EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN/SL/93-?? (AP)

The MAD Program
(Methodical Accelerator Design)
Version 8.10
Programmer's Reference Manual

Hans Grote
F. Christoph Iselin

Abstract

MAD is a tool for charged-particle optics in alternating-gradient accelerators and beam lines. It can handle from very large to very small accelerators, and solve various problems on such machines.

This document outlines the data structures used in MAD. It intends to help programmers to add new features to the program.

Geneva, Switzerland
September 15, 1994

The MAD program contains the following copyright note:

CERN

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

Program name: MAD --- Methodical Accelerator Design

CERN program library entry: T5001

Authors or contacts: Hans Grote, F. Christoph Iselin
SL Division
CERN
CH-1211 GENEVA 23
SWITZERLAND
Tel. [0041] (022) 767 36 57
FCI at CERNVM.BITNET

Preface

The MAD framework should make it easy to add new features in the form of program modules. The authors of MAD hope that such modules will also be contributed and documented by others. The contributions of other authors are acknowledged in the relevant chapters.

Misprints and obscurity are almost inevitable in a manual of this size. Comments from readers are therefore most welcome. They may also be sent to one of the following BITNET addresses:


```
FCI   at CERNVM.CERN.CH.bitnet
HANSG at CERNVM.CERN.CH.bitnet
KEIL  at CERNVM.CERN.CH.bitnet
```

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1. A Short Introduction to ZEBRA

1.1 Why Use ZEBRA?

For various reasons MAD [12] is written in the FORTRAN 77 language. While this language offers certain advantages, it suffers from its lack of dynamic data structuring facilities. The only data structures available are arrays of fixed length, and common blocks for shared data. There is no standard way for allocating arrays dynamically or to change their length at run time.

To overcome this defect, the ZEBRA system [3] has been designed and written at CERN. It allows the management of large amounts of data in a computer by providing the functions required to construct a logical graph of the data and their interrelations. The data are stored in FORTRAN common blocks, called *stores*. Each store can be subdivided into up to 20 *divisions*. Relations between the basic units of data, or *banks*, are expressed by linking banks to form more complex structures. A bank is accessed by specifying its address in a given store. Such addresses (called *links*) are kept inside the banks or in *link areas* inside common blocks.

- The memory management part of ZEBRA is handled by the `MZ` package. Utilities are available for allocating, reorganizing, sorting and deleting banks and data structures.
- Individual banks, data structures or complete divisions can be output with the `FZ` package.
- Direct access files for data structures and the management of the data by keywords are provided by the `RZ` package. This part of ZEBRA is presently not used in MAD.
- Dumps and verification of ZEBRA structures and documentation tools are available in the `DZ` package.

The following sections give a short introduction to the details of ZEBRA which are required for understanding MAD's data structure.

1.2 Logical Data Structures

1.2.1 The Bank

Assume that we wish to store the data related to a given *object*, say an element (magnet) definition, containing details about its *attributes* (length, strength, etc.). Using a call to the ZEBRA routine `MZBOOK`, we can allocate a contiguous area of storage with a given length. The location of this area is returned by `MZBOOK` as a *base address* which has to be used in any *reference* to that area. The unit of storage is called a *bank*, and in FORTRAN code it will be referenced as in

```
Q(L+MELEN), Q(L+MEK1Q), ...
```

where `Q`, by convention, is the name of the FORTRAN array underlying the data store, and `L` is the base address, provided by `MZBOOK`. It is the index into the store of the word preceding the first data word of the bank.

ZEBRA banks may contain data of different types. Thus we can address other data words in the bank for example as integers:

```
IQ(L+MBAT), IQ(L+MBSP)
```

For structuring purposes ZEBRA requires no knowledge of the actual contents of a bank. The internal details of data in a bank are the responsibility of the user, and it is vital to maintain an adequate documentation of the bank contents. However, for input and output ZEBRA has to know the type of the bank contents. For this purpose it uses *format codes*, explained later.

1.2.2 The Linear Structure

The number of element definitions in MAD is variable and may be very large. To realize sets of objects of the same kind, MAD uses the ZEBRA construct of a *linear structure*. A linear structure consists of a series of linked banks, with each bank holding a reserved system word, called the *next link*, i. e. the base address of the next bank in the set. The next link of the last bank in a linear structure is zero, indicating that there is no next bank. The *address of a linear structure* is simply the address of its first bank.

A linear structure may be visualized as in Figure 1.1. Note that the first bank in the chain has been booked last.

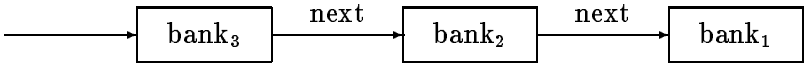

Figure 1.1: A Linear Structure containing Element Definitions

The next link is stored in the word `LQ(LCELM)` of the bank, with the vector `LQ` displaced by `EQUIVALENCE` with respect to the vectors `Q` and `IQ`. The elements of the set can then be accessed in their order by a loop of the form:

```
      LCELM = LFIRST
10    IF (LCELM .NE. 0) THEN
         ...
         GO TO 10
      ENDIF
```

Banks are connected dynamically at run time. Because each bank has one word set apart to connect it to the rest of the structure, an arbitrary number of objects can be linked to the set. The order of the banks in the set may or may not have significance, and the user is allowed to change it if required.

1.2.3 The General Data Structure

In the general case, more complex structures are needed. Often the address of a bank or of a linear structure is stored in another bank. It is then called a *down link*. A given bank may have many down links, and it can depend on a logically yet higher bank through a down link in that bank. The down links thus allow to construct a tree, and at each node there may be a single bank or a linear structure.

In MAD, for example, all command keywords form a linear structure that depends on the bank describing the master keyword `KEYWORD`. All elements of a given type form a linear structure that depends on the bank describing the corresponding element keyword; it is shown schematically in Figure 1.2.

All links described so far are stored by ZEBRA as part of the bank concerned. The down and next links are referred to collectively as *structural links*, as they represent the basic connections of the data structure.

1.2.4 Reverse Links

Each ZEBRA bank contains a link pointing to the bank on which the whole linear structure of which it is a member depends. This link is called an *up link*. The value of the up link is zero if the bank is at the top of the tree structure. The up link is useful for moving towards the root of the tree.

Each bank also has an *origin link*, which points to the structural link supporting the bank. The origin link is usually of no interest to the user, its purpose is to free the user from having to remember the supporting link.

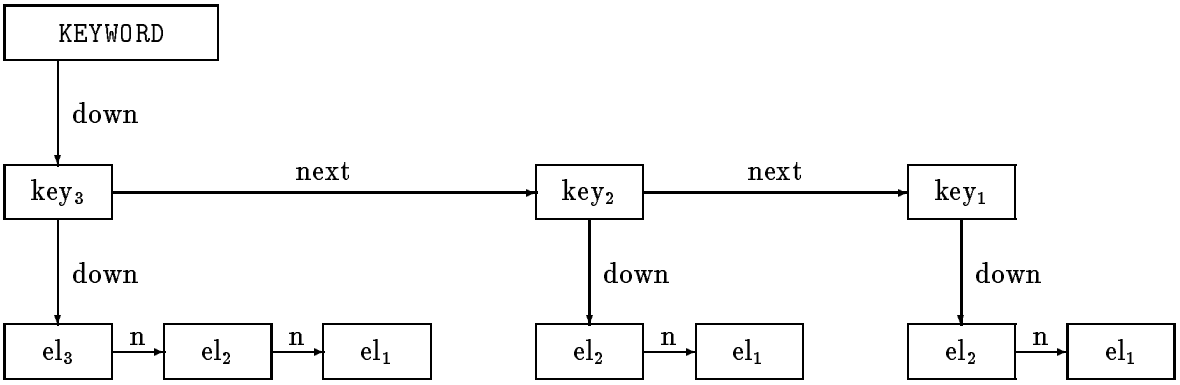

Figure 1.2: Example of a General Data Structure

The up link and the origin link are known as *reverse links* . A summary of the four types of links is given in Figure 1.3.

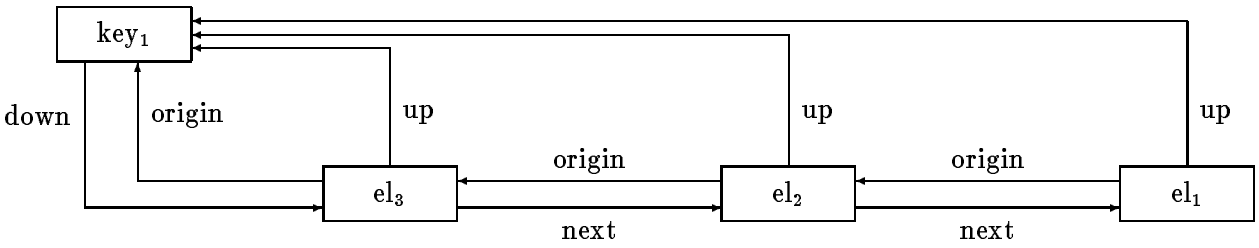

Figure 1.3: Schematic Overview of ZEBRA Links

1.2.5 Reference Links

The links described above are an integral part of the data structure. ZEBRA also permits the program to establish links between various banks which are not part of the structure, but which may represent relations between banks. These links are known as *reference links*. A bank may contain many reference links, and their use is at the discretion of the user. ZEBRA merely has the task to change their values in the case that banks are moved in memory for whatever reason.

Reference links, in contrast to the links mentioned so far, may point to any place inside a bank and not only to its *status word* (see Figure 1.4).

1.3 Physical Storage

The banks just described must be mapped onto the physical storage of the computer. In MAD a common block is declared for this purpose by the statements

```
PARAMETER          (MEMMIN = 100 000)
PARAMETER          (MEMLEN = 500 000)
COMMON /MEMORY/    FENCE, LQ(MWFLT*MEMLEN)
SAVE               /MEMORY/
INTEGER            IQ(MWFLT*MEMLEN)
REAL               FENCE(2), Q(MWFLT*MEMLEN)
DIMENSION          DQ(MEMLEN)
EQUIVALENCE        (IQ(1), Q(1), DQ(1), LQ(9))
EQUIVALENCE        (LROOT, LQ(1)), (LLUMP, LQ(2))
```

It is made known to ZEBRA by calling the routine MZSTOR (Section 35.20). The area FENCE is required by ZEBRA. It helps detecting out-of-bound indices. We note that the effect of the EQUIVALENCE statement is to offset the arrays IQ and Q with respect to LQ by eight words. This permits a simple form of subscripts in the references to data words and to links. The n^{th} data word is addressed as Q(L+n) or IQ(L+n), and the m^{th} link as LQ(L-m). The first 8 words of the array LQ are reserved for permanent links. At present only two of them are used as explained later.

Every bank allocated by ZEBRA has the layout shown in Figure 1.4.

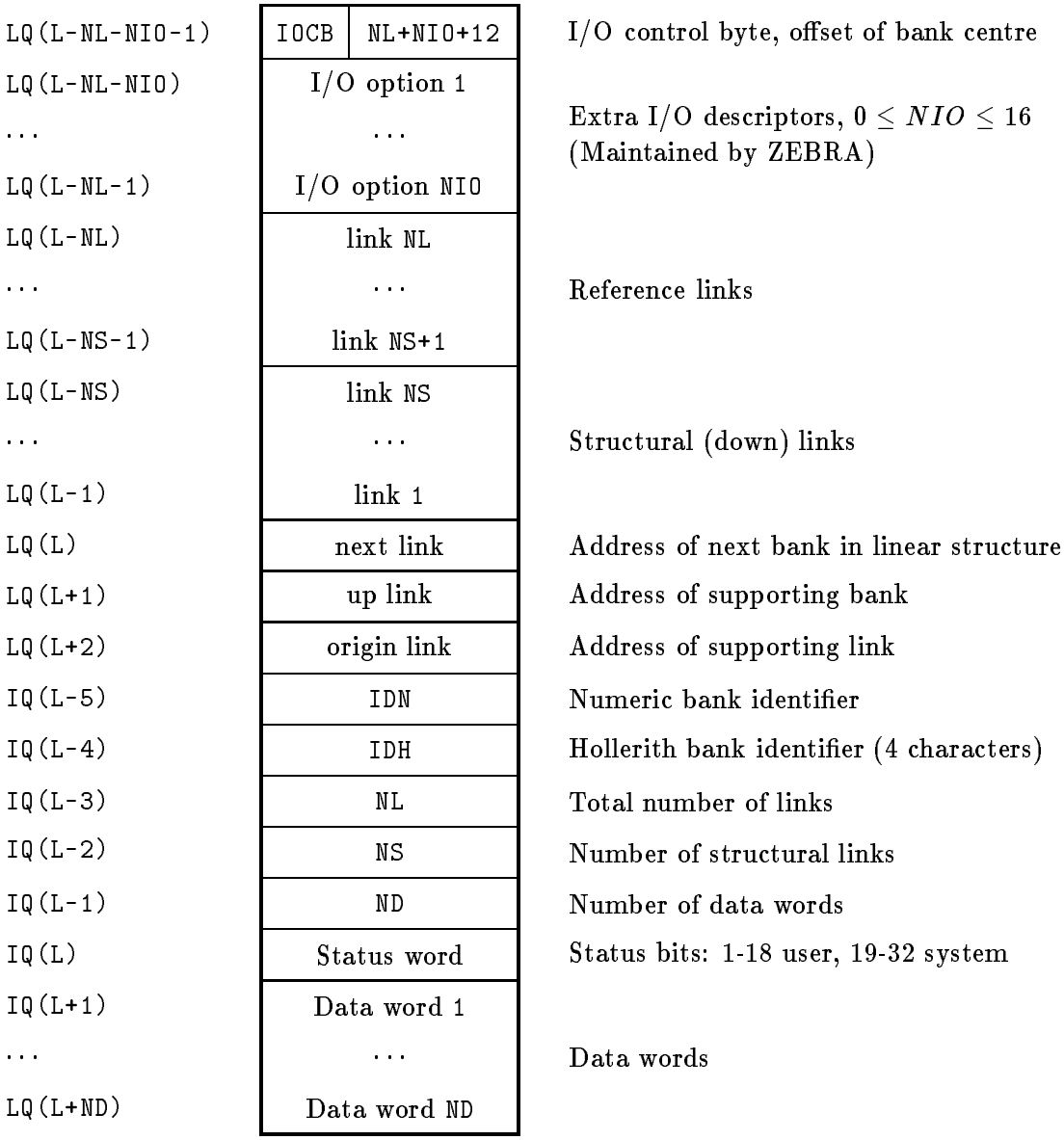

Figure 1.4: Layout of a ZEBRA Bank

The central part of the bank starts with the next link, accessed as LQ(L). The up link at LQ(L+1) points to the bank supporting the linear structure of which the bank is a member. It is zero if the bank is at the root of the tree. The origin link at LQ(L+2) points to the link through which the bank is reached. These three links are present in every bank and are not counted in NL and NS.

The two words IQ(L-5) and IQ(L-4) contain the numeric and Hollerith bank identifiers, IDN and IDH. Usually all banks of a linear structure have the same IDH, but different IDN's to permit identification of a particular bank. IDN is assigned by ZEBRA according to given rules, but the user may freely change it after bank creation. Words IQ(L-3) and IQ(L-2) hold the total number of links

(NL) and the number of structural links (NS) respectively, and the word IQ(L-1) holds the number of data words (ND). The status word at IQ(L) provides bits 1 to 18 for user status bits, while bits 19 to 32 are reserved for ZEBRA.

With this format the smallest possible bank (NL=NS=ND=0) occupies 10 words. The store size can be changed by altering the parameters MEMLEN and MEMMIN, or by selecting one of the *Historian*^{Plus} flags BIG or SMALL when compiling MAD. MWFLT is used to double the storage capacity for the version in *double precision*.

1.3.1 Divisions

The dynamic store may be physically divided into *divisions*. When MZSTOR initializes a store, it automatically creates three divisions, one for ZEBRA, and two for the user. MAD uses division 1 for short-lived data, and division 2 for data which must be kept over longer times. It uses no further divisions. The *dynamic store* itself is referred to by the number 0.

1.3.2 Link Areas

The user can store bank addresses or links, for ease of manipulation, in a user-defined area, or *link area*. Link areas must be kept in common blocks, and declared to ZEBRA with a call to MZLINK. ZEBRA will automatically update these links if banks must be moved.

1.3.3 Double Precision Data

There is no certitude that double precision data are properly aligned when stored in banks. If double precision data must be stored in a bank, they must therefore be moved between the bank and local double precision variables by calls to UCOPY (Section 36.4), and accessed from the local variables.

1.3.4 Character Data

The FORTRAN standard does not allow equivalencing of character data with numeric data. ZEBRA however supports the data type *Hollerith*. Such data can be moved between a bank and local CHARACTER variables by calls to UCTOH and UHTOC (Section 36.4), and accessed from the local variables. There must be enough words reserved in the bank to accommodate the desired number of characters.

1.3.5 Working Space

Often program modules of MAD require temporary working space. MAD uses two methods for this purpose:

1. Working space is allocated in banks in division 1. This mechanism is used when working space contains no double-precision data. After execution of the program module the whole division 1 is wiped out by a call to MZWIPE.

2. Working space is allocated in a stack-like fashion using MZWORK. This method is used by modules requiring double-precision data. The actual working array is the vector DQ, declared as REAL or DOUBLE PRECISION according to the program version, and equivalenced to IQ. During execution of the module MAD maintains two indices in the common block

   ```
   COMMON /WSTACK/ IWORK, NWORK
   ```

   IWORK is the index into DQ of the last work used for working storage, and NWORK is the index into DQ of the last word allocated by the latest call to MZWORK. Assuming that a subroutine wants to allocate an array of length ISIZE, it executes the following code sequence upon entry:

```
ISAVE = IWORK
ITEMP = IWORK
IWORK = IWORK + ISIZE
IF (IWORK .GT. NWORK) THEN
   CALL MZWORK(0, DQ(1), DQ(NWORK+1), 0)
   NWORK = IWORK
ENDIF
```

It may then use the area from DQ(ITEMP+1) to DQ(ITEMP+ISIZE) freely. Before exit it executes the code

```
IWORK = ISAVE
```

This does not free the working space for bank allocation, but it makes it available as working space for other subroutines. When the program module has finished, it executes the sequence

```
IWORK = 0
NWORK = 0
CALL MZWORK(0, DQ(1), DQ(1), -1)
```

to free the whole working space. This method minimizes the number of calls to MZWORK, and thus the time overhead for managing the working storage.

Since a call to MZWORK also wipes out division 1, the two methods cannot be mixed.

1.4 Dropping Banks and Garbage Collection

Initially the dynamic store contains only a few banks in the system division. As banks are created, the occupied space increases and the free space decreases. By calling MZDROP the user may *drop* banks which are no longer needed. MZDROP logically removes banks, or whole sub-structures, from the surrounding structure and marks the banks as dropped. The dropped banks remain intact in storage and reference links pointing to them continue to point to valid information.

If free space is not sufficient to satisfy a further request for creating a bank, ZEBRA will recuperate the space occupied by dropped banks. This process, called *garbage collection*, moves all active banks to form one contiguous area for each division, and updates all links. Links pointing to active banks are changed to point to the new positions, Any *reference link* pointing to a dropped bank is reset to zero. Garbage collection is triggered automatically, but the user may also request it by a call to MZGARB.

For garbage collection to function ZEBRA has to know *all* links used by the program. It is essential that the user keeps all bank addresses in locations known to ZEBRA, either in the *link part* of banks, or in a *link area*. Link areas are created in common blocks by calls to MZLINK. Any link kept elsewhere will be obsolete after garbage collection.

Chapter 2. MAD Data Structures

2.1 Conventions

ZEBRA Data Types: For structuring purposes ZEBRA requires no knowledge of the actual contents of a bank. The internal details of data in a bank are the responsibility of the user, and it is vital to maintain an adequate documentation of the bank contents. However, for input and output ZEBRA needs to know the type of the bank contents. Some banks in MAD contain data words which have all the same type. In this case the data type is declared to ZEBRA in the booking `MZBOOK` call.

Often the types of data words are declared explicitly on a word by word basis. In this case MAD declares the bank to be of type 7 (*self-defining*) and stores *format codes* along with the data. These have the form $16 * l + t$, where l is the number of words and t is the ZEBRA data type for the following data. Permissible data type codes are listed in Table 2.1.

Table 2.1: ZEBRA Data Types

0	unknown data type	1	bit string	2	integer
3	single precision	4	double precision	5	Hollerith
7	self-defining				

Links: Following the ZEBRA conventions all links have names beginning with an `L`. Links residing in ZEBRA banks are called `LQ`. In graphical representations of data structures this manual shows structural links (supporting links) as solid lines; reference links are shown as dashed lines. Reverse links are not shown in the figures. Solid boxes represent data banks or parts thereof; dashed boxes indicate substructures described elsewhere.

Parameters: Many constant integers have been defined in MAD by FORTRAN `PARAMETER` statements to allow easy change. The names of these parameters all begin with an `M`. Most parameters are documented together with the data structures they describe. The following parameters, defined in the comdeck `IMPLICIT`, are available to all subroutines:

`MWFLT` Number of ZEBRA words needed to store a real value, it has the value 1 (single precision) or 2 (double precision).

`MREAL` ZEBRA data type code for a MAD real, it has the value 3 (single precision) or 4 (double precision).

`MCWRD` Number of characters that can be stored in a word, it is 8 (single precision) or 4 (double precision).

`MCNAM` Number of characters allowed in an identifier (normally 16).

`MWNAM` Number of words required to store an identifier (`MCNAM/MCWRD`).

2.2 The Root Bank

The entire data structure of MAD forms a single tree in ZEBRA division 2. This allows the complete data structure to be dumped on disk or reloaded with a single ZEBRA call. The root of the tree is the *root bank*. Its address LROOT is stored in common block MEMORY.

The root bank has 20 structural and 20 reference links, but no data words. Links used are listed in Table 2.2. Parameters defining the root bank structure are set in comdeck STRGROUP. Unused links are available to program modules of MAD, provided they are freed upon exit.

Table 2.2: Structure of the Root bank

LQ(LROOT-MDKEY) ... LQ(LROOT-MDKEY-3)	} links for the keyword directory (Section 2.9)
LQ(LROOT-MDBNK) ... LQ(LROOT-MDBNK-3)	} links for the data bank directory (Section 2.9)
LQ(LROOT-MDEXP)	link for the expression table (Section 2.6.4)
LQ(LROOT-MDVAR)	link for the variable reference table (Section 2.6.4)
LQ(LROOT-MRKEY)	link for the keyword tree (Section 2.3)
LQ(LROOT-MCSEQ)	link for the current beam line expansion (Section 2.10)
...	central part of the bank

2.3 Keyword and Command Tree

The first action of MAD is to build the command keyword KEYWORD, known as the *master keyword*. The KEYWORD bank is supported by the link MRKEY in the root bank. While reading the command dictionary, the master keyword serves as a template. Each new *keyword* bank built is linked to the linear structure supported by link 1 of the master keyword bank.

When a command or definition is decoded, it is linked to the linear structure supported by link 1 of the corresponding keyword bank. Definition banks are always kept unless redefined. If a command bank is labeled or if it belongs to a subroutine, it is kept in store; otherwise it is dropped after execution.

Some command attributes need additional banks to hold their values. For the i^{th} attribute such a bank is linked to the i^{th} link of the *command bank* as described in Section 2.6.1. Command attributes can take their values from three sources:

1. If the attribute is not entered at all, the so-called *first default* is used. For values used refer to Table 2.8.

2. If the attribute name is entered, but without a value, the so-called *second default* is used. This value is normally specified in the command dictionary. It is stored in a bank having the same format as a command bank, supported by link 2 of the keyword bank.

3. If the attribute value is entered by position, or together with its name, the value is used as entered.

Link 3 of an *element definition* keyword (e. g. QUADRUPOLE) points to the generic *class object* of the same name. This object is built when the keyword is decoded. Note that the up link LQ(L) of a command or definition bank points to the relevant keyword.

The tree obtained is called *keyword tree*. A simplified example is depicted in Figure 2.1.

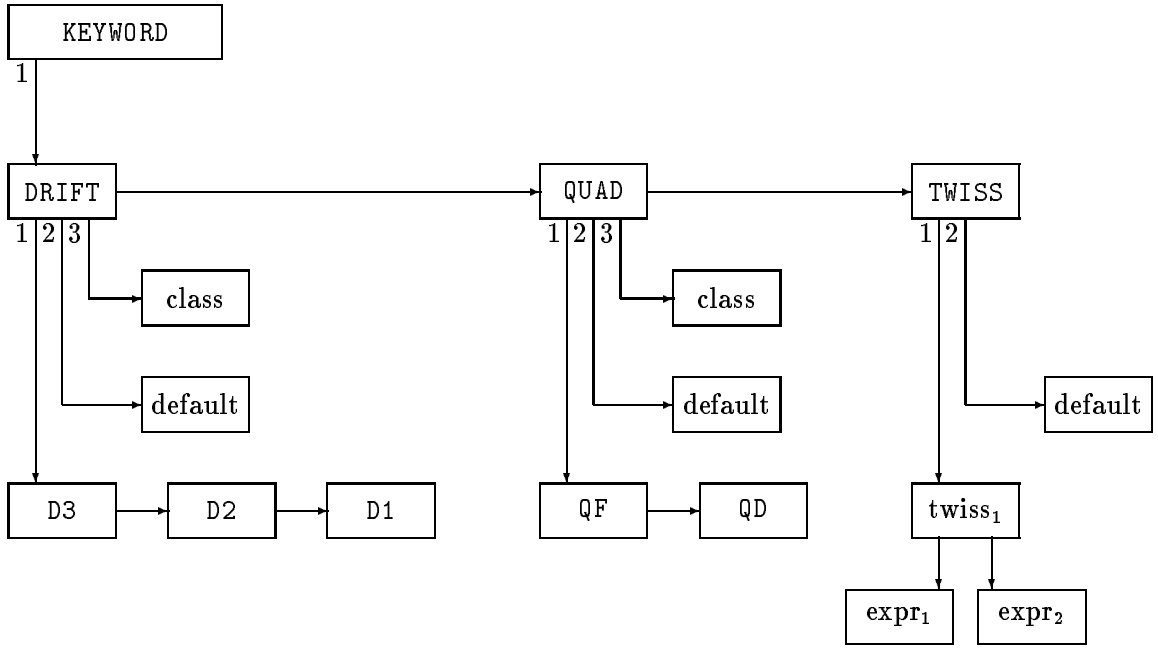

Figure 2.1: Structure of the Keyword Tree

2.4 Keyword Banks

Keyword banks have the structure shown in Table 2.3. Structure parameters are found in the comdecks BANKHEAD and KEYGROUP.

Table 2.3: Structure of Keyword Bank
(LCKEY is the address of the keyword bank)

LQ(LCKEY-3)	supports the generic class object (For element definitions only, Section 2.6)
LQ(LCKEY-2)	supports the banks containing the second defaults (Same format as a command bank, Section 2.6)
LQ(LCKEY-1)	supports the linear structure of commands or definitions of this class
...	central part of the bank
IQ(LCKEY+MBFRM)	format code: 4 integers ($4*16+2$)
IQ(LCKEY+MBNAM)	directory index in keyword directory (Section 2.9)
IQ(LCKEY+MBPR)	process code (Section 2.5)
IQ(LCKEY+MBSP)	subprocess code (Section 2.5)
IQ(LCKEY+MBAT)	number of keyword attributes NKAT, counting arrays as a single attribute
IQ(LCKEY+MBAT+1) ... IQ(LCKEY+MBAT+NKAT*MKSIZ)	} Keyword attribute groups (Table 2.4)

2.4.1 Keyword Attributes

The structure of a keyword attribute group is shown in Table 2.4. Structure parameters are defined in comdeck KEYGROUP.

Table 2.4: Structure of Keyword Attribute Group
(LL=LCKEY+MBAT+(N-1)*MKSIZ points to the word preceding the attribute group)

IQ(LL+MKF1)	format code: 4 integers ($4*16+2$)
IQ(LL+MKTYPE)	attribute type (MAD data type code, Table 2.8)
IQ(LL+MKDIM1)	first dimension of attribute
IQ(LL+MKDIM2)	second dimension of attribute
IQ(LL+MKDIM3)	third dimension of attribute
IQ(LL+MKF2)	format code: 1 name (MCNAM*16+5)
IQ(LL+MKNAME)	Attribute name (MCNAM words)
IQ(LL+MKSIZ)	Next keyword attribute group

2.5 Process and Subprocess Codes

The MAD commands are distinguished by two numeric codes, the process code and the subprocess code. The process code serves as a switch to select a program module, and the subprocess code selects the action to be performed by the program module. The two codes are defined for each command in the command dictionary. Table 2.5 summarizes the codes used at time of writing of this manual. Process code parameters are defined in comdeck PRCGROUP.

Table 2.5: Process and Subprocess Codes

keyword definitions					
process	subprocess	keyword	process	subprocess	keyword
MPKEY	1	KEYWORD	MPKEY	2	KEYEDIT
parameter and constant definitions					
process	subprocess	keyword	process	subprocess	keyword
MPPAR	1	CONSTANT	MPPAR	2	PARAMETER
element definitions					
process	subprocess	keyword	process	subprocess	keyword
MPELM	1	DRIFT	MPELM	2	SBEND
	3	RBEND		4	MATRIX
	5	QUADRUPOLE		6	SEXTUPOLE
	7	OCTUPOLE		8	MULTIPOLE
	9	SOLENOID		10	RFCAVITY
	11	ELSEPARATOR		12	SROTATION
	13	YROTATION		14	HKICKER
	15	KICKER		16	VKICKER
	17	HMONITOR		18	MONITOR
	19	VMONITOR		20	ECOLLIMATOR
	21	RCOLLIMATOR		22	BEAMBEAM
	23	LUMP		24	INSTRUMENT
	25	MARKER			
beam line and list definitions					
process	subprocess	keyword	process	subprocess	keyword
MPLIN	1	LINE	MPLIN	2	SEQUENCE
	3	LIST			
subroutines					
process	subprocess	keyword	process	subprocess	keyword
MPSUB	1	DO	MPSUB	2	ENDDO
	3	STORE		4	ENDSTORE
	5	SUBROUTINE		6	ENDSUBROUTINE
	7	CALLSUBROUTINE			

(continued on next page)

Table 2.5: Process and Subprocess Codes (continued)

executable commands					
process	subprocess	keyword	process	subprocess	keyword
MPSRV	1	PACKMEMORY	MPSRV	2	OPTION
	3	STOP		4	SET
	5	VALUE			
MPFIL	1	ARCHIVE	MPFIL	2	RETRIEVE
	3	ASSIGN		4	CALL
	5	RETURN		6	EXCITE
	7	INCREMENT		8	POOLDUMP
	9	POOLLOAD		10	SAVE
	11	STATUS		12	SYSTEM
	13	HELP		14	SHOW
MPENV	1	BEAM	MPENV	2	BETAO
	3	PRINT		4	SAVEBETA
	5	TITLE		6	USE
	7	SELECT		8	SPLIT
MPPLT	1	PLOT	MPPLT	2	SETPLOT
MPSUR	1	SURVEY			
MPTWS	1	TWISS	MPTWS	2	OPTICS
	3	BMPM		4	IBS
MPMAT	1	MATCH	MPMAT	2	CELL
	3	ENDMATCH		4	MIGRAD
	5	SIMPLEX		6	CONSTRAINT
	7	COUPLE		8	FIX
	9	LEVEL		10	VARY
	11	WEIGHT		12	LMDIF
	13	RMATRIX		14	TMATRIX
MPTRK	1	TRACK	MPTRK	2	ENDTRACK
	3	RUN		4	CONTINUE
	5	START		6	NOISE
MPHAR	1	HARMON	MPHAR	2	ENDHARMON
	3	HRESONANCE		4	HCHROMATICITY
	5	HFUNCTIONS		6	HTUNE
	7	HVARY		8	HCELL
	9	HWEIGHT		10	HLEVEL
MPERR	1	EALIGN	MPERR	2	EFIELD
	3	EOPTION		4	EPRINT
	5	ESAVE		6	EFCOMP
MPCOR	1	CORRECT	MPCOR	2	GETORBIT
	3	PUTORBIT		4	GETKICK
	5	PUTKICK		6	MICADO
MPLIE	1	DYNAMIC	MPLIE	2	STATIC
	3	NORMAL			
MPEDI	1	SEQEDIT	MPEDI	2	INSTALL
	3	MOVE		4	REMOVE
	5	ENDEDIT			
MPPOL	reserved				

2.6 Command Banks

For each command or definition read, MAD creates a command bank and links it to the corresponding keyword. Command have the structure shown in Table 2.6. Structure parameters are defined in comdeck BANKHEAD; MCSIZ is defined in comdeck CMDGROUP.

Table 2.6: Structure of Command Bank
(LCCMD is the address of the command bank)

LQ(LCCMD-NCAT) ... LQ(LCCMD-1)	bank for NCATth attribute, if required ... bank for 1st attribute, if required
...	central part of the bank
IQ(LCCMD+MBFRM) IQ(LCCMD+MBNAM) IQ(LCCMD+MBLN) IQ(LCCMD+MBPR) IQ(LCCMD+MBSP) IQ(LCCMD+MBAT)	format code: 4 integers ($4*16+2$) directory index for name in data object directory (Section 2.9) input line number of last definition process code (Section 2.5) subprocess code (Section 2.5) number of attributes NCAT, counting each component of an array individually
IQ(LCCMD+MBAT+1) ... IQ(LCCMD+MBAT+NCAT*MCSIZ)	} Attribute groups (Table 2.7)

2.6.1 Command Attributes

For each attribute of a command an *attribute data group* is allocated in the command bank with the structure shown in Table 2.7. Structure parameters are defined in comdeck CMDGROUP.

Table 2.7: Structure of Command Attribute Group
(LL = LCCMD+MBAT+(N-1)*MCSIZ points to the word preceding the attribute group)

IQ(LL+MCF1)	format code: 1 integer ($1*16+2$)
IQ(LL+MCTYP)	data type code: Ten times the MAD data type (Table 2.8) plus a flag value
IQ(LL+MCF2)	format code: MWNAM words (MWNAM*16+ZEBRA data type)
IQ(LL+MCVAL)	data value or empty, depending on data type

MAD defines its own data types listed in Table 2.8. Note that the data type codes are different from the ZEBRA type codes.

The space reserved in the value part of the attribute group is sufficient to store attribute values of most types. Some other data types require an additional bank for their value. For each attribute data group there is a corresponding *attribute link*, designed to support such a bank. More details about how attributes are stored can be found by consulting subroutine DCATTR and its subordinates.

2.6.2 Name Attributes

A name attribute value occupies all MWNAM words of the value part as a Hollerith string.

Table 2.8: MAD Data Types

data type code			meaning	default value
dictionary	keyword bank	command bank		
N	1	10	unspecified name	blank
		11	known name	
I	2	20	unspecified integer value	0
		21	integer constant	
R	3	30	unspecified real value	0.0
		31	real constant	
		32	real expression	
D	4	30	unspecified real value	0.0
		31	real constant	
		32	real expression	
		33	deferred expression	
L	5	50	unspecified logical value	false
		51	known logical value	
S	6	60	unspecified string	unspecified
		61	known string	
B	7	70	unspecified beam line	unspecified
		71	beam line name	
		72	beam line name with arguments	
		73	beam line list	
P	8	80	unspecified range reference	unspecified
		81	known range reference	
C	9	90	unspecified constraint	unspecified
		91	known constraint	
V	10	100	unspecified variable reference	unspecified
		101	known variable reference	

2.6.3 Integer Attributes

An integer attribute is stored in the first word of the value part.

2.6.4 Real Attributes or Deferred Expressions

A *real attribute* or the value of a *deferred expression* is stored in the first one or two words of the value part. If the value is defined as an expression (MAD data type code 32 or 33), the expression is encoded in *postfix notation*, and each operation gives rise to one *expression group* whose structure is shown in Table 2.9. The encoded expression is stored in an *expression bank* supported by the corresponding attribute link. The length of each expression group is `MXSIZ`. Structure parameters are defined in comdeck `EXPGROUP`. For each expression group the expression bank has a corresponding structural link which may support a *variable reference bank* .

Before a command is executed, expressions of type 32 are evaluated. Expressions of type 33 must be evaluated before using them by a call to `EXEVAL`. A call to `UTGFLT` will also ensure that a new value is available.

2.6.5 Logical Attributes

A logical attribute is encoded as an integer stored in the first word of the value part. Zero represents false, one represents true.

Table 2.9: Structure of Expression Bank
(LCEXP is the expression bank address,
LL = LCEXP+(N-1)*MXSIZ points to the word preceding the expression group)

LQ(LCEXP-IQ(LCEXP-2)) ... LQ(LCEXP-1)	} Variable reference banks (Section 2.6.10)
...	central part of the bank
IQ(LL+MXF1) IQ(LL+MXOP) IQ(LL+MXF2) IQ(LL+MXVAL)	format code: 1 integer ($1 * 16 + 2$) operation code (Table 2.10) format code: 1 vale (1*16+MREAL) constant value, if needed

Table 2.10: Operation codes in Expression Bank

-4	Load position from beam line sequence. The bank pointer is link 1 of a *variable reference bank* supported by the expression bank (section 2.6.10). The bias in the bank is stored in that same bank.
-2	Load attribute or parameter. The bank pointer is link 1 of a *variable reference bank* supported by the expression bank (Section 2.6.10). The attribute number to be loaded is stored in that same bank.
-1	Load constant value. The value is taken from IQ(LX+LXVAL).
0	Store result. The bank is given by the up link at LQ(LCEXP+1), and the bias in the bank is stored in IQ(LX+MXVAL).
1	binary plus (X + Y)
2	binary minus (X - Y)
3	multiply (X * Y)
4	divide (X / Y)
5	unary plus (+ X)
6	unary minus (- X)
7	square root (SQRT(X))
8	logarithm (LOG(X))
9	exponential (EXP(X))
10	sine (SIN(X))
11	cosine (COS(X))
12	absolute value (ABS(X))
13	tangent (TAN(X))
14	arc sine (ASIN(X))
15	maximum of two values (MAX(X,Y))
16	minimum of two values (MIN(X,Y))
17	uniform distribution in [0..1] (RANF())
18	gaussian distribution with $\sigma = 1$ (GAUSS())
19	user-defined random generator (USER0())
20	truncated gaussian (TGAUSS(X))
21	user-defined random generator (USER1(X))
21	user-defined random generator (USER2(X,Y))

2.6.6 String Attributes

The number of characters is stored in the first word of the value part. The actual string is stored in a bank supported by the corresponding attribute link.

2.6.7 Beam Line References

If a beam line is referred to by name, the directory index (relative to the data object directory) for its name is stored in the first word of the value part. If there is an actual argument list, it is encoded like a beam line definition, and its bank is supported by the corresponding attribute link.

If a beam line reference has the form of a literal list, the list is encoded as a beam line and referred to like a named line. The format of a *beam line reference* bank is the same as for a beam line bank (Section 2.7).

2.6.8 Range References

A range reference requires an extra *range reference bank* with six integer data words linked to the attribute link. The meaning of the six integers is summarized in Table 2.11.

Table 2.11: Structure of Range Reference Bank
(LCATTR is the address of the range reference bank)

bias	meaning
IQ(LCATTR+1)	ICODE1, code for beginning of range: 1: beginning of line (#S) 2: end of line (#E) 3: numeric index of the form #index1 4: name, optionally with occurrence: (name1 or name1[index1]) 5: selected elements only (name1[index1/index2])
IQ(LCATTR+2)	Value of index1 for ICODE1 ≥ 3
IQ(LCATTR+3)	Directory index for name for ICODE1 ≥ 4
IQ(LCATTR+4)	ICODE2, code for end of range: 1: beginning of line (#S) 2: end of line (#E) 3: numeric index of the form #index2 4: name, optionally with occurrence: (name or name[index]) 5: selected elements only (name[index1/index2])
IQ(LCATTR+5)	value of index2 for ICODE2 ≥ 3
IQ(LCATTR+6)	directory index for name2 for ICODE2 ≥ 4

2.6.9 Constraints

A constraint requires an extra *constraint bank* whose structure is show in Table 2.12. The two real attribute data groups have the same format as a real value. If they contain expressions, the expression banks are linked to the links 1 or 2 respectively of the constraint bank.

Table 2.12: Structure of Constraint Bank

...	central part of the bank
IQ(LCATTR+1)	code for the type of constraint: 1: minimum specified (> value) 2: maximum specified (< value) 3: minimum and maximum specified 4: equality specified (= value)
IQ(LCATTR+2) ... IQ(LCATTR+MCSIZ+1)	} real attribute data group for minimum or value
IQ(LCATTR+MCSIZ+2) ... IQ(LCATTR+2*MCSIZ+1)	} real attribute data group for maximum

2.6.10 Variable References

A *variable reference* requires an extra *variable reference bank*. Its structure is shown in Table 2.13. Structure parameters are defined in comdeck VARGROUP.

Table 2.13: Structure of Variable Reference Bank
(LCVAR is the address of the variable reference bank)

LQ(LCVAR-1)	reference link, points to bank referred to
...	central part of the bank
IQ(LCVAR+MVF1)	format code: 2 names (2*MWNAM*16+5)
IQ(LCVAR+MVBANK)	name of bank in which attribute resides
IQ(LCVAR+MVATTR)	name of attribute referred to
IQ(LCVAR+MVF2)	format code: 5 integers (5*16+2)
IQ(LCVAR+MVSEEN)	type of reference: 1=parameter, 2=attribute
IQ(LCVAR+MVIND1)	first index
IQ(LCVAR+MVIND2)	second index } for dimensioned attributes
IQ(LCVAR+MVIND3)	third index
IQ(LCVAR+MVBIAS)	bias of attribute in its bank

2.7 Beam Line Banks

Beam lines and replacement lists are stored as doubly linked lists. The structure of a beam line bank is shown in Table 2.14.

The structure of the list cells is represented in Table 2.15. The pointers "last", "first", "next", "previous", and "sublist head" contain the bias in the beam line bank of the list cell referred to. This avoids to declare them as links, and to give ZEBRA the burden of updating them when banks are moved. The field "dir. index" refers to a position in the *data object directory*. The field "repeat" contains the repeat count, as defined in the beam line or replacement list definition (one, if not specified).

The stack pointers and the stack area in list cells are used during expansion of a beam line by subroutine LNXPND. They are of no concern to the program user.

Table 2.14: Structure of Beam Line List
(LCCMD is the address of the beam line list bank)

LQ(LCCMD-3) LQ(LCCMD-2)	} stack pointers used during expansion of the beam line
LQ(LCCMD-1)	formal parameter bank (MWNAM words per name)
...	central part of the bank
IQ(LCCMD+MBFRM)	format code: 4 integers ($4 * 16 + 2$)
IQ(LCCMD+MBNAM)	directory index for beam line name
IQ(LCCMD+MBPR)	process code MPLIN (Section 2.5)
IQ(LCCMD+MBSP)	subprocess code (Section 2.5)
IQ(LCCMD+MBAT)	number of attributes NCAT = 0
IQ(LCCMD+MLFM)	format code: Rest of bank is integer (2)
IQ(LCCMD+MLHD)	offset in this bank of list header: IQ(LCCMD+IQ(LCCMD+MLHD))
IQ(LCCMD+MLF1)	LINE: bias of first dummy list for formals: first formal begins at IQ(LCCMD+IQ(LCCMD+MLF1)) LIST: stack area used during line expansion
IQ(LCCMD+MLF2)	LINE: bias of last dummy list for formals: last formal begins at IQ(LCCMD+IQ(LCCMD+MLF2)) LIST: stack area used during line expansion
IQ(LCCMD+MLF2+1) ... IQ(LCCMD+IQ(LCCMD-1))	} list cells (6 words each, Table 2.15)

Table 2.15: Beam Line List Cells

cell usage	cell contents					
	MLTYP	MLPRV	MLNXT	MLREF	MLREP	MLACT
Header of main list, or header of actual arguments list.	1	last	first	stack area		
Header of a sublist or of a dummy list replaced by name	2	last	first	stack area		
Header of a dummy list replaced by a sublist.	3	last	first	stack area		
Cell refers to a sublist of the same beam line.	4	previous	next	sublist head	repeat	unused
Cell refers to a sublist of another beam line, that is to an actual argument which is a sublist.	5	previous	next	sublist head	repeat	unused
Cell refers to a name without actual arguments.	6	previous	next	dir. index	repeat	unused
Cell refers to a name with actual arguments.	7	previous	next	dir. index	repeat	actual

2.8 Beam Sequences

For space reasons, a SEQUENCE has a somewhat special structure. It consists of two banks. The first bank contains the positions and has the structure shown in Table 2.16. The second bank contains the directory indices and has the structure shown in Table 2.17.

Walking through the members of the sequence implies a loop for $I = 2 \ldots N$. The directory index for the member is then found at IQ(L2+I), while the position value is stored in position IQ(L1+MBAT+(I-2)*MWFLT+2). If there is an expression bank for the position, it is linked to link LQ(L1-I).

2.9 Directories

2.9.1 Directory Structures

MAD uses two directories to keep track of names. Each directory consists of four banks, supported by four links in the root bank. The two directories serve the following purposes:

keyword directory: Holds all keyword definitions. The four banks of the keyword directory are supported by the links MDKEY through MDKEY+3 in the root bank. For accessing the keyword directory an array of reference links LDKEY is provided in common REFER (Section 3.1).

data object directory: Holds all data object definitions. The four banks for the data object directory are supported by the links MDBNK through MDBNK+3 in the root bank. For accessing the keyword directory an array of reference links LDBNK is provided in common REFER (Section 3.1).

The two arrays LDKEY and LDBNK may be thought of as *directory handles* for easy access to the two directories. The four banks of a directory have the following functions:

index bank: This bank has no links. Its data part contains an ordering vector (one word per entry) permitting access to names in alphabetical order and binary search.

name bank: This bank has no links. Its data part contains a 16-character object name per entry. This requires two or four words per entry for the single precision and double precision version of MAD respectively.

bank pointers: This bank has one reference link per entry, pointing to the data structure containing the defined object. If an object has been referred to, but not defined, the corresponding link is zero. The single data word contains the number of used entries.

occurrence count bank: This bank has no links. The data part has one word per entry used during beam line expansion to keep track of occurrence counts for sublines and beam elements.

The structure of a directory can be depicted as in Figure 2.2.

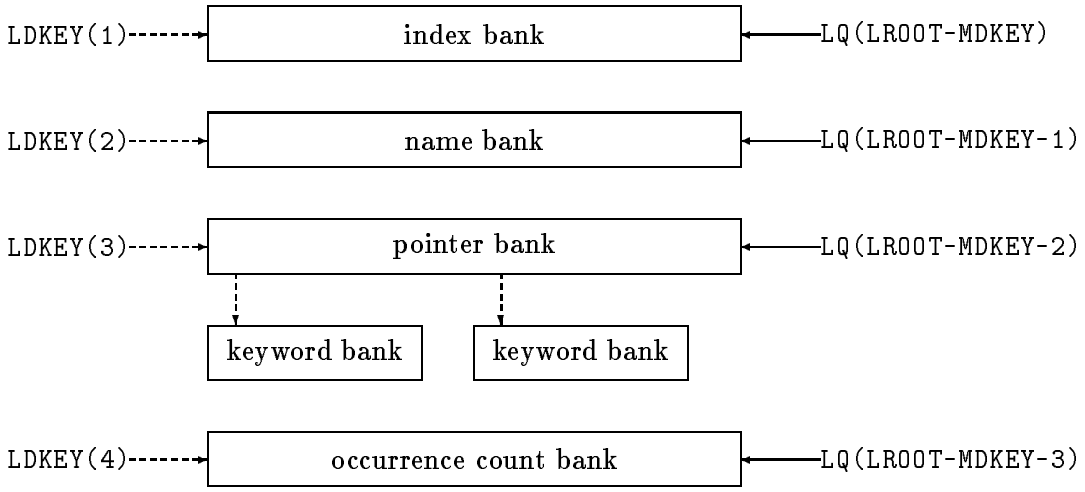

Figure 2.2: Structure of Keyword Directory

When MAD sees a new name, it creates a new *directory entry* . It assigns it a fixed *directory index*, namely the next free position in the name, pointer, and occurrence count bank of the directory. All later references to the same name use this index. This makes later redefinition easy by simply replacing the bank pointer. The index bank is updated to permit addressing the name bank in alphabetical order.

2.10 Beam Line Expansions

For each reference to a beam line or sequence, MAD creates a tree describing the beam line's expansion. If the reference occurs in a `USE` command, its topmost bank is supported by link `MCSEQ` of the root bank. For faster access this link is duplicated in `LCSEQ`, stored in common block `REFER`. When a `LINE` attribute occurs in a `MATCH` or `TWISS` command, this beam line expansion is only required as long as the program module is active. Its supporting link becomes local to a link area of the relevant program module.

The structure of a beam line expansion is shown in Table 2.19 and Figure 2.3. Structure parameters are defined in comdeck `SEQGROUP`. The various banks all contain an entry for each *position* in the machine, i. e. for each physical element and for the entrance and exit of each beam line.

2.10.1 Flag Words

The format of the flag words is given in Table 2.18, and its parameters are defined in comdeck `SEQFLAG`. Packing and unpacking may be done using the CERN program library routines `SBITO`, `SBIT1`, `SBYT`, `JBIT` and `JBYT` (Chapter 34).

Table 2.16: Structure of the First Bank of a Beam Line Sequence
(L1 is the address of the first sequence bank)

LQ(L1-NEL1EM) ... LQ(L1-2)	} pointers to any expression banks for positions
LQ(L1-1)	pointer to the second bank for the SEQUENCE
...	central part of the bank
IQ(L1+MBFRM)	format code: 4 integers ($4 * 16 + 2$)
IQ(L1+MBNAM)	directory index for sequence name
IQ(L1+MBPR)	process code MPLIN (Section 2.5)
IQ(L1+MBSP)	subprocess code (Section 2.5)
IQ(L1+MBAT)	number of attributes NCAT = 1
IQ(L1+MBAT+1) ... IQ(L1+MBAT+MCSIZ)	} attribute group for REFER
IQ(L1+MBAT+MCSIZ+1)	format code: rest of bank is real
IQ(L1+MBAT+MCSIZ+2) ... IQ(L1+MBAT+MCSIZ+N*MWFLT+1)	} Positions ($N - 1$ real values)
...	free space

Table 2.17: Structure of the Second Bank of a Beam Line Sequence
(L2 is the address of the second sequence bank)

...	central part of the bank
IQ(L2+1)	Number of members plus 1
IQ(L2+2) ... IQ(L2+N)	} $N - 1$ directory indices
...	free space

Table 2.18: Flag Words in Beam Line Expansions

bit position(s)	meaning
1...MCODE	1: exit of a physical element 2: beginning of a beam line 3: end of a beam line
MFRST	dump flag for subroutine TMFRST
MLUMP	dump flag for subroutine LMLUMP
MREFE	dump flag for subroutine TMREFE
MSCND	dump flag for subroutine TMSCND
MOPTC	output flag for OPTICS
MPRNT	print flag for TWISS or SURVEY
MTRAK	print flag for TRACK
MOCC1...MOCC2	occurrence counter for the element or line in this position

Table 2.19: Structure of Beam Line Expansion
(LCSEQ is the address of the expansion for the main beam line)

LQ(LCSEQ-MSELM)	Reference link: list of quadrupoles and sextupoles affecting dispersion in the beam line (Section 2.10.3). In each bank reference link 1 points to the next item.
LQ(LCSEQ-MSMON)	Reference link: list of monitors in the beam line (Section 2.10.3). In each bank reference link 1 points to the next item.
LQ(LCSEQ-MSCOR)	Reference link: list of orbit correctors in the beam line (Section 2.10.3). In each bank reference link 1 points to the next item.
LQ(LCSEQ-MSLIE)	This link supports a linear structure with Lie-algebraic maps for the line, one bank for each map order used. See subroutine LMLUMP for the bank format.
LQ(LCSEQ-MSMAP)	This link supports a linear structure with TRANSPORT maps for the line, one bank for each value of $\delta p/p$ used. See subroutine TMTURN for the bank format.
LQ(LCSEQ-MSCOM)	The bank supported by this link holds the supporting links for the banks in the lists pointed at by the links MSELM, MSMON, MSMON. (Section 2.10.3).
LQ(LCSEQ-MSNUM)	The bank supported by this link contains the number of physical elements preceding the position.
LQ(LCSEQ-MSFLD)	This link supports a bank supporting the field error banks for all positions which have errors (Section 2.10.2).
LQ(LCSEQ-MSALI)	This link supports a bank supporting the alignment error banks for all positions which have errors (Section 2.10.2).
LQ(LCSEQ-MSFLG)	This link supports a bank containing a flag word for each position (Table 2.18).
LQ(LCSEQ-MSDIR)	This link supports a bank containing the directory indices (relative to the data object directory) for the element or beam line of each position.
...	central part of the bank
IQ(LCSEQ+MSF1)	format code: 4 integers ($4 * 16 + 2$)
IQ(LCSEQ+MSR1)	first position in range
IQ(LCSEQ+MSR2)	last position in range
IQ(LCSEQ+MSYM)	symmetry code
IQ(LCSEQ+MSUP)	number of superperiods
IQ(LCSEQ+MSF2)	format code: Hollerith follows ($0 * 16 + 5$)
IQ(LCSEQ+MSBN)	name of beam line (MCNAM words)
IQ(LCSEQ+MSRN)	encoded range name (MCRNG words)

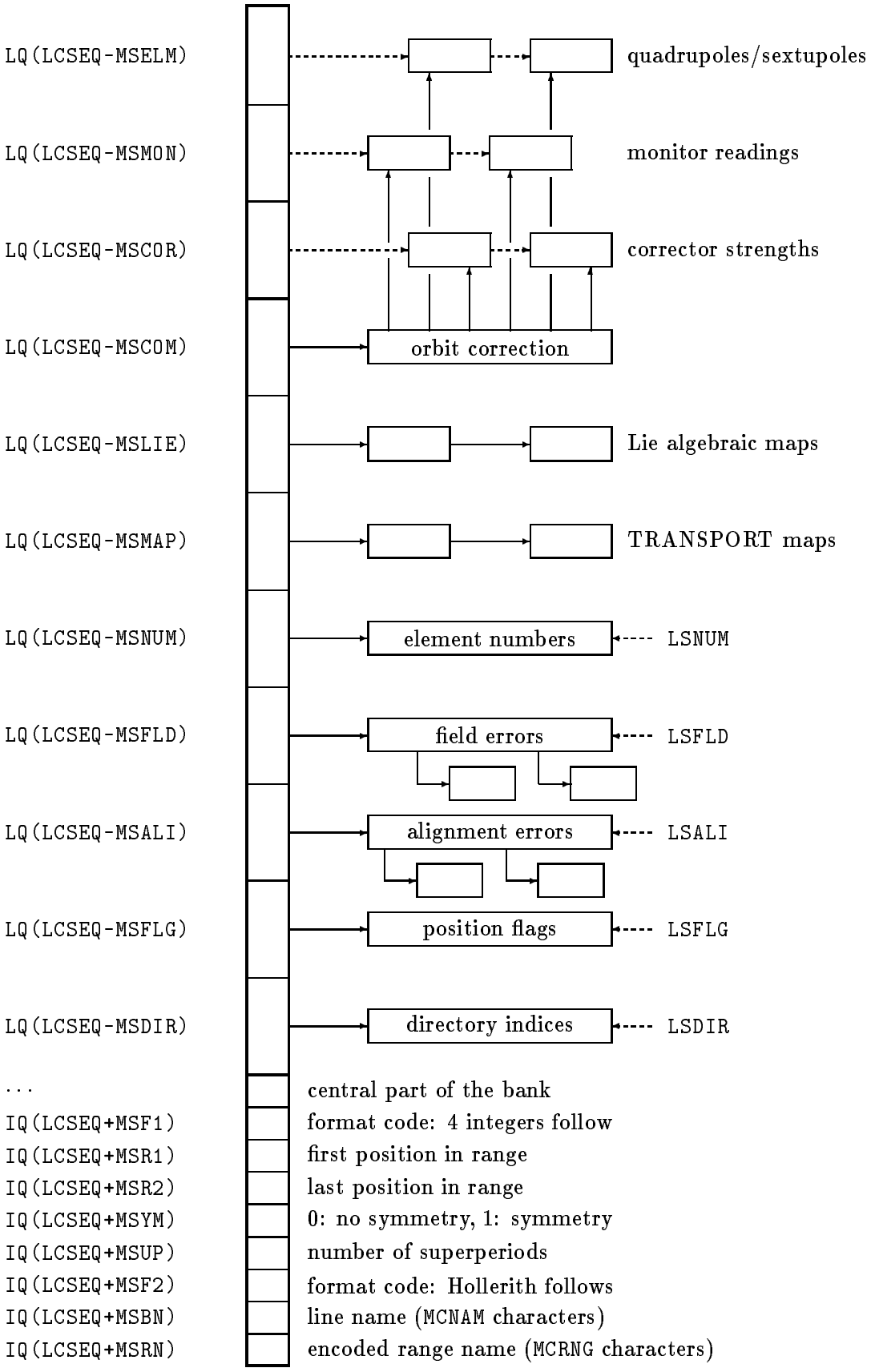

Figure 2.3: Structure of Beam Line Expansion

2.10.2 Machine Imperfections

If any misalignment errors exist in the working beam line, a pointer bank is linked to the link at LSALI). The misalignment errors for an element at position n are stored in a *misalignment bank* supported by link n in this pointer bank. The misalignment error bank contains six or ten real or double-precision values in the following order:

1. Horizontal displacement (DX),
2. Vertical displacement (DY),
3. Longitudinal displacement (DS),
4. Rotation angle about y-axis (DTHETA),
5. Rotation angle about x-axis (DPHI),
6. Rotation angle about s-axis (DPSI),
7. Horizontal orbit error (MREX, for monitors only),
8. Vertical orbit error (MREY, for monitors only).
9. Horizontal dispersion error (MREDX, for monitors only),
10. Vertical dispersion error (MREDY, for monitors only).

Whole beam lines may also be misaligned, and the errors are linked to their entrance. To facilitate handling the error banks are duplicated for the exit.

Field errors are stored as absolute errors, even when entered as relative values. The *field error bank* for an element in position n is linked to link n of the pointer bank at LSFLD. Lines or lumps may not have field errors. The field error banks contain pairs of real or double precision values in ascending multipole order, beginning at 0 with no gaps. The first value of each pair is the integrated normal multipole, and the second the integrated skew multipole. The following elements may have field errors:

Dipole: stores and uses dipole through octupole components.

Orbit Corrector: stores and uses dipole component only.

Quadrupole: stores dipole and quadrupole, uses quadrupole only.

Sextupole: stores dipole through sextupole, uses sextupole only.

Octupole: stores dipole through octupole, uses octupole only.

Thin Multipole: truncates at order MAXMUL, stores all orders from zero to the highest non-zero component, uses all stored error components.

2.10.3 Element, Corrector and Monitor Tables

The orbit correction module "CO" requires space to store lattice functions, kicks, quadrupole and sextupole strengths, and orbit readings. For this purpose it sets up three lists, pointed at by reference links LQ(LCSEQ-MSELM), LQ(LCSEQ-MSCOR), and LQ(LCSEQ-MSMON) respectively. The structural links to the banks in these lists are stored in a bank at LQ(LCSEQ-MSCOM), and the "next" links is stored in each bank in reference link 1. Walking through one of these structures enables reference to all correctors or all monitors in order of occurrence, e. g. for printing purposes.

Each bank in these lists contains the following real or double precision values:

1. Horizontal kick (corrector) or orbit reading (monitor),
2. Vertical kick (corrector) or monitor reading (monitor),
3. Horizontal dispersion reading (monitor),
4. Vertical dispersion reading (monitor),
5. Longitudinal position s,
6. Horizontal β_x,
7. Vertical β_y,
8. Horizontal phase μ_x,
9. Vertical phase μ_y,
10. Quadrupole strength (quadrupole) or sextupole strength times dispersion (sextupole).

The corrector and monitor banks are eacg marked with the six status bits listed in table 2.20

Table 2.20: Status bits in Corrector and Monitor Banks

	Correctors:
1	Corrector has horizontal action
2	Corrector has vertical action
3	Corrector has been set by MICADO for horizontal correction
4	Corrector has been set by MICADO for vertical correction
5	Corrector is activated for horizontal correction
6	Corrector is activated for vertical correction
	Monitors:
1	Monitor reads horizontal plane
2	Monitor reads vertical plane
3	Monitor is valid for horizontal orbit reading
4	Monitor is valid for vertical orbit reading
5	Monitor is valid for horizontal dispersion reading
6	Monitor is valid for vertical dispersion reading

2.11 Bank Status Bits

ZEBRA uses a the upper 14 bits of the status word of each bank for status information. The lower 18 bits are available for the user program. MAD uses eight of these bits to keep track of events. The definitions for these bits can be found in comdeck `MARKBITS`:

`MXDRP` Drop bit (Section 4.6),

`MXORD` Order bit (Section 12.7),

`MXDEF` Defer bit (Section 12.7),

`MXCLS` Class bit (Section 18),

`MXALS` Alias bit (Section 18),

`MXLMP` Recursion bit (Section 4.9),

`MXKNW` Known bit (Section 4.9),

`MXMOD` Modify bit (Section 4.9).

Chapter 3. Global Common Blocks

3.1 Common Block /REFER/

MAD keeps the most used ZEBRA reference links in a common block REFER:

```
      COMMON /REFER/    LREF1,
     +                  LCALI, LCATT, LCCLS, LCCMD, LCCOM, LCDEF, LCELM,
     +                  LCEXP, LCFLD, LCKEY, LCSEQ, LCSPL, LCSRC, LCVAR,
     +                  LDBNK(4), LDKEY(4),
     +                  LSALI, LSCOM, LSDIR, LSFLD, LSFLG, LSNUM, LSSPL,
     +                  LREF2
      SAVE              /REFER/
```

There are service routines to fill these reference links, and the ZEBRA system keeps them normally up to date.

LCALI	Misalignments for the current element (Section 2.10.2).
LCATT	Current attribute bank.
LCCLS	Current class bank.
LCCMD	Current command or element being decoded.
LCCOM	Current corrector or monitor bank.
LCDEF	Current default values bank.
LCELM	Current beam element during optics or tracking calculation.
LCEXP	Current expression bank.
LCFLD	Field errors for the current element (Section 2.10.2).
LCKEY	Keyword for current command.
LCSEQ	Current main beam line sequence.
LCSPL	Current split pointer.
LCSRC	Current source bank for copy or defaults.
LCVAR	Current variable reference bank.
LDBNK(4)	Data object directory (see Section 2.9).
LDKEY(4)	Keyword directory (see Section 2.9).
LSALI	Bank for misalignment pointer sequence.
LSCOM	Bank for corrector and monitor table.
LSDIR	Bank for directory index sequence.
LSFLD	Bank for field error pointer sequence.
LSFLG	Bank for position flag sequence.
LSNUM	Bank for occurrence numbers.
LSSPL	Bank holding split pointers.

3.2 Common block BEAM and BEAM bank

When MAD sees a BEAM command, it builds a special command bank with the name BEAM to hold the information read on the BEAM command. This command bank holds also any information about the circulating beam which has been computed by the program. For faster access the beam description is duplicated in comdeck BEAM:

```
      COMMON /BEANAM/   PRTNAM
      COMMON /BEAFLT/   AMASS, CHARGE, ENERGY, PC, GAMMA,
     +                  EX, EXN, EY, EYN, ET, SIGT, SIGE,
     +                  BUNCH, PARNUM, CURRNT, SIGX, SIGY,
     +                  FREQO, BETA, UO, ARAD, PDAMP(3)
      COMMON /BEAINT/   IETFLG, IPNFLG
      COMMON /BEALOG/   FBCH, FRAD
      SAVE              /BEANAM/, /BEAFLT/, /BEAINT/, /BEALOG/
      LOGICAL           FBCH, FRAD
      CHARACTER*(MCNAM) PRTNAM
```

Here are the data entered with the BEAM command:

PRTNAM — Name of the beam particles in character format,

AMASS — Mass of the beam particles in GeV/c^2,

CHARGE — Charge of the beam particles in elementary charges,

ENERGY — Nominal energy of a beam particle in GeV,

PC — Nominal momentum of a beam particle in GeV/c,

GAMMA — Relativistic parameter $\gamma = E/m_0$,

EX — Horizontal emittance (Courant-Snyder invariant) in m,

EXN — Normalized horizontal emittance $E_{xn} = 4\beta\gamma E_x$,

EY — Vertical emittance (Courant-Snyder invariant) in m,

EYN — Normalized vertical emittance $E_{yn} = 4\beta\gamma E_y$,

ET — Longitudinal emittance $E_t = c\sigma_t\sigma_E$,

SIGT — Bunch length $c\sigma_t$,

SIGE — Energy spread σ_E,

BUNCH — Number of bunches,

PARNUM — Number of particles per bunch,

CURRNT — Bunch current in A,

FBCH — If .TRUE., the beam is bunched.

FRAD — If .TRUE., synchrotron radiation is considered.

The following values are computed and stored by MAD:

FREQO — The revolution frequency in *MHz*.

SIGX — Horizontal beam size σ_x,

SIGY — Vertical beam size σ_y,

BETA — Relativistic parameter $\beta = v/c$,

U0 — Synchrotron radiation loss per turn in GeV,

ARAD — Classical particle radius (electron radius for electrons),

PDAMP(3) — Damping partition numbers.

IETFLG — Tells which value was given to specify the longitudinal emittance:

0: None given,

1: ET given,

2: SIGT given,

3: SIGE given.

IPNFLG — Tells which value was given to specify the number of particles:

0: None given,

1: BCURRENT given,

2: NPART given,

Before each command executed, MAD calls subroutine ENGET (Section 10.3) to copy the contents of the BEAM bank into this block.

To store something into the beam bank, one has to store the information in the BEAM common, and then to call ENPUT (Section 10.4) to copy the data to the BEAM bank.

3.3 Keyword Data

Simple access to the contents of a keyword definition is possible by using the comdeck KEYWORD:

```
PARAMETER (MAXAT = 100)
COMMON /KEYWDI/ IATYPE(MAXAT),IADIM1(MAXAT),IADIM2(MAXAT),IADIM3(MAXAT)
COMMON /KEYWDC/ KATNAM(MAXAT)
SAVE /KEYWDI/, /KEYWDC/
CHARACTER*(MCNAM) KATNAM
```

It can be loaded from a keyword bank by calling KWGET (Section 15.2). The variables used above have the following meaning:

MAXAT — A parameter defining the maximum number of arguments, counting arrays for one argument.

IATYPE(i) — The MAD data type for the i^{th} attribute.

IADIM1(i) — The first dimension for the i^{th} attribute.

IADIM2(i) — The second dimension for the i^{th} attribute.

IADIM3(i)

KATNAM(i) — The name of the i^{th} attribute.

3.4 TRANSPORT Map for Current Element

The subroutine TMMAP (Section 28.5) returns the TRANSPORT map for an element in the common block MAPELM:

```
      COMMON /MAPELM/   RE(6,6), TE(6,6,6)
      SAVE              /MAPELM/
```

with the arrays

RE — the first-order transfer matrix for the element.

TE — The second-order terms of the TRANSPORT map.

3.5 TRANSPORT Map for one Turn

TMREFE (Section 28.7), TMFRST (Section 28.3), and TMSCND (Section 28.9) return the linear transfer matrix or the TRANSPORT map for one turn in the common block MAPTRN:

```
      COMMON /MAPTRN/   RT(6,6), TT(6,6,6), RTP(6,6)
      SAVE              /MAPTRN/
```

with the arrays

RT — the first-order transfer matrix for one turn.

TT — The second-order terms of the TRANSPORT map for one turn (not for TMREFE).

RTP — Some routines of the Twiss module use this array to store the first derivative of the linear transfer matrix with respect to the relative energy error.

3.6 Lattice Functions for Beginning of Line

The initial values for the lattice functions are kept in the common block OPTICO:

```
       COMMON /OPTICO/   BETXO, ALFXO, AMUXO, BETYO, ALFYO, AMUYO,
     +                   ORBITO(6), DISPO(6),
     +                   WXO, PHIXO, DMUXO, WYO, PHIYO, DMUYO,
     +                   DDISPO(6), CIRC, ROMAT(2,2)
       SAVE              /OPTICO/
```

It contains the variables

BETXO — β_x or, for coupled machines, β_1.

ALFXO — α_x or, for coupled machines, α_1.

AMUXO — μ_x or, for coupled machines, μ_1.

BETYO — β_y or, for coupled machines, β_2.

ALFYO — α_y or, for coupled machines, α_2.

AMUYO — μ_y or, for coupled machines, μ_2.

ORBITO(6) — closed orbit position (x, p_x, y, p_y, ct, $\delta E/p_0$).

DISPO(6) dispersion vector, i.e. derivative of closed orbit with respect to the relative energy error.

WXO The chromatic amplitude W_x.

PHIXO The chromatic phase Φ_x.

DMUXO The derivative of μ_x with respect to the relative energy error.

WYO The chromatic amplitude W_y.

PHIYO The chromatic phase Φ_y.

DMUYO The derivative of μ_y with respect to the relative energy error.

DDISPO(6) The second derivative of the closed orbit vector with respect to the relative energy error.

CIRC The machine circumference.

ROMAT(2,2) The coupling matrix, defined as $R \tan \phi$ in the formalism by L. C. Teng [17].

3.7 Lattice Functions for Current Position

The values of the lattice functions are stored in common block OPTIC1, which is organized in the same way as OPTICO:

```
      COMMON /OPTIC1/   BETX, ALFX, AMUX, BETY, ALFY, AMUY,
     +                  ORBIT(6), DISP(6),
     +                  WX, PHIX, DMUX, WY, PHIY, DMUY,
     +                  DDISP(6), SUML, RMAT(2,2)
      SAVE              /OPTIC1/
```

It contains the variables

BETX β_x or, for coupled machines, β_1.

ALFX α_x or, for coupled machines, α_1.

AMUX μ_x or, for coupled machines, μ_1.

BETY β_y or, for coupled machines, β_2.

ALFY α_y or, for coupled machines, α_2.

AMUY μ_y or, for coupled machines, μ_2.

ORBIT(6) closed orbit position (x, p_x, y, p_y, ct, $\delta E/p_0$).

DISP(6) dispersion vector, i.e. derivative of closed orbit with respect to the relative energy error.

WX The chromatic amplitude W_x.

PHIX The chromatic phase Φ_x.

DMUX The derivative of μ_x with respect to the relative energy error.

WY The chromatic amplitude W_y.

PHIY The chromatic phase Φ_y.

DMUY The derivative of μ_y with respect to the relative energy error.

DDISP(6) The second derivative of the closed orbit vector with respect to the relative energy error.

SUML The length accumulated at the current position.

RMAT(2,2) The coupling matrix, defined as $R \tan \phi$ in the formalism by L. C. Teng [17].

3.8 Option Flags

Execution of MAD is controlled by flags which may be set by the user. These flags are set by the OPTION command and reside in the common block OPTION:

```
      COMMON /OPTFLT/   OPTFLT(10)
      EQUIVALENCE       (COFACT, OPTFLT( 1))
      COMMON /OPTINT/   OPTINT(10)
      INTEGER           OPTINT
      INTEGER           ICMDFL, IDEFFL, IEXPFL, IKEYFL, ILINFL
      EQUIVALENCE       (ICMDFL, OPTINT(1)), (IDEFFL, OPTINT(2))
      EQUIVALENCE       (IEXPFL, OPTINT(3)), (IKEYFL, OPTINT(4))
      EQUIVALENCE       (ILINFL, OPTINT(5))
      COMMON /OPTLOG/   OPTFLG(20), OPTCON(5)
      LOGICAL           OPTFLG, OPTCON
      LOGICAL           DEBUG, DOUBLE, ECHO, INTER, TRACE, VERIFY,
     +                  WARN, INFO, SYMPL
      LOGICAL           RESET, TELL
      EQUIVALENCE       (DEBUG, OPTFLG(1)), (DOUBLE, OPTFLG(2))
      EQUIVALENCE       (ECHO, OPTFLG(3)), (INTER, OPTFLG(4))
      EQUIVALENCE       (TRACE, OPTFLG(5)), (VERIFY, OPTFLG(6))
      EQUIVALENCE       (WARN, OPTFLG(7)), (INFO, OPTFLG(8))
      EQUIVALENCE       (SYMPL, OPTFLG(9))
      EQUIVALENCE       (RESET, OPTCON(1)), (TELL, OPTCON(2))
```

COFACT A real value used to limit the corrections at each iteration in closed orbit search.

ICMDFL Dump all new commands.

IDEFFL Dump all new elements and parameters.

IEXPFL Dump all new expressions, and evaluated ones.

IKEYFL Dump all new keywords.

ILINFL Dump all new lines.

DEBUG True implies printing of messages for marking and dropping modified banks.

DOUBLE If true, TFS tables are created in double precision.

ECHO True, to print input echo.

INTER True, if MAD runs in interactive mode.

TRACE — True, to print messages for command begin and end. The message includes information about timing.

VERIFY — True implies VERIFY option.

WARN — If true, warning messages are printed.

INFO — If true, information messages are printed.

SYMPL — True forces symplectification of transfer matrices.

RESET — True implies rest of all options to their defaults in the (OPTION command).

TELL — True implies printing of all options in the OPTION command.

The values of the 5 integer flags mean:

0: No dump.

1: Dump in "readable" format.

2: Dump in ZEBRA format.

3: Dump in both formats.

3.9 Physical Constants

Many useful physical constants are available in comdeck PHYSICPM: CLIGHT AMUO EPSO HBAR QELECT FALFA EMASS ERAD ELAMDA ASUBE PMASS PRAD PLAMDA ASUBP

```
*---- Universal physical constants.
*     Velocity of light [m/s]:
      PARAMETER (CLIGHT = 2.997 924 58 D+08)
*     Permeability of vacuum [V*s/A*m]:
      PARAMETER (AMUO = 1.256 637 061D-06)
*     Permittivity of vacuum [A*S/V*m]:
      PARAMETER (EPSO = 8.854 187 817D-12)
*     Reduced Planck's constant [GeV*s]:
      PARAMETER (HBAR = 6.582 122 0 D-25)

*---- Electromagnetic constants.
*     Elementary charge [A*s]:
      PARAMETER (QELECT = 1.602 177 33 D-19)
*     Fine structure constant [1]:
      PARAMETER (FALFA = 7.297 353 08 D-03)

*---- Electron.
*     Rest mass [GeV]:
      PARAMETER (EMASS = 0.510 999 06 D-03)
*     Classical radius [m]:
      PARAMETER (ERAD = 2.817 940 92 D-15)
*     Reduced Compton wavelength [m]:
      PARAMETER (ELAMDA = 3.861 593 23 D-13)
*     Magnetic moment anomaly [1]:
```

```
      PARAMETER (ASUBE = 1.159 652 193D-03)

*---- Proton.
*     Rest mass [GeV]:
      PARAMETER (PMASS = 0.938 272 31 D+00)
*     Classical radius [m]:
      PARAMETER (PRAD = 1.534 698 57 D-18)
*     Reduced Compton wavelength [m]:
      PARAMETER (PLAMDA = 2.103 089 37 D-16)
*     Magnetic moment anomaly [1]:
      PARAMETER (ASUBP = 1.792 847 386D+00)
```

The value of π is available in comdeck PI:

```
PARAMETER (PI = 3.1415926535898D0)
```

3.10 Data for Main Working Beam Line

Most program modules require a working beam line to be set. For simple access they may copy the most important data for this line into a set of common blocks by calling UTBEAM (Section 33.1). The comdeck RANGE defines these common blocks as:

```
      COMMON /RNGCHR/ LINNAM, RNGNAM
      CHARACTER LINNAM*(MCNAM), RNGNAM*(MCRNG)
      COMMON /RNGINT/ IRG1, IRG2, NSUP
      COMMON /RNGLOG/ SYMM
      LOGICAL SYMM
      SAVE /RNGCHR/, /RNGINT/, /RNGLOG/
```

with the variables:

LINNAM The name of the main working beam line.

RNGNAM A character string encoding the current range set by a USE command.

IRG1 The index into the working beam line for the beginning of the range.

IRG2 The index into the working beam line for the end of the range.

NSUP Number of superperiods.

SYMM The symmetry flag.

3.11 Status Flags

The status flags are set by the program in response to certain situations to help in deciding how to deal with computations. They are held in common block STATUS:

```
      COMMON /STATUS/   ERROR, SCAN, NWARN, NFAIL, IMODUL, IPLFLG,
     +                  INVAL, MAYCPL, STABX, STABY, STABT,
     +                  NEWCOR, NEWMAP, PROMPT
      SAVE              /STATUS/
      LOGICAL           ERROR, SCAN,
     +                  INVAL, MAYCPL, STABX, STABY, STABT,
     +                  NEWCOR, NEWMAP, PROMPT
```

ERROR Fatal error in current command.

SCAN Scanning mode.

NWARN Warning message count.

NFAIL Fatal error message count.

IMODUL Process code for the current module (for HARMON, MATCH, TRACK, SEQEDIT, etc.).

IPLFLG Zero, if no plot has been generated, set to one by first plot.

INVAL Optics computed with initial values.

MAYCPL Coupling effects found.

STABX First mode is stable (usually horizontal).

STABY Second mode is stable (usually vertical).

STABT Third mode is stable (usually longitudinal).

NEWCOR Optical functions in table of correctors and monitors should be recomputed.

NEWMAP Precomputed maps should be dropped.

PROMPT Print prompt ''M: ==>'' before reading an input line.

A second common block contains flags which control program actions. Several commands in MAD are not able to deal with all situations. The control flags are set according to the data before each command; those command routines may change the settings within certain limits.

```
PARAMETER          (MAXCPF = 10, MAXDOF = 10)
COMMON /STFLAG/    CPFLAG(MAXCPF), DOFLAG(MAXDOF)
LOGICAL            CPFLAG, CPLXY, CPLXT
LOGICAL            DOFLAG, DOCAV, DORAD, DOALI, DOFLD, DOKICK
SAVE               /STFLAG/
EQUIVALENCE        (CPLXY, CPFLAG(1)), (CPLXT, CPFLAG(2))
EQUIVALENCE        (DOCAV, DOFLAG(1)), (DORAD, DOFLAG(2))
EQUIVALENCE        (DOALI, DOFLAG(3)), (DOFLD, DOFLAG(4))
EQUIVALENCE        (DOKICK, DOFLAG(5))
```

The meaning of the flags is:

CPLXY If true, transverse coupling has been detected.

CPLXT If true, synchro-betatron coupling has been detected.

DOCAV If true, cavities are considered.

DORAD If true, synchrotron radiation is considered.

DOALI If true, alignment errors are considered.

DOFLD If true, field errors are considered.

DOKICK If true, the orbit corrector table is considered.

The flags whose names begin with CP are set by various optics routines. They should be reset to false before running through the machine.

The flags whose names begin with DO are set to true before each command (except DORAD which is set from the BEAM command. If any of the effects is not desired, a program module should reset the flag to false, do its computations, and then reset it to the value it had upon entry.

3.12 Summary Data for Optics

The global results of an optics calculation are kept in common block SUMMRY:

```
      COMMON /SUMMRY/   QX, XIX, COSMUX, SINMUX, BXMAX, DXMAX, XCOMAX,
     +                  SIGXCO, SIGDX,
     +                  QY, XIY, COSMUY, SINMUY, BYMAX, DYMAX, YCOMAX,
     +                  SIGYCO, SIGDY,
     +                  QS, ALFA, GAMTR, RBAR, DELTA
      SAVE              /SUMMRY/
```

It contains the variables:

QX	Horizontal tune Q_x.
XIX	Horizontal chromaticity $\xi_x = dQ_x/d\delta$.
COSMUX	$\cos(2\pi\mu_x)$
SINMUX	$\sin(2\pi\mu_x)$
BXMAX	Maximum value of β_x.
DXMAX	Maximum value of horizontal dispersion D_x.
XCOMAX	Maximum horizontal orbit deviation.
SIGXCO	Horizontal standard deviation for orbit.
SIGDX	Horizontal standard deviation for dispersion.
QY	Vertical tune Q_y.
XIY	Vertical chromaticity $\xi_y = dQ_y/d\delta$.
COSMUY	$\cos(2\pi\mu_y)$
SINMUY	$\sin(2\pi\mu_y)$
BYMAX	Maximum value of β_y.
DYMAX	Maximum value of vertical dispersion D_y.
YCOMAX	Maximum vertical orbit deviation.
SIGYCO	Vertical standard deviation for orbit.
SIGDY	Vertical standard deviation for dispersion.
QS	Synchrotron tune Q_s.
ALFA	Momentum compaction.

GAMTR — Transition energy factor $\gamma_{tr} = E_{tr}/m_0$.

RBAR — Average machine radius.

DELTA — Relative energy error (global average in the presence of cavities).

3.13 Logical Unit Numbers for Input and Output

The logical unit numbers for the standard input/output files are kept in common block ZUNIT. Most of them are set by the ZEBRA system:

```
      COMMON /ZUNIT/    IQREAD, IQPRNT, IQPR2, IQLOG, IQPNCH,
     +                  IQTTIN, IQTYPE
      SAVE              /ZUNIT/
```

The unit numbers have the following meaning:

IQREAD — Logical unit number for current input file.

IQPRNT — Logical unit number for ZEBRA print file. Since ZEBRA uses this unit for some messages, it is made the same as for the ECHO file.

IQPR2 — Logical unit number for PRINT print file (receives all listings).

IQLOG — Logical unit number for ECHO file (receives all messages).

IQPNCH — Logical unit for PUNCH file (not used).

IQTTIN — Logical unit for terminal input.

IQTYPE — Logical unit for terminal output.

Part II

MAD Subroutines and Functions

Chapter 4. Main Module "AA"

The AA module is the main executive of MAD. It also contains a few general-purpose utilities.

Table 4.1: Routines in the AA Module

Name	Purpose	Section
AAATTR	Decode attributes for a statement	4.4
AABOOK	Book new statement bank	4.3
AACMND	Decode executable command	-
AACOPY	Copy attributes between banks	4.5
AADROP	Drop statement bank	4.6
AADUMP	Dump command or element definition	4.7
AAELEM	Decode element definition	-
AAEXEC	Execute command	4.2
AAFAIL	Print fatal error message	4.8
AAINFO	Print fatal error message	4.8
AAINIT	Initialize MAD data structure	-
AAMAIN	Main loop routine for commands and definitions	4.1
AAMARK	Mark bank as modified	4.9
AAOPTS	Command routine for OPTION	-
AAPARA	Command routine for PARAMETER	-
AAPDRP	Delete precomputed maps after change in BEAM common	4.9
AAPMOD	Delete precomputed maps after parameter change	4.9
AAPROC	Build procedure list for SUBROUTINE or DO	-
AAREAD	Read complete command	-
AARUNS	Execute SUBROUTINE or DO	-
AASERV	Switch routine for process code MPSRV (services)	-
AASET	Command routine for SET command	-
AASMOD	Check validity for redefinitions	4.10
AASUBR	Switch routine for process code MPSUB (subroutines)	-
AAVALU	Command routine for VALUE command	-
AAWARN	Print warning message	4.8

4.1 Statement Execution

The call

```
CALL AAMAIN
```

is the main program loop of MAD. It reads statements from the current input file until it sees a STOP statement. MAD calls it twice, the first time to read the command dictionary, and the second time to read the user's commands.

For each command AAMAIN calls AAREAD to read and decode it. Then it calls the proper switch routine for any definition or subroutine command. For executable commands it calls AAEXEC to force the correct housekeeping to be done.

4.2 Switch Routine for Executable Commands

```
CALL AAEXEC(LABEL,KEY)
```

The parameters of AAEXEC are the command label and keyword respectively. The routine performs the necessary book-keeping and uses the process code of the command to select another switch routine which in turn calls the proper command routine. It is called for all executable commands, i. e. for commands having process codes greater than 10. If a new process code is introduced, a new path must be added to the multi-way IF-statement within this routine.

4.3 Book a New Statement Bank

```
CALL AABOOK(LBANK,LABEL,IPR,ISP,ILN,ILINK)
```

books a new statement bank and links it to the current keyword, pointed at by LCKEY. The routine assumes that the current keyword pointer LCKEY has been set, and that the KEYWORD common has been filled by a call to KWGET. The arguments of AABOOK are:

LBANK — Pointer to the bank booked (output).

LABEL — Name for the new bank (4 characters used, input).

IPR — Process code to be stored (input).

ISP — Subprocess code to be stored (input).

ILN — Input line number, indicating the beginning of the definition command (input).

ILINK — Number of the link in the current keyword bank which supports the new bank (input).

4.4 Attribute Decoding

```
CALL AAATTR(LDEF,LBANK,NKAT,EFLAG)
```

decodes the attributes of a statement and fills a command bank. It assumes that the current keyword pointer LCKEY is set, and that the KEYWORD common has been filled by a call to KWGET. It normally also checks that all attributes given are used, but optionally it returns when it sees the first unknown attribute. This permits to continue decoding using a different template, a feature used by the sequence editor of MAD. AAATTR has the following arguments:

LDEF — Pointer to the bank containing the *second default* (input).

LBANK — Pointer to the command bank to be filled (input).

NKAT — Number of attributes of the corresponding keyword (input). If this value is positive, AAATTR prints an error message if it sees an unknown attribute. If it is negative, its absolute value is the number of attributes, and the routine returns when it sees the first unknown attribute.

EFLAG — Logical error flag (output).

4.5 Copy Attribute from a Bank to Another

```
CALL AACOPY(LSRC,ILINK,LTAR)
```

copies one attribute from the source bank to the corresponding attribute in the target bank and builds all the required links. It has the arguments

LSRC Pointer to the source bank (input).

ILINK Number of the attribute to be copied (same in both banks, input).

LTAR Pointer to the target bank (input).

4.6 Dropping a Command or Definition

```
CALL AADROP(L)
```

deletes a command or definition pointed at by L. This call first marks any dependent expression and variable reference banks by setting the *drop bit* MXDRP and then unlinks them from the corresponding tables. Finally it deletes the command or definition bank and all its dependent banks.

4.7 Dump a Command or Element Definition

```
CALL AADUMP(L)
```

writes the contents of a command or definition pointed at by L on the ECHO file.

4.8 Program Messages

The calls

```
CALL AAFAIL(SUBR,NLINES,MSG)
CALL AAINFO(SUBR,NLINES,MSG)
CALL AAWARN(SUBR,NLINES,MSG)
```

are used for all messages produced by MAD. The arguments are the same for all three routines:

SUBR — Character constant containing the name of the calling routine. This will be printed as part of the message.

NLINES — Number of lines to be printed.

MSG — A character array dimensioned in these routines as

```
CHARACTER*(*) MSG(*)
```

containing the message. A common block has been defined as

```
      COMMON /MESSAGE/  MSG(8)
      CHARACTER*120     MSG
```

which may be filled before calling one of these routines.

The routines have the following purposes:

AAINFO — Print informational message. These messages can be suppressed by the command

```
OPTION,-INFO
```

or reactivated by the command

```
OPTION,INFO
```

AAWARN — Print warning message. These messages can be suppressed by the command

```
OPTION,-WARN
```

or activated by the command

```
OPTION,WARN
```

AAFAIL — Print fatal error message. These messages cannot be suppressed. If MAD reads input from a file, a fatal error causes it to enter scanning mode.

Example:

```
CHARACTER*(MCNAM) LABEL
...
LABEL = 'LABEL'
CALL UTLENG(LABEL, ILENG)
MSG(1) = 'Recursive call to "' // LABEL(1:ILENG) // '",'
MSG(2) = 'Call skipped.'
CALL AAWARN('AARUNS', 2, MSG)
```

prints the message

```
AARUNS.   ## Warning ## Recursive call to "LABEL",
          Call skipped.
```

The same format could be obtained by

```
      CHARACTER*(MCNAM) LABEL
      ...
      LABEL = 'LABEL'
      CALL UTLENG(LABEL, ILENG)
      WRITE (MSG, 910) LABEL(1:ILENG)
910   FORMAT('Recursive call to "',A,'".'/'Call skipped.')
      CALL AAWARN('AARUNS', 2, MSG)
```

4.9 Precomputed Maps

```
CALL AAPDRP
```

For efficiency reasons MAD stores various transfer maps for later use as long as there is no data change which makes them obsolete. An update of the BEAM bank makes transfer maps obsolete. Therefore it must always be done via ENPUT, which will call AAPDRP when it has finished. This ensures that all precomputed maps are dropped.

```
CALL AAPMOD
```

Before executing a command, AAEXEC calls AAPMOD to check definitions and their dependences. AAPMOD sets the status bit MXMOD for all obsolete maps, and then drops any marked maps. Marking is done by the call

```
CALL AAMARK(SUBR,LBANK)
```

where

SUBR — Character constant containing the name of the calling routine (used for debug messages).

LBANK — Pointer to the bank to be marked as modified.

Two more status bits are used for LUMPs. MXKNW marks lumps when their modify status is known, and MXLMP helps to detect recursive LUMP definitions.

4.10 Check Validity for Redefinitions

```
CALL AASMOD)(IDIR,LOLD,LNEW)
```

When a definition is replaced, its name must never be transferred to a different class of objects. One exception is allowed: Beam elements and beam lines may be interchanged. At each redefinition of a name MAD calls AASMOD in order to detect the acceptability of the redefinition, and to delete precomputed maps which may become obsolete due to the redefinition. The parameters are:

IDIR — Directory index of the item being redefined.

LOLD — Pointer to the old definition.

LNEW — Pointer to the new definition.

Chapter 5. Beamparam Module "BM"

The `BM` module implements the `BMPM` command. For a description refer to the MAD Physicist's Manual [14].

Chapter 6. Closed Orbit Correction Routines "CO"

The CO module implements the commands related to closed orbit. The routines are listed in table 6.1. The algorithms are described in the MAD Physicist's Manual.

Table 6.1: Routines in the CO module

Name	Purpose	Section
COCORR	Command routine for CORRECT command	-
COGDIS	Fetch dispersion readings for a plane from table	-
COGKIK	Fetch kicker strengths for a plane from table	-
COGMON	Fetch monitor readings for a plane from table	-
COLDIS	Orbit and dispersion correction by micado algorithm	-
COLORB	Orbit correction only by micado algorithm	-
COMAIN	Switch routine for CO module	-
COMICA	Command routine for MICADO command	-
COMDIS	Set up influence matrix for orbit and dispersion for a plane	-
COMORB	Set up influence matrix for orbit for a plane	-
COPDIS	Print dispersion readings	-
COPKIK	Print kicker strengths	-
COPMON	Print orbit readings	-
CORDIS	Command routine for GETDISP command	-
CORKIK	Command routine for GETKICK command	-
CORMON	Command routine for GETORBIT command	-
COSKIK	Increment kicker strengths for a plane in table	-
COTBLE	Set up corrector and monitor table	-
COWDIS	Command routine for PUTDISP command	-
COWKIK	Command routine for PUTKICK command	-
COWMON	Command routine for PUTORBIT command	-

Chapter 7. Decoder Routines "DC"

The DC module contains the high-level routine for data decoding. Its main entry point is DCATTR, the routine to decode a single command attribute. It is called from various places in the AA module, and details on the other routines are of no concern to the programmer.

Table 7.1: Routines in the DC module

Name	Purpose	Section
DCATTR	Main entry point to DC module	7.1
DCBEAM	Decode beam line reference	-
DCCONS	Decode constraint	-
DCFORM	Decode formal parameter list for beam line	-
DCINDX	Decode subscript list	-
DCINIT	Called from AAINIT to initialize DC module	-
DCLIST	Called from DCBEAM to decode beam line list	-
DCNAME	Decode name attribute	-
DCRANG	Decode range reference	-
DCREPT	Decode repetition count	-
DCSTRG	Decode character string	-
DCVREF	Decode variable reference	-

7.1 Decode a Single Command Attribute

```
CALL DCATTR(ITYPE,LBANK,IFRST,ILAST,EFLAG)
```

decodes a single statement attribute preceded by an optional repeat count. The routine assumes that the current keyword pointer LCKEY is set, and that the KEYWORD common has been filled by a call to KWGET. DCATTR has the arguments

ITYPE — MAD data type of the attribute (input),

LBANK — Pointer to the command bank to be filled (input),

IFRST — Attribute number within the command bank where to store the value (input),

ILAST — For dimensioned attributes, the last attribute number which may be filled with repetitions. This is incremented for each value read (input/output),

EFLAG — Logical error flag (output).

Direct use of this subroutine is not recommended, but it may be useful under special circumstances.

Chapter 8. Directory Routines "DI"

The routines of the DI module provide methods to create and access directories (Section 2.9).

Table 8.1: Routines in the DI module

Name	Purpose	Section
DIADD	Add a name to a directory (internal routine)	-
DIDEFI	Define a name in a directory	8.1
DIDROP	Drop a name from a directory	8.2
DIFIND	Find a name in a directory	8.3
DILOOK	Look up a name (internal routine)	-
DIMAKE	Build new directory	8.5
DINAME	Retrieve name from directory	8.4
DIREFE	Find reference to a possibly indefined name	8.6

8.1 Defining a Name

```
CALL DIDEFI(LDIR,LABEL,LBANK)
```

defines a new name in a given directory. If the name does not exist in the directory, a new entry is created for the name. If the name was already defined, DIDEFI gives appropriate messages. It may also drop and/or modify some precomputed data, when a redefinition makes this necessary. In all cases DIDEFI stores the bank pointer LBANK in the proper position of the pointer bank.

LDIR(4) Directory handle (input).

LABEL Name to be defined (input).

LBANK(1) Bank pointer to be associated with LABEL (input).

8.2 Removing a Definition

```
CALL DIDROP(LDIR, LABEL)
```

removes a bank pointer from a directory. If LABEL is found in LDIR, the corresponding bank pointer is cleared in the directory. The name itself remains in the directory, and the bank pointed at by the original bank pointer is not dropped.

LDIR(4) Directory handle (input).

LABEL Name whose definition is to be removed (input).

8.3 Finding a Name

When it is known that a name exists in a directory, both directory index and bank pointer can be retrieved by the call

```
CALL DIFIND(LDIR, LABEL, IDIR, LBANK)
```

`LDIR(4)` Directory handle (input).

`LABEL` Name to be looked for (input).

`IDIR` Directory index for LABEL in LDIR (output).

`LBANK(1)` Bank pointer associated with LABEL in LDIR (output).

If the name is not found, both `IDIR` and `LBANK` are returned as zero.

8.4 Retrieving a Name

```
CALL DINAME(LDIR, IDIR, LABEL)
```

Retrieves a name from a directory.

`LDIR(4)` Directory handle (input).

`IDIR` Directory index for the name requested (input).

`LABEL` Label at index `IDIR` in directory `LDIR` (output).

8.5 Creating a Directory

```
ALL DIMAKE(NENTRY, ISUP, LDIR)
```

creates an empty directory and links the new directory to the root bank. Thus the directory will automatically be dumped when the MAD data structure is dumped.

`NENTRY` The initial number of entries (input). If required, the directory grows later by adding `NENTRY` entries at a time.

`ISUP` Bias of the first supporting link in the root bank (input). There must be 4 contiguous links allocated to each directory.

`LDIR(4)` *directory handle* to be transmitted to other directory utilities (output). It must be dimensioned as `LDIR(4)`, and must reside in a reference link area. `DIMAKE` fills it with pointers to the four directory banks.

8.6 Referring to a Name

```
CALL DIREFE(LDIR, LABEL, IDIR)
```

Returns a reference to a name in a directory. If the name does not exist in the directory, a new entry is created for the name with a zero pointer. In any case the directory index for the name is returned.

`LDIR(4)` Directory handle (input).

`LABEL` Name to be looked for (input).

`IDIR` Directory index for `LABEL` in `LDIR` (output).

Chapter 9. Emittance-Related Routines "EM"

The EM module implements the emittance-related commands. The routines are listed in Table 9.1. The algorithms are described in in [5] and in the MAD Physicist's Manual.

Table 9.1: Routines in the EM module

Name	Purpose	Section
EMCE2I	Convert eigenvectors to internal sigma matrix form	-
EMCI2T	Convert beam matrix from internal to TRANSPORT form	-
EMCT2I	Convert beam matrix from TRANSPORT to internal form	-
EMDAMP	Calculate radiation damping in an element	-
EMEMD0	Command routine for EMIT command	-
EMEMG0	Work routine for EMIT command	-
EMEND0	Command routine for ENVELOPE command	-
EMENG0	Work routine for ENVELOPE command	-
EMENPR	Print beam sizes for ENVELOPE command	-
EMENSV	Save beam sizes for ENVELOPE command	-
EMEVD0	Command routine for EIGEN command	-
EMEVG0	Work routine for EIGEN command	-
EMEVPR	Print orbit and eigenvectors for EIGEN command	-
EMEVSV	Save orbit and eigenvectors for EIGEN command	-
EMINIT	Initialize radiation damping calculations	-
EMNORM	Command routine for NORMAL command	-
EMSSIG	Work routine for SAVESIGMA command	-
EMSUMM	Make summary calculations for radiation damping	-
EMTWD0	Command routine for TWISS1 command	-
EMTWG0	Work routine for TWISS1 command	-
EMTWPR	Print Mais-Ripken betatron functions	-
EMTWSV	Save Mais-Ripken functions for TWISS1 command	-

Chapter 10. Environment Setup "EN"

The EN module implements the commands to set up the machine environment. It also contains some services for other modules. The algorithms are described in the MAD Physicist's Manual as far as they concern physics calculations.

Table 10.1: Routines in the EN module

Name	Purpose	Section
ENBEAM	Command routine for BEAM command	-
ENDUMP	Command routine for SELECT command	-
ENFIX	Must be called for some commands in other modules to finish the environment set up begun with a BEAM command	10.1
ENFLAG	Set an output flag (internal routine)	-
ENFREQ	Must be called for some commands to change the RF frequency	10.2
ENGET	Called by AAEXEC to update BEAM common from the BEAM bank	10.3
ENMAIN	Switch routine for EN module	-
ENPRNT	Command routine for PRINT command	-
ENPUT	Called by any routine which changes the BEAM common, to update the BEAM bank.	10.4
ENRANG	Encode a range reference	10.5
ENSBET	Command routine for SAVEBETA command	-
ENSPCA	Define a SPLIT position (internal routine)	-
ENSPLT	Command routine for SPLIT command	-
ENSRNG	Service routine to perform an action over a range	10.6
ENSTYP	Service routine to perform an action on all elements with a certain TYPE attribute	10.7
ENUSE	Command routine for USE command	-

10.1 Fix up BEAM Data

```
CALL ENFIX
```

must be called by physics modules before using RF data and/or beam sizes which depend on a BEAM command. ENFIX figures out which data have been entered in BEAM and computes the missing values accordingly. It adjusts the frequencies of all RF cavities according to the *revolution frequency*.

10.2 Alter RF Frequencies

```
CALL ENFREQ(DELTAP,DELTAT)
```

adjusts the RF frequencies of all cavities such as to get an approximate average energy error of DELTAP. It returns the expected difference in revolution time DELTAT.

10.3 Update BEAM Common from BEAM Bank

```
CALL ENGET
```

This call is made by AAEXEC before each executable command to insure that the contents of the BEAM common agree with those of the BEAM bank.

10.4 Update Beam Bank from BEAM Common

```
CALL ENPUT
```

This call must be made by any routine which modifies the contents of the BEAM common, to ensure that the changes are transmitted into the BEAM bank.

10.5 Encode a Range Reference

```
CALL ENRANG(LRNG,RNGNAM)
```

returns a character representation of a range reference. It has two arguments:

LRNG — Pointer to a range reference bank (input).

RNGNAM — Character string equivalent to the input form which gave the range reference (output). This string must accommodate at least 60 characters.

10.6 Perform an Action on all Elements of a Range

```
CALL ENSRNG(LRNG,ACTION,IDUM1,IDUM2,DONE)
```

loops over the elements in the current range of the working beam line and calls the EXTERNAL routine ACTION for each element which belongs to a given range reference. It has the arguments:

LRNG — Pointer to the range reference bank to be used for selection (input).

ACTION — FORTRAN subroutine which will be called for each selected element. It must appear in an EXTERNAL statement in the calling routine and be declared as

```
SUBROUTINE ACTION(IPOS,IDUM1,IDUM2,EFLAG)
```

with the arguments

- IPOS — Position number for the selected element (input).
- IDUM1,IDUM2 — Transmitted from the routine calling ENSRNG (input).
- EFLAG — Logical error flag (output).

IDUM1,IDUM2 — Two arguments which will be transmitted to ACTION (input/output).

DONE — Is returned as .TRUE., if any element was selected (output).

10.7 Perform an Action on all Elements Having a Given TYPE

```
CALL ENSTYP(TYPE,ACTION,IDUM1,IDUM2,DONE)
```

loops over the elements in the current range of the working beam line nd calls the EXTERNAL routine ACTION for each element which has the specified value of the TYPE attribute. It has the arguments:

TYPE — Value of the TYPE attribute for selection (input).

ACTION — FORTRAN subroutine which will be called for each selected element. It must appear in an EXTERNAL statement in the calling routine and be declared as

```
SUBROUTINE ACTION(IPOS,IDUM1,IDUM2,EFLAG)
```

with the arguments

- IPOS — Position number for the selected element (input).
- IDUM1,IDUM2 — Transmitted from the routine calling ENSRNG (input/output).
- EFLAG — Logical error flag (output).

IDUM1,IDUM2 — Two arguments which will be transmitted to ACTION (input/output).

DONE — Returned as .TRUE., if any element was selected (output).

Chapter 11. Error Definitions "ER"

The ER module provides definitions for machine imperfections. The routines are listed in table 11.1. Details about the data structure are found in Section 2.10.2.

Table 11.1: Routines in the ER module

Name	Purpose	Section
ERALCA	Internal routine for ERALIG	-
ERALIG	Command routine for EALIGN command	-
ERFCCA	Internal routine for ERFCOM	-
ERFCOM	Command routine for EFCOMP command	-
ERFICA	Internal routine for ERFIEL	-
ERFIEL	Command routine for EFIELD command	-
ERLIST	Internal routine for ERPRNT	-
ERMAIN	Switch routine for ER module	-
EROPT	Command routine for EOPT command	-
ERPRNT	Command routine for EPRINT command	-
ERSAVE	Command routine for ESAVE command	-

Chapter 12. Expression Handler "EX"

The EX module provides methods to decode and evaluate expressions. Details about the data structure are found in Section 2.6.4.

Table 12.1: Routines in the EX module

Name	Purpose	Section
EXBIN	Internal routine for expression building	-
EXCONS	Internal routine for expression building	-
EXCOPY	Internal routine for expression building	-
EXDUMP	Dump an expression bank	12.1
EXEVAL	Evaluate a normal expression	12.2
EXEVL1	Evaluate a string expression	12.2
EXFILL	Called from AAEXEC to fill in references	12.3
EXHALF	Internal routine for expression building	-
EXINIT	Called from AAINIT to initialize EX module	
EXLKEX	Link a new expression bank	12.4
EXLKVR	Link a new variable reference bank	12.5
EXLOAD	Internal routine for expression building	-
EXMAKE	Build normal expression bank	12.6
EXMAK1	Build string expression bank	12.6
EXOPER	Internal routine for expression interpreter	-
EXORDR	Called from AAEXEC to order expressions	12.7
EXREAD	Decode normal expression	12.8
EXREFE	Internal routine for expression building	-
EXSTRG	Decode string expression	12.8
EXUNST	Internal routine for expression building	-
EXUPDT	Called from AAEXEC to update expressions	12.9

The EX module uses a set of common blocks, defined in COMDECK EXPRESS, for the expression table. This table is filled in by the expression decoders before building expression banks:

```
      PARAMETER          (MAXEXP = 100)
      COMMON /EXPRSA/    NXOPR, IXOPR(MAXEMP),
     +                   IXSUB1(MAXEXP), IXSUB2(MAXEXP), IXSUB3(MAXEXP)
      COMMON /EXPRSC/    AXBANK(MAXEXP), AXATTR(MAXEXP)
      COMMON /EXPRSR/    RXVAL(MAXEXP)
      SAVE               /EXPRSA/, /EXPRSC/, /EXPRSR/
      CHARACTER*(MCNAM) AXBANK, AXATTR
```

MAXEXP — Maximum number of postfix operations allowed.

NXOPR — Actual number of operations.

IXOPR — Operation codes (Section 2.6.4).

IXSUBi — i^{th} subscript for variable load operation.

AXBANK — Bank name for a variable load operation (parameter or bank attribute).

AXATTR — Attribute name for a variable load operation (bank attribute load).

RXVAL — Value for a constant load.

12.1 Dump an Expression Bank

```
CALL EXDUMP(L)
```

dumps the expression bank pointed at by L on the ECHO file.

12.2 Evaluate a Single Expression

```
CALL EXEVAL(L)
```

evaluates the expression pointed at by L by calling EXOPER for each operation in turn.

12.3 Fill in Variable References

```
CALL EXFILL
```

loops over the *variable reference table* and fills in any missing pointers. It gives messages for improper use of variables and/or undefined operands.

12.4 Link a New Expression Bank

```
CALL EXLKEX(L)
```

adds the expression bank pointed at by L to the *expression table*.

12.5 Link a New Variable Reference Bank

```
CALL EXLKVR(L)
```

adds the variable reference bank pointed at by L to the *variable reference table*.

12.6 Build an Expression Bank from Table

```
CALL EXMAKE(LBANK,ILINK,IDATA,RVAL,IEXPR)
```

builds a new expression bank, fills it with the current expression from common EXPRESS, and links the new bank to the *expression table* . It has the parameters

LBANK — Bank into which the result shall be stored, and by which the expression bank is supported (input).

ILINK — Link number in the bank at LBANK which will support the new expression bank (input).

IDATA — Bias in the bank at LBANK where the result shall be stored (input).

RVAL — Value of the expression (for a constant expression only, input).

IEXPR — Type of expression (input):

1: Constant expression,

2: Ordinary expression requiring evaluation,

3: Deferred expression.

A special routine builds expression banks for expressions coming from internal strings:

```
CALL EXMAK1(LTAB,LBNK,ILINK)
```

builds a new expression bank, fills it with the current expression from common EXPRESS, and links the new bank to the *expression table* . It has the parameters

LTAB — Pointer to an open dynamic table (Chapter 27) which may be used as a source for operands.

LBNK — Bank into which the result shall be stored, and by which the expression bank is supported (input).

ILINK — Link which will support the new expression bank (input).

In this case the operands of the expression can also come from a table column, allowing to combine the columns like in a spread-sheet program.

12.7 Order Arithmetic Expressions for Proper Evaluation

```
CALL EXORDR
```

orders the *expression table* for correct evaluation of dependent expressions. It skips *deferred expression*, but checks that all their operands are defined. The *order bit* MXORD marks expressions which are already in order, and the *defer bit* MXDEF identifies deferred expressions.

12.8 Decode a Single Expression

```
CALL EXREAD(IEVAL,RVAL,ISEEN)
```

decodes an arithmetic expression and stores the decoded expression in postfix notation in common EXPRESS. It does not build an expression bank. To this purpose EXMAKE must be called. The arguments are

IEVAL — Type of expression expected (input):

1: Must be constant expression.

2: Expression may depend on variables, but may not deferred.

3: All expression types allowed.

RVAL — Value of the expression (constant expression only, output).

ISEEN — Actual type of expression seen (output):

1: Constant expression, no later evaluation required.

2: Ordinary expression, must be linked to the expression table.

3: Deferred expression, requires deferred evaluation.

Expressions stored in strings, as used for plotting and listing tables, are decoded by the call

```
CALL EXSTRG(NAME,LTAB,LBNK,ILINK,RVAL,EFLAG)
```

It stores the decoded expression in postfix notation in common EXPRESS, and then builds an expression bank by calling EXMAK1. Its arguments are

NAME — Name of a table column, of a table descriptor, of a global string, or of an expression related to a table (input). The name is searched for in this order.

LTAB — Pointer to an open dynamic table (input).

LBNK — Bank into which the result shall be stored, and by which the expression bank is supported (input).

ILINK — Link which will support the new expression bank (input).

RVAL — Value of the expression (constant expression only, output).

EFLAG — Error flag (logical, output).

12.9 Evaluate All Non-Deferred Expressions

```
CALL EXUPDT
```

loops over the expressions ordered by EXORDR and evaluates each of them in turn by a call to EXEVAL.

Chapter 13. File Handlers "FL"

The FL module provides the switch and command routines for file access. It also contains a few file-handling utilities.

Table 13.1: Routines in the FL module

Name	Purpose	Section
FLASSI	Command routine for ASSIGN command	-
FLCALL	Command routine for CALL and RETURN commands	-
FLCLOS	Close a file	13.1
FLCSYS	Command routine for SYSTEM command	-
FLDELE	Delete a file	13.2
FLDUMP	Command routine for POOLDUMP command	-
FLEND	Shut down file system	13.3
FLINIT	Initialize file system	13.4
FLLOAD	Command routine for POOLLOAD command	-
FLMAIN	Switch routine for FL module	-
FLNAME	Retrieve file name form file table	13.5
FLNFIX	Convert file name depending on operating system	13.6
FLNSET	Build file name for standard file	13.6
FLOPEN	Open file, system-independent part	13.7
FLRTFS	Command routine for RETRIEVE command	-
FLSYST	Open file, system-dependent part	13.7
FLTELL	Command routine for STATUS command	-
FLTEXT	Open text file, system-dependent	13.7
FLWTFS	Command routine for ARCHIVE command	-
FLXCIT	Command routine for EXCITE and INCREMENT commands	-

The FL module uses a set of common blocks, FLTABLE, for the file table, used to hold all known file names:

```
      PARAMETER          (MAXDEF = 20, MAXFIL = 50)
      COMMON /FLTABC/    IDFN(0:MAXFIL), IDAC(0:MAXFIL), IDDR(0:MAXFIL),
     +                   IDFR(0:MAXFIL), IDLC(0:MAXFIL)
      COMMON /FLTABI/    IDST(0:MAXFIL), IDLR(0:MAXFIL), IDLF(0:MAXFIL)
      SAVE               /FLTABC/, /FLTABI/
      CHARACTER*(MCFIL)  IDFN
      CHARACTER*1        IDAC, IDDR, IDFR, IDLC
```

The table is indexed by logical unit number and contains:

MAXDEF — Highest reserved unit number (for standard files).

MAXFIL — Highest allocatable unit number.

IDFN — File names as character strings.

IDAC — Access codes: S for sequential, D for direct.

IDDR — Data flow direction: R for read, W for write.

IDFR — Format code: F for formatted, U for unformatted.

IDLC File location: D for disk, T for terminal, S for scratch.

IDST Status code: 0 for free entry, 1 for used entry, -1 for deleted file.

IDLR Record length in characters for direct access files.

IDLF File length in records for direct access files (relevant on IBM only).

13.1 Close a File

```
CALL FLCLOS(IUNIT,EFLAG)
```

Closes a logical file.

IUNIT Logical unit number for the file (input).

EFLAG Logical error flag (output).

13.2 Delete a File

```
CALL FLDELE(IUNIT,EFLAG)
```

Deletes a file from the file system.

IUNIT Logical unit number for the file (input).

EFLAG Logical error flag (output).

13.3 Shut down File System

```
CALL FLEND
```

closes all open files and deletes any scratch files.

13.4 Initialize File System

```
CALL FLINIT
```

initializes the internal file table.

13.5 Retrieve File Name from File Table

```
CALL FLNAME(IUNIT,FILNAM)
```

retrieves the file name for a logical unit from the file table. The name returned is the name under which the file is known to the operating system.

IUNIT Logical unit number for the file (input).

FILNAM String to receive the file name (output, at least MCFIL characters).

13.6 Convert File Name Depending on Operating System

```
CALL FLNFIX(STRNAM,MODE,FILNAM,LENG,EFLAG)
```

converts an internal logical stream name, as defined by the MAD program user, to an external file name which will be known to the operating system. This includes any format change to the file name required to make it acceptable to the operating system. On VM/CMS systems, e. g. the name is converted to upper case and split into the three parts (filename, filetype, filemode).

`STRNAM` Internal logical stream name (input character string specified by the program user).

`MODE` String containing the following four characters (input):

1^{st} **character, Access mode:** `S` for sequential, `D` for direct.

2^{nd} **character, Direction:** `R` for read, `W` for write.

3^{rd} **character, Format flag:** `F` for formatted, `U` for unformatted.

4^{th} **character, Location of file:** `D` for disk, `T` for terminal, `S` for scratch.

`FILNAM` External file name built, acceptable to the operating system (output). This must accommodate at least `MCFIL` characters.

`LENG` The actual length of the external file name (output).

`EFLAG` Logical error flag (output).

```
CALL FLNSET(STRNAM,FILNAM)
```

is a simplified version of the above, it is used for the MAD standard files.

`STRNAM` Internal logical stream name.

`FILNAM` External file name built, acceptable to the operating system (output). This must accommodate at least `MCFIL` characters.

13.7 Open a File

```
CALL FLOPEN(STRNAM,MODE,LREC,LFIL,IUNIT,EFLAG)
```

opens a file, and stores its external file name in the file table.

`STRNAM` Internal logical stream name.

`MODE` String containing the following four characters (input):

1^{st} **character, Access mode:** `S` for sequential, `D` for direct.

2^{nd} **character, Direction:** `R` for read, `W` for write.

3^{rd} **character, Format flag:** `F` for formatted, `U` for unformatted.

4^{th} **character, Location of file:** `D` for disk, `T` for terminal, `S` for scratch.

`LREC` Record length in characters (input, not relevant for formatted sequential files).

`LFIL` File length in records (input, only relevant for IBM direct access files).

IUNIT Logical unit number (output).

EFLAG Logical error flag (output).

The operating system dependent part of file open is done by FLSYST

```
CALL FLSYST(FILNAM,MODE,LREC,LFIL,IUNIT,EFLAG)
```

FILNAM External file name built by FLNFIX

MODE String containing the following four characters (input):

1^{st} **character, Access mode:** S for sequential, D for direct.
2^{nd} **character, Direction:** R for read, W for write.
3^{rd} **character, Format flag:** F for formatted, U for unformatted.
4^{th} **character, Location of file:** D for disk, T for terminal, S for scratch.

LREC Record length in characters (input, not relevant for formatted sequential files).

LFIL File length in records (input, only relevant for IBM direct access files).

IUNIT Logical unit number (input).

EFLAG Logical error flag (output).

For standard MAD files, MAD uses the simpler call

```
CALL FLTEXT(FILNAM,DIR,LREC,IUNIT,EFLAG)
```

FILNAM External file name built by FLNSET

DIR One-character variable (input): R for read, W for write.

LREC Record length in characters (input, no longer relevant, should be zero).

IUNIT Logical unit number (input).

EFLAG Logical error flag (output).

Chapter 14. HARMON Module "HA"

The HA module implements the commands of the HARMON module. The algorithms used are described in [7] and in the MAD Physicist's Reference Manual.

Table 14.1: Routines in the HA module

Name	Purpose	Section
HAATUN	Find derivatives $\partial Q/\partial\varepsilon$	-
HABEGN	Command routine for HARMON command	-
HACELL	Command routine for HCELL command	-
HACFIT	Adjust chromaticities to desired values	-
HACHCL	Calculate chromaticities	-
HACHRM	Command routine for HCHROMATICITY command	-
HADBET	Find derivatives $\partial\beta/\partial\delta$	-
HADDSP	Find derivatives $\partial D/\partial\delta$ and $\partial^2 D/\partial\delta^2$	-
HADTUN	Find derivatives $\partial^2 Q/\partial\delta^2$ and $\partial^3 Q/\partial\delta^3$	-
HAFCN	Matching routine for HA module	-
HAFUNC	Command routine for HFUNCTION command	-
HALONG	Set up "long" table of lattice functions	-
HAMAIN	Switch routine for HA module	-
HAPAVE	Print lattice functions averaged over elements	-
HAPRNT	Matching print-out	-
HARESC	Command routine for HRESONANCE command	-
HARESO	Compute resonance coefficients	-
HARSIG	Internal routine for HARESO	-
HASHRT	Set up "short" table of averaged lattice functions	-
HASTRG	Retrieve multipole strengths	-
HATHIN	Organize thin lens calculations	-
HATUNE	Command routine for HTUNE command	-
HAVARY	Command routine for HVARY command	-
HAWEIG	Command routine for HWEIGHT command	-
HA4ANA	Find fourth-order resonance coefficients	-
HA4SUM	Internal routine for HA4ANA	-

Chapter 15. Keyword Module "KW"

The KW module is the keyword decoder. It provides utilities for accessing keyword banks.

Table 15.1: Routines in the KW module

Name	Purpose	Section
KWDIM	Decode type and dimension for one keyword attribute	-
KWDUMP	Dump keyword bank on ECHO file	15.1
KWGET	Unpack keyword bank to KEYWORD common	15.2
KWGRP	Decode one keyword attribute group	-
KWMAIN	Command routine for KEYWORD and KEYEDIT commands	-
KWMAKE	Book keyword bank and link it to master keyword	15.3
KWPUT	Pack KEYWORD common to keyword bank	15.4

15.1 Write Keyword Definition on ECHO File

```
CALL KWDUMP(L)
```

writes the keyword definition pointed at by L on the ECHO file.

15.2 Unpack Keyword Definition

```
CALL KWGET(LKEY,ILN,IPR,ISP,NKAT)
```

unpacks a keyboard definition and stores information in local variables and in the KEYWORD common.

LKEY Pointer to the keyword to be unpacked (input).

ILN Line number of last definition of the keyword (output).

IPR Process code for this keyword (output).

ISP Subprocess code for this keyword (output).

NKAT Number of keyword attributes for this keyword (output).

This call fills the following common arrays:

IATYPE MAD data types for the keyword attributes.

IADIM1 First dimensions for the keyword attributes.

IADIM2 Second dimensions for the keyword attributes.

IADIM3 Third dimensions for the keyword attributes.

KATNAM Names of keyword attributes.

15.3 Book New Keyword Bank

```
CALL KWMAKE(LABEL,IPR,ISP,NKAT)
```

books a new keyword bank, links it to the master keyword and to the *keyword!directory*.

`LABEL` Name of the new keyword (input).

`IPR` The process code (input).

`ISP` The subprocess code (input).

`NKAT` Number of keyword attributes (input).

15.4 Pack Keyword Definition

```
CALL KWPUT(LKEY,NKAT)
```

packs the following common arrays to a keyword bank:

`IATYPE` MAD data types for the keyword attributes.

`IADIM1` First dimensions for the keyword attributes.

`IADIM2` Second dimensions for the keyword attributes.

`IADIM3` Third dimensions for the keyword attributes.

`KATNAM` Names of keyword attributes.

The call has two arguments:

`LKEY` Pointer to the keyword bank to be filled (input).

`NKAT` Number of keyword attributes (input).

Chapter 16. Lie-Algebra Module "LA"

The LA module contains routines to operate on Lie-algebraic maps [8]. A Lie-algebraic map represents the transformation of phase space from entrance to exit of an element as the Lie transformation

$$z_i^{(2)} = e^{:f_1:} e^{:f_2:} e^{:f_3:} e^{:f_4:} \ldots z_i^{(1)}, \qquad for \quad i = 1 \ldots 6.$$

The f_i are homogeneous polynomials in Z.

Many routines have been lifted from MARYLIE, but most of them have been extensively modified. There are also some new routines. Note that not all routines keep a f_1 term. The descriptions below use the following conventions:

FP is an array containing the generators for a map $\mathcal{F}$.

FM is a 6×6 matrix representing the linear terms of the map $\mathcal{F}$.

The algorithms used are documented in [8] and in the MAD Physicist's Reference Manual.

Table 16.1: Routines in the LA module

Name	Purpose	Section
LABETA	Extract betatron portion of a transfer map	16.2.1
LABRKS	Compute Poisson brackets relevant for the Lie series	-
LACHRM	Chromatic expansion of a static transfer map	16.2.2
LADC2R	Transform dynamic map from Cartesian basis to resonance basis	16.3.2
LADEIG	Find eigenvalues of a dynamic transfer matrix	16.3.1
LADPUR	Internal routine to purify dynamic maps	-
LADPU2	Transform dynamic map to Floquet variables	16.3.2
LADPU3	Transform third order part of dynamic map to normal form	16.3.2
LADPU4	Transform fourth order part of dynamic map to normal form	16.3.2
LADR2C	Transform dynamic map from resonance basis to Cartesian basis	16.3.2
LADYNA	Command routine for the DYNAMIC command	-
LAEMIT	Command routine for the NORMAL command Note that this routine no longer uses Lie algebra.	-
LAFXFN	Transform polynomial on phase space with a transfer map	16.1.2
LALUMP	Find Lie algebraic transfer map for a LUMP	16.1.1
LAMAIN	Switch routine for the LA module	-
LAMOVE	Transform orbit with a transfer map and return map around orbit	16.1.3
LASC2R	Transform static map from Cartesian basis to resonance basis	16.2.4
LASEIG	Find eigenvalues of a static transfer matrix	16.2.3
LASPUC	Transform third order chromatic part of static map to normal form	16.2.4
LASPUG	Transform third order geometric part of static map to normal form	16.2.4
LASPUR	Internal routine to purify static maps	-
LASPU4	Transform fourth order part of static map to normal form	16.2.4
LASR2C	Transform static map from resonance basis to Cartesian basis	16.2.4
LASTAT	Command routine for the STATIC command	-
LATRNS	Track orbits with the truncated Lie series	-
LATURN	Find one turn map for the working beam line	16.1.1

16.1 Operations on General Maps

16.1.1 One-Turn Maps

```
CALL LATURN(N,FP,FM)
```

returns the map $\mathcal{F}$ for the main beam line expansion. There are three arguments:

`N` Desired order N of the map (input, $2 \leq$ `N` ≤ 6), i. e. the order of the highest polynomial kept.

`FP,FM` Generators and matrix of $\mathcal{F}$ (output).

To improve speed, transfer maps are kept in ZEBRA banks linked to the sequence bank (Section 2.10), as soon as they are calculated. When `LATURN` is called, MAD looks for such a map with the given order. If finds one, it immediately returns the map. Otherwise it creates this map by using the call

```
CALL LALUMP(N,LSEQ,FP,FM)
```

which concatenates the maps for all elements in a specified beam line expansion. There are four arguments:

`N` Desired N order of the map (input, $2 \leq$ `N` ≤ 6), order of the highest polynomial kept.

`LSEQ` Pointer to the beam line expansion, normally `LCSEQ` (input, Section 2.10),

`FP,FM` Generators and matrix of $\mathcal{F}$ (output).

16.1.2 Transform Polynomial with a Transfer Map

```
CALL LAFXFM(N,GP,GM,FP,HP)
```

Transforms a polynomial F with a map $\mathcal{G}$. There are five arguments:

`N` Order N of the map $\mathcal{G}$ ($2 \leq$ `N` ≤ 6, input),

`GP,GM` Generators and matrix of $\mathcal{G}$ (input),

`FP` Polynomial F to be transformed (input),

`HP` Resulting polynomial H (output).

F and H are stored in the same format as the generators of a map.

16.1.3 Transform Map to Map about an Orbit

```
CALL LAMOVE(N,FP,FM,ORBIT,HP,HM)
```

Moves a given orbit through a map $\mathcal{F}$ and returns the map $\mathcal{H}$ for the motion around the orbit. There are six arguments:

`N` Order N of the map $\mathcal{F}$ ($2 \leq$ `N` ≤ 6, input),

`FP,FM` Generators and matrix for $\mathcal{F}$ (input),

`ORBIT` The orbit to be moved through $\mathcal{F}$,

`HP,HM` Map $\mathcal{H}$ of order `N` around the orbit (output).

16.2 Operations on Static Maps

A static map does not change particle energy.

16.2.1 Betatron factor of a Static Map

```
CALL LABETA(N,FP,FM,BP,BM)
```

extracts the betatron factor $\mathcal{B}$ from a static map. There are five arguments:

`N` Order N of the given map (input, $2 \leq$ `N` ≤ 4),

`FP,FM` Static map $\mathcal{F}$,

`BP,BM` Betatron factor $\mathcal{B}$ of $\mathcal{F}$ (output).

16.2.2 Chromatic expansion of a Static Matrix

```
CALL LACHRM(N,FP,FM,AM1,AM2)
```

extracts the first and second derivatives of the transfer matrix with respect to the energy error from a static map. There are five arguments:

`N` Order N of the given map $\mathcal{F}$ (input, $2 \leq$ `N` ≤ 4),

`FP,FM` Generators and matrix of $\mathcal{F}$ (input),

`AM1,AM2` Two 6×6 matrices which will receive the first and second derivative.

16.2.3 Eigenvalues and Eigenvectors of a Static Matrix

```
CALL LASEIG(A,REEIG,AIEIG,E)
```

Computes the eigenvalues and eigenvectors of the 4×4 submatrix of a static 6×6 matrix A. The routine has four arguments:

`A` Static matrix A,

`REEIG,AIEIG` Two 6 vectors which receive the real and imaginary parts of the eigenvalues. The fifth and sixth eigenvalues are set to one.

`E` 6×6 matrix which receives the eigenvectors. A real vector is stored in a single column. A complex pair is stored in two consecutive columns, using one column for the real part, and one for the imaginary part.

16.2.4 Purify Static Map

Transformation to normal form involves several steps of "purification". This operation consists in repeated conjugation with several maps to remove undesired terms successively. The four routines

```
CALL LASPU2(N,FP,FM,GP,GM,TP,TM)
CALL LASPUC(N,FP,FM,GP,GM,TP,TM)
CALL LASPUG(N,FP,FM,GP,GM,TP,TM)
CALL LASPU4(N,FP,FM,GP,GM,TP,TM)
```

have the following action:

`LASPU2` Removes coupling terms of second order by conjugation with the matrix of eigenvectors,

`LASPUC` Removes the chromatic terms of third order,

`LASPUG` Removes the geometric terms of third order,

`LASPU4` Removes terms of fourth order,

The four routines must be called in the order listed. They all have the arguments

`N` Order of all maps involved ($2 \leq$ `N` ≤ 4, input),

`FP,FM` Map $\mathcal{F}$ to be purified (input),

`GP,GM` Result $\mathcal{G}$ of conjugation (output),

`TP,TM` Map $\mathcal{T}$ used for conjugation (output): $\mathcal{G} \leftarrow \mathcal{T}\mathcal{F}\mathcal{T}^{-1}$.

A purified static map still contains some non-removable non-linearities. The call

```
CALL LASC2R(N,F,G)
```

transforms such a map to static resonance form. It has three arguments:

`N` Order of polynomials involved ($2 \leq$ `N` ≤ 4, input),

`F` Generators F of the purified static map $\mathcal{F}$,

`G` Resonance coefficients G in a static resonance base.

```
CALL LASR2C(N,G,F)
```

is the opposite operation.

16.3 Operations on Dynamic Maps

A dynamic map is a map which may change the particle energy.

16.3.1 Eigenvalues and Eigenvectors of a 6×6 Matrix

```
CALL LADEIG(FM,REEIG,AIEIG,AM)
```

Computes the eigenvalues and eigenvectors of the 6×6 matrix A. The routine has four arguments:

`A` Dynamic matrix A,

`REEIG,AIEIG` Two 6 vectors which receive the real and imaginary parts of the eigenvalues.

`E` 6×6 matrix which receives the eigenvectors. A real vector is stored in a single column. A complex pair is stored in two consecutive columns, using one column for the real part, and one for the imaginary part.

16.3.2 Purifying Dynamic Map

Transformation to normal form involves several steps of "purification". This operation consists in repeated conjugation with several maps to remove undesired terms successively. The three routines

```
CALL LADPU2(N,FP,FM,GP,GM,TP,TM)
CALL LADPU3(N,FP,FM,GP,GM,TP,TM)
CALL LADPU4(N,FP,FM,GP,GM,TP,TM)
```

have the following action:

`LADPU2` Removes coupling terms of second order by conjugation with the matrix of eigenvectors,

`LADPU3` Removes the terms of third order,

`LADPU4` Removes terms of fourth order,

The three routines must be called in the order listed. They all have the arguments

`N` Order N of all maps involved ($2 \leq$ `N` ≤ 4, input),

`FP,FM` Map $\mathcal{F}$ to be purified (input),

`GP,GM` Result $\mathcal{G}$ of conjugation (output),

`TP,TM` Map $\mathcal{T}$ used for conjugation (output): $\mathcal{G} \leftarrow \mathcal{T}\mathcal{F}\mathcal{T}^{-1}$.

A purified dynamic map still contains some non-removable non-linearities. The call

```
CALL LADC2R(N,F,G)
```

transforms such a map to dynamic resonance form. It has three arguments:

`N` Order N of polynomials involved ($2 \leq$ `N` ≤ 4, input),

`F` Generators F of the purified dynamic map $\mathcal{F}$,

`G` Resonance coefficients G in a dynamic resonance base.

```
CALL LADR2C(N,G,F)
```

is the opposite operation.

Chapter 17. Lie-Algebraic Maps "LM"

The "LM" module computes Lie-algebraic maps for single elements and performs elementary operations thereon. Note that not all routines keep a f_1 term. The algorithms used are documented in the MAD Physicist's Reference Manual.

Table 17.1: Routines in the LM module

Name	Purpose	Section
LMARB	Transfer map for an arbitrary element (not yet implemented)	-
LMBEND	Transfer map for a bend includeing fringe fields	-
LMCANX	Prepare transfer map for symplectic tracking	-
LMCAT	Concatenate two transfer maps	17.1
LMCLOR	Find fixed point (closed orbit) of a transfer map	17.2
LMCOPY	Copy a transfer map	17.3
LMCORR	Transfer map for an closed orbit corrector	-
LMDPRT	Print transfer map in dynamic resonance form	17.9
LMDRF	Transfer map for a drift space	-
LMDSP1	Transfer map for a misalignment at element entrance	-
LMDSP2	Transfer map for a misalignment at element exit	-
LMELEM	Switch routine to track through any element	17.4
LMEXPO	Calculate Lie transformation of a polynomial	17.5
LMFIXP	Find fixed point of a static map	17.6
LMFRG1	Transfer map for a fringe field at dipole entrance	-
LMFRG2	Transfer map for a fringe field at dipole exit	-
LMG1MV	Move g_1 part over a Lie operator	-
LMINV	Invert a transfer map	17.7
LMMAP	Switch routine to compute transfer map for any element	17.7.1
LMMASK	Mask out terms for given orders	17.8
LMMULT	Transfer map for a thin multipole	-
LMNEWT	Non-linear part of symplectic tracking	-
LMOCT	Transfer map for a long octupole	-
LMONE	Return identity map	17.8.1
LMPRNT	Print transfer map matrix and generators	17.9
LMQUAD	Transfer map for a long quadrupole	-
LMREFL	Reflect transfer map	17.10
LMREVF	Reverse factorization of a map	17.11
LMRF	Transfer map for an RF cavity of zero length	-
LMSAND	Conjugate transfer map with another transfer map	17.12
LMSECT	Transfer map for a sector dipole without fringe fields	-
LMSEP	Transfer map for an electrostatic separator	-
LMSEXT	Transfer map for a long sextupole	-
LMSOL	Transfer map for a long solenoid without fringe fields	-
LMSPRT	Print transfer map in static resonance form	17.9
LMSROT	Transfer map for a rotation about the s-axis	-
LMTILT	Conjugate transfer map with rotation around s-axis	17.13
LMTRAK	Track orbits by the symplectic method	-
LMUSER	Transfer map for a user-define element	17.13.1
LMYROT	Transfer map for a rotation about the y-axis	-

17.1 Concatenate two Maps

```
CALL LMCAT(N,FP,FM,GP,GM,HP,HM)
```

concatenates two Lie-algebraic maps $\mathcal{F}$ and $\mathcal{G}$ and returns the resulting map $\mathcal{H}$. It takes five input arguments:

`N` Order N of all maps involved ($2 \leq$ `N` ≤ 6).

`FP,FM` Generators and matrix of the *first* map $\mathcal{F}$ in beam order.

`GP,GM` Generators and matrix of the *second* map $\mathcal{G}$ in beam order.

It returns two arguments:

`HP,HM` Generators and matrix of the resulting map $\mathcal{H}$.

LMCAT keeps displacement terms, if present.

17.2 Find Closed Orbit and Map about the Closed Orbit

```
CALL LMCLOR(LINE,N,FP,FM,DELTAT,DELTAP,HP,HM,ORBIT)
```

computes the first-order fixed point, i. e. the closed orbit of a map $\mathcal{F}$. It returns the orbit for the fixed point and the map $\mathcal{H}$ relative to the fixed point. It has five input arguments:

`LINE` Name of the beam line having the map $\mathcal{F}$ as `CHARACTER*(MCNAM)`.

`N` Order of the map F.

`FP,FM` Generators `FP` and matrix `FM` of the map $\mathcal{F}$.

`DELTAP` (Average) relative energy error desired for the fixed point.

and returns four arguments:

`DELTAT` Path length difference for the fixed point, compared to reference particle.

`HP,HM` Generators `HP` and matrix `HM` of the map $\mathcal{H}$.

`ORBIT` Orbit for the fixed point.

17.3 Copy a Map

```
CALL LMCOPY(N,FP,FM,GP,GM)
```

Copies a map $\mathcal{F}$, into a map $\mathcal{G}$. It has three input arguments:

`N` Order of the map.

`FP,FM` Map $\mathcal{F}$ to be copied.

and two output arguments:

`GP,GM` Copied map $\mathcal{G}$.

17.4 Track with the Generating Function Method

```
CALL LMELEM(ITURN,NORD,NSUP,IPOS,SUML,TRACK,NUMBER,NTRACK)
```

tracks a set of orbits through any element using the symplectic tracking method. It assumes that UTBEAM and UTELEM have been called beforehand to set the relevant pointers in common REFER.

ITURN Turn number, used for messages about lost particles (input).

NORD Order to be used for the transfer map(input).

ISUP Superperiod number, used for messages about lost particles (input).

IPOS Position number in the machine, used for messages(input).

SUML Accumulated length, updated by the routine (input/output)

TRACK Orbit positions to be updated (input/output).

NUMBER List of orbit numbers (input/output).

NTRACK Number of surviving orbits (input/output).

The orbits are compacted when an orbit is lost. Thus the n^{th} orbit is stored in TRACK(*,i), and its number is found in NUMBER(i).

17.5 Apply Exponential of a Lie operator to a Polynomial

```
CALL LMEXPO(FP,IFPORD,GP,IARORD,HP,IMXORD)
```

applies the Lie transformation for a homogeneous polynomial F_n to a set of generators $\{G_i\}$ and returns the resulting set $\{H_k\}$:

$$H_k = \sum e^{:F_n:} G_i, \qquad \text{terms of order } k.$$

It has five input arguments:

FP Coefficients of the homogeneous polynomial F_n of order n.

IFPORD Order n of F_n.

GP The coefficients of the set of homogeneous polynomials $\{G_i\}$.

IARORD If positive, $\{G_i\}$ contains only order IARORD. If negative, $\{G_i\}$ contains all polynomials up to order -IARORD.

IMXORD Order at which to truncate the result.

The result is returned in one argument:

HP Set of homogeneous polynomials $\{H_k\}$ generated.

17.6 Find Fixed Point of a Static Map as a Function of δ

```
CALL LMFIXP(FP,FM,AP,AM,TP,TM)
```

transforms an order-4 map $\mathcal{F}$ by conjugation with a map $\mathcal{T}$: $\mathcal{A} \leftarrow \mathcal{T}\mathcal{F}\mathcal{T}^{-1}$ such that $\mathcal{T}$ contains the first through third order dispersion of $\mathcal{F}$, and $\mathcal{A}$ is the map about the fixed point. There are two input arguments:

`FP,FM` Fourth-order map $\mathcal{F}$.

and four output arguments:

`AP,AM` Map $\mathcal{A}$ about the fixed point.

`TP,TM` Transformation $\mathcal{T}$ to the fixed point.

17.7 Invert Lie-algebraic Map

```
CALL LMINV(N,FP,FM,GP,GM)
```

inverts the map $\mathcal{F}$ of order `N` and returns the result in $\mathcal{G}$.

17.7.1 Return Transfer Map for Any Element

```
CALL LMMAP(N,EL,FP,FM)
```

expects that the relevant pointers have been set by the calling routine:

`LCELM` The data bank for the current element (must be non-zero).

`LCALI` Pointer to misalignment bank (may be zero).

`LCFLD` Pointer to field error bank (may be zero).

`LCCOM` Pointer to corrector or monitor bank (must be non-zero for a corrector or monitor, when an orbit correction has been made).

It extracts the relevant data and returns:

`EL` The length of the current element,

`FP,FM` Map $\mathcal{F}$ for the current element.

17.8 Wipe out Monomial Coefficients as Specified

```
CALL LMMASK(N,WIPE,FP,FM,GP,GM)
```

wipes out selected orders in the map $\mathcal{F}$. It has four input arguments:

`N` Order of $\mathcal{F}$.

`WIPE` Logical array of dimension at least `N`. If `WIPE(I)` is `.FALSE.`, the terms of order `I` will be copied from $\mathcal{F}$ to $\mathcal{G}$, otherwise these terms will be cleared in $\mathcal{G}$.

`FP,FM` Input map $\mathcal{F}$.

and two output arguments:

`GP,GM` Map $\mathcal{G}$ resulting from wiping out the selected orders.

17.8.1 Identity Map

```
CALL LMONE(N,FP,FM)
```

Sets the map $\mathcal{F}$ to the identity map.

17.9 Print Representations of Lie-algebraic Maps

```
CALL LMPRNT(IUNIT,N,FP,FM)
```

prints the map $\mathcal{F}$ of order `N` on logical unit `IUNIT`.

```
CALL LMDPRT(IUNIT,N,FP)
```

prints the dynamic resonance generators of a map computed by `LADC2R`. It has three input parameters:

`IUNIT` Logical unit number to receive the output.

`N` Maximum order of the terms to be printed.

`FP` Array containing the dynamic resonance generators.

```
CALL LMSPRT(IUNIT,N,FP)
```

prints the static resonance generators of a map computed by `LASC2R`. It has three input parameters:

`IUNIT` Logical unit number to receive the output.

`N` Maximum order of the terms to be printed.

`FP` Array containing the static resonance generators.

17.10 Reflect a Map

```
CALL LMREFL(FP,FM,N,GP,GM)
```

Reflects the map $\mathcal{F}$ of order `N` and returns the result in $\mathcal{G}$. Reflection is equivalent to running through the elements producing $\mathcal{F}$ in inverse (reflected) order.

17.11 Reverse the Order of Factorization

```
CALL LMREVF(FP,FM,N,GP,GM)
```

Reverses the order of factorization of the map $\mathcal{F}$ (order `N`), and returns the result in $\mathcal{G}$.

17.12 Conjugate a Map

```
CALL LMSAND(N,GP,GM,TP,TM,AP,AM)
```

Conjugates the map $\mathcal{G}$ with the map $\mathcal{T}$: $\mathcal{A} \leftarrow \mathcal{T}\mathcal{G}\mathcal{T}^{-}1$.

17.13 Modify Map for Rotated Elements

```
CALL LMTILT(N,TILT,FP,FM)
```

Conjugates the map $\mathcal{F}$ with the map $\mathcal{R}$ representing a rotation about the s-axis by the angle `TILT` (in rads): $\mathcal{F} \leftarrow \mathcal{R}F\mathcal{R}^{-1}$.

17.13.1 Map for User-Defined Element

```
CALL LMUSER(N,ISP,EL,FP,FM)
```

may be replaced by the user to implement tracking through user-defined elements. It should take the element parameters from the element bank, pointed at by `LCELM` (Section 3.1).

`N`	Order of the map desired (input).
`ISP`	Subprocess code for the element (input).
`EL`	Element length (output).
`FP,FM`	Computed transfer map (output).

Chapter 18. Beam Lines "LN"

The LN module provides methods for decoding and accessing beam lines, beam lists, and sequences.

Table 18.1: Routines in the LN module

Name	Purpose	Section
LNBEAM	Command routine for LINE command	-
LNCHCK	Check that a valid USE command was seen	18.1
LNDROP	Delete a beam line expansion and anonymous drifts which it may contain due to a SEQUENCE	-
LNDUMP	Write a LINE or LIST definition on the ECHO file	18.2
LNEBGN	Command routine for SEQEDIT command	-
LNECYC	Command routine for CYCLE command	-
LNEINS	Command routine for INSTALL command	-
LNEMOV	Command routine for MOVE command	-
LNEREF	Command routine for REFLECT command	-
LNEREM	Command routine for REMOVE command	-
LNFORM	Fill in formal arguments for a LINE to be expanded	-
LNINIT	Called by AAINIT to initialize LN module	-
LNLIST	Command routine for LIST command	-
LNMAIN	Switch routine for LN module	-
LNMAKE	Book a LINE bank	-
LNMARK	Mark maps for working beam line as obsolete	-
LNPMOD	Propagater modification flags for LINE and LUMP	-
LNREFE	Build line expansiong from a beam line reference	18.3
LNSEQ	Command routine for SEQUENCE command	-
LNXLST	Expand beam line sequence, called by LNXPND	-
LNXPND	Get next item of replacement list, called by LNREFE	-
LNXPUT	Add element to line expansion	-
LNXRES	Reset replacement lists, called by LNXPND	-
LNXSEQ	Expand beam line sequence, called by LNