't find the symbol table and requests a new one to be loaded. Load the ASCII version of the symbol table with the _LO_S command, e.g. _LO_S'rtg1_sym'.

Exit TASCOM with the EXIT command thereby saving a binary version of the ASCII symbol table just loaded. Note that this sets all symbols to default values.

Switch-on/connect the ECB system.

Start TASCOM. If motor multiplexing is used, load the multiplexer configuration set up file. Load the motor parameter from the motor gearing file. If the name of the gearing file is rtg1_gea.tas, the command to load it is RTG1_GEA. Load the display parameter specification file.

Check TASCOM symbols like FINA, SEGN, DATA_DIR, TCOM_DIR, etc., and check the motor positions.

Appendix M

Linux. Quick Reference Guide.

alias - Displays or creates aliases

alias	Display alias
alias name 'alias-to name'	Create an alias

cd - Changes the current directory

cd [directory]

cd	Change to log-in directory (HOME directory)

chmod - Changes permission codes

chmod absolute-mode file

cp - Copies a file

cp [-p] source-file destination-file
cp [-p] source-file destination-directory
cp [-r][-p] source-directory destination-directory

-r Copies the directory and the entire subtree to the new location
-p Keep date and time of files

df - Displays statistics on free disk space

df [-k]

-k Reports statistics in kilobytes rather than 512-byte bloccks

diff - Compares text files

diff file1 file2

du - Displays a summary of disk usage

du

exit - Causes the shell to exit

exit

find - Finds files

find pathname [-name expression] [-print]

-name expression	Searches for filenames that match the named expression
-print	Displays path to all matching files

grep - Searches a file for a pattern
grep pattern file

head - Displays the beginning of a file
head [-count] file

-count Number of lines to display, default is 10

lpr - Sends files to the printer queue for printing
lpr [-h] [-Pprinter] file

-h	Supress printing of the cover page
-Pprinter	Sends the file to the specified printer

less - Displays a file one screenful at a time
less file

Up-arrow, Down-arrow	Scroll one line
Page-Up-key, Page-Down-key	Scroll up screen
q Quit	

ls - Lists information about files
ls [-al] [file]

-a	List all entries in a directory, including dot (.) entries
-l	Display mode, owner, group, size, and time for each file
ll	Identical to ls -l

man - Displays manual pages
man title
man -k keyword

mkdir - Makes a directory
mkdir directory

more - Displays a file one screenful at a time
more file

d	Scroll down half a screen
u	Scroll up half a screen
space	Scroll down a full screen
q	Quit

mv - Moves (Renames) files and directories
mv file1 file2
mv file1 directory
mv directory1 directory2

ps - Displays current process status
ps [-la]

-a all
-l long listning

pwd - Displays pathname of current directory
pwd

rm - Removes files and directories
rm [-ifr] file

-i Prompts before deleting each file
-f Does not prompt before deleting files
-r Deletes directory trees

rmdir - Removes an empty directory
rmdir directory

scp - Copies files between a local and a remote host
scp [-p] source destination

-p Keep date and time of files

ssh - Connects the local host with a remote host
ssh remote-host

su - Log into another user account
su name-of-new-account

tail -Displays the end of files
tail [-f] file

-f Keep looking in the file

w – Shows who is logged on and what they are doing

who - Identifies users currently logged in
who

write - Sends message to other users
write user [line]
write user@node [line]

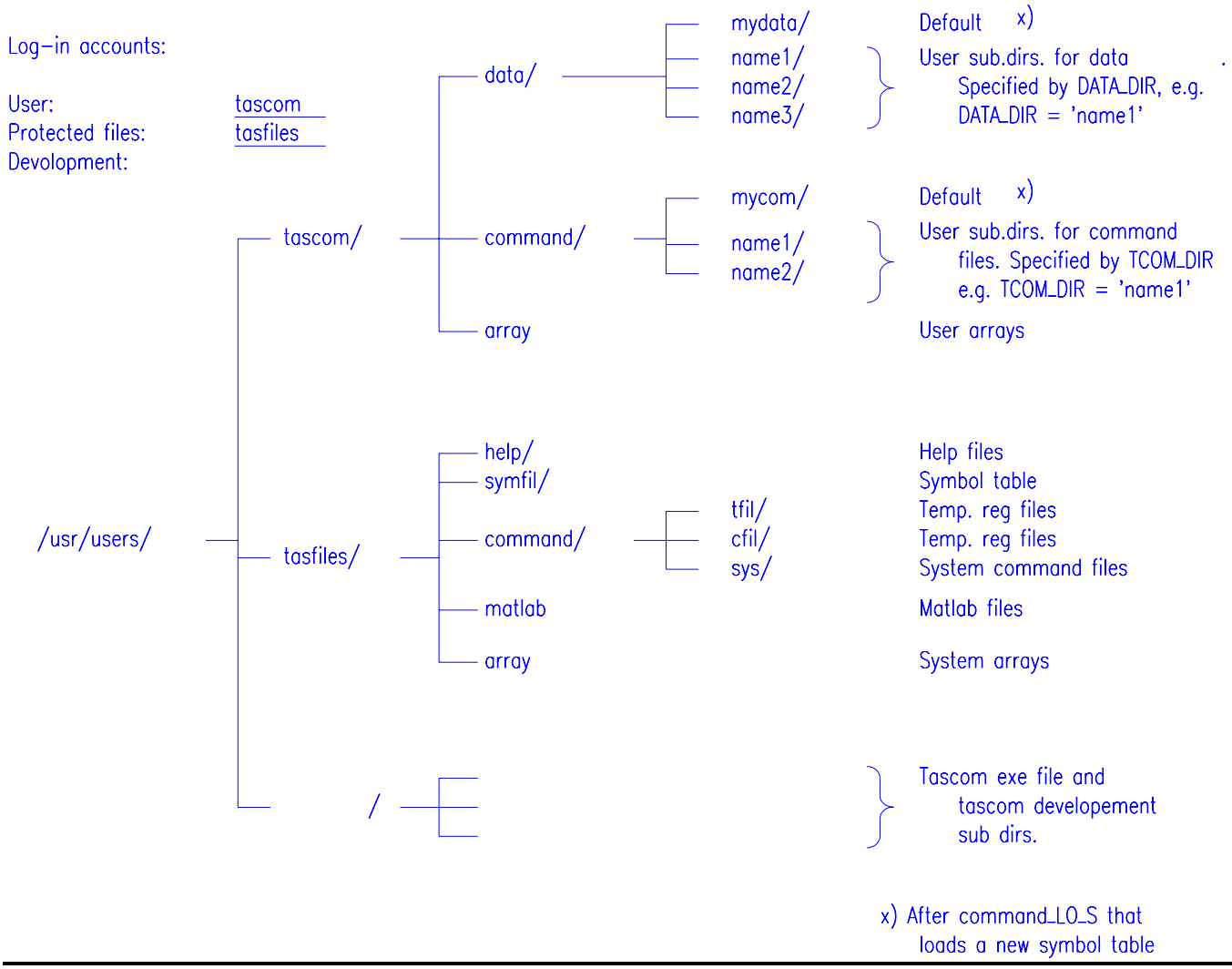
Appendix N

Emacs editor. Quick Reference Guide.

Start the editor	emacs <file-name>	
Move the cursor by:		
character	CTRL(b) CTRL(f)	
line	CTRL(p) CTRL(n)	
screen	CTRL(v)	
Move the cursor to:		
top of file	Esc <	Esc is function key F11
bottom of file	Esc >	on the DEC-alpha keyboard
start of line	CTRL(a)	
end of line	CTRL(e)	
Kill text:		
from cursor to end of line	CTRL(k)	
from mark to cusor	CTRL(w)	
in a rectangle mark,cursor	CTRL(x) r k	
Restore text at cursor	CTRL(y)	
Restore rectangle at cursor	CTRL(x) r y	
Search text:		
forward	CTRL(s)	
backward	CTRL(r)	
repete	CTRL(s) or CTRL(r)	
Set a mark	CTRL(space)	
Put text before cursor	CTRL(y)	
Put rectangle at cursor	CTRL(x) r y	
Save without leaving emacs	CTRL(x) followed by CTRL(s)	
Save changes and leave emacs	CTRL(x) followed by CTRL(c)	

Appendix O

TASCOM Directory tree.



Appendix P

The PLOT program

The following is a description of a program called PLOT that is used to create a 'live' display of the data measured by TASCOM while the data are accumulated. PLOT can also display old data stored in data files created by TASCOM. In addition, PLOT can print out the displayed picture on paper.

PLOT is started by typing plot followed by up to five arguments arg1 to arg5. If no arguments are typed, a help message is shown. See the example below.

plot

```
In the directory where the data files are saved type:
     plot arg1 arg2 arg3 arg4 arg5
 where the arguments are:
 arg1:
  1   Tascom scan on-line plot
  2   Data from .dat file
  3   Linear detector on-line plot
  4   Data from .dar file from linear detector
  5   Area detector (128 x 128) on-line plot
  6   Data from .dar file from area detector
  7   Array data from p_a[]
 arg2:
  0   No hard copy
  1   Hard copy after each scan
  2   Make PostScript file
  3   Hard copy, several scans, default order
  4   Hard copy, several scans, page 1 top left
 arg3:
  0   Linear Y-axes with error bars
  1   LOG10 Y-axes with error bars
  2   Linear Y-axes without error bars
  3   LOG10 Y-axes without error bars
 arg4:
  0   Gaussian fit to data points
  1   Lorentzian fit to data points
  2   Triangular fit to data points
  3   No fit to data points
  4   Tascom fit to data points
 arg5:
  0   Do not connect data points
  1   Connect data points
```

In the following some typical uses of the PLOT program and the transfer of TASCOM variables to PLOT will be discussed.

Tascom scan on-line plot (arg1 = 1)
This mode of PLOT creates a 'live' display of a TASCOM scan while the scan is being measured, for example the position of a motor on the x-axis and the measured counts on the y-axis. To enter this mode, PLOT must be started in the directory where TASCOM stores the files with plot data and that depends on the TASCOM variable _PLOTDIR. If _PLOTDIR=0, the plot data files are in the directory specified by DATA_DIR, ie the same directory as the data files. If _PLOTDIR=1, the plot data files are in directory

/usr/users/tascom/data. The names of the plot data files are plot.mat and plot.plt and they are updated by TASCOM during the scan with variables and data used by PLOT. When the scan is finished, EOF is written at the end of the file plot.plt and when PLOT reads that, it starts the actions wanted at the end of the scan, for example tries to make a fit to the data or make a printout of the scan. When a new scan is started, the file plot.plt is deleted and a new plot.plt is created, but no new plot is displayed before four points of the new scan have been measured.

To start PLOT in this mode, open a window, cd to the right directory and type, eg,
PLOT 1
In this example arguments arg2 to arg5 are not specified and are set to 0. If instead a plot without a fit to the data points at the end of the scan is wanted, the command that starts PLOT could be
PLOT 1 0 0 3
A plot window should now appear. In the original window the user is asked to type the number of seconds between update of the plot. One second is minimum. A number of status messages will appear in the original window during the running of PLOT. PLOT is terminated by typing ctrl(c) in the original window.

In TASCOM the command DPLO determines the x and y cordinates of the plot, eg,
DPLO MOTOR,I
will result in the variables MOTOR and I being written to the plot file plot.plt when each point in the scan has been measured. PLOT reads the file every time it wants to update the plot.

Several TASCOM variables specify the plot created by PLOT. These variables are written by TASCOM at the start of each scan to the plot file plot.mat from where they are read by PLOT. The variables are:

XMIN XMAX	Min and max for the x-axis, eg, XMIN=0 XMAX=100 If XMAX=XMIN, autoscaling on the x-axis.
LIMY	Scaling parameter for the y-axis: LIMY=0: autoscaling LIMY=1: use YMIN and YMAX LIMY=2: autoscaling with baseline 0 LIMY=3: autoscaling with baseline YMIN
YMIN YMAX	Min and max for the y-axis if LIMY=1, eg, YMIN=0 YMAX=10000
	Background correction at Tascom fit (arg4=4):
NBAC	Number of background points
BACK	Background counts
	If both BACK=0 and NBAC=0, no background correction

Plot of old Tascom data files (arg1 = 2)
This mode of PLOT creates a display of data from a Tascom data file. To enter this mode, PLOT must be started in the directory where the data file is stored.

To start PLOT in this mode, open a window, cd to the right directory and type, eg,
PLOT 2
A plot window should now appear. In the original window the user is asked several questions to specify the plot, eg the name of the data file and which variables from the file should be plotted. PLOT is terminated by typing ctrl(c) in the original window.

Linear detector on-line plot (arg1 = 3)
This mode of PLOT creates a 'live' display of data from a linear detector, ie the data in the P2126a Histogramming memory NIM module. A part of the PC's memory is shared by TASCOM and PLOT, and the data in the histogramming memory module is read by TASCOM to this shared memory from where PLOT reads the data. The histogramming memory module is read and the shared memory thereby updated when the read detector command (MCRE or RMCA) is executed. If a counting is started with the COUN command and if the TASCOM variable DD_DISP=1, then the shared memory is updated approximately once a second. This can produce a 'live' display of the detector data. Also the SHOW_DD command may produce a 'live' display, see Chapter 16.

In this mode PLOT does not read any files and may thus be started from any directory. To start PLOT in this mode, open a window and type, for example,
PLOT 3
A plot window should now appear. In the original window the user is asked to type the number of seconds between update of the plot. One second is minimum. Also the highest and the lowest channels that should be displayed must be specified. PLOT is terminated by typing ctrl(c) in the original window.

Plot of old Tascom detector data files (arg1 = 4)
This mode of PLOT creates a display of data from a Tascom detector data file, ie a file with extension .dar. To enter this mode, PLOT must be started in the directory where the detector data file is stored.

To start PLOT in this mode, open a window, cd to the right directory and type, eg,
PLOT 4
A plot window should now appear. In the original window the user is asked several questions to specify the plot, eg the name of the data file and the highest and lowest channels that should be displayed. PLOT is terminated by typing ctrl(c) in the original window.

Array data from p_a[] (arg1 = 7)

This mode of PLOT creates a display of the TASCOM array P_A[]. To enter this mode, PLOT must be started in the directory where TASCOM stores the files with the plot array data and that depends on the TASCOM variable _PLOTDIR. If _PLOTDIR=0, the plot array files are in the directory specified by DATA_DIR, ie the same directory as the data files. If PLOTDIR=1, the plot array files are in directory /usr/users/tascom/data. The names of the plot array files are plot.pa and plot.pam and they are updated with the TASCOM command P_A_OUT. A new plot is generated when the files are updated.

In TASCOM the array P_A[] must be defined as either a one dimensional array or a two dimensional array with the first index being 2, for example

DIM P_A[n] or

DIM P_A[2,n]

In the first case the array elements are plotted on the y-axis while x is incremented by 1. In the second case the points plotted are x,y = P_A[0,n] , P_A[1,n].

To start PLOT in this mode, open a window, cd to the right directory and type, eg,

PLOT 7

In this example arguments arg2 to arg5 are not specified and are set to 0. If instead a plot of the array data without a fit to the data points is wanted, the command that starts PLOT could be

PLOT 7 0 0 3

A plot window should now appear.

Several TASCOM variables specify the plot created by PLOT. These variables are written by TASCOM at the start of each plot to the plot file plot.pam from where they are read by PLOT. The variables are:

_AXMIN _AXMAX	Min and max for the x-axis, eg, _AXMIN=0 _AXMAX=100 If _AXMAX = _AXMIN, autoscaling on the x-axis.
_ALIMY	Scaling parameter for the y-axis: _ALIMY=0: autoscaling _ALIMY=1: use _AYMIN and _AYMAX _ALIMY=2: autoscaling with baseline 0 _ALIMY=3: autoscaling with baseline _AYMIN
_AYMIN _AYMAX	Min and max for the y-axis if _ALIMY=1, eg, _AYMIN=0 _AYMAX=10000
_AXLAB _AYLAB	Labels for x and y axis, string variables. Eg: _AXLAB='MOT' _AYLAB='Counts'
_ATEXT	Plot headline, string variable. Eg: _ATEXT = 'Testing'

Making prints (arg2)

After completion of each scan, a print is made depending on the value of arg2. The print is made on the system's default printer, ie the printer used when the Linux lpd command is given without the –P switch to specify a printer.

If arg2=1, a print is made of the scan just completed with one scan on each piece of paper.

If arg2=2, a postscript file is made for later printing.

If arg2=3 or arg2=4, a postscript file is made that is printed after a specified number of scans, ie with more that one scan on each piece of paper. When PLOT is started with arg2=3 or arg2=4, the user is asked

Number of pictures in each Postscript file:

In response type a value between 1 and 25. If a print is wanted before all the scans are made, type ctrl(c). PLOT asks the question

Abort[a] or complete[c] print:

Answer c if you want a print of the scans remaining to be printed and then continue PLOT. Answer a if you want to print the scans remaining and then leave PLOT.

Fit of scan (arg4)

After completion of each scan, a fit of the data is made assuming that a peak has been measured and depending on the value of arg4.

If arg4=4, a fit is made using the same algorithms that TASCOM uses to analyse peak data and the same variables are calculated, see Chapter 10. Therefore, when PLOT is started with arg4=4, the user is asked to input NBAC, the number of background points. If NBAC=0, the user is futher asked to input BACK, the fixed background.

Index.

E

F

G

H

I

J

T

U

V

W

X

Y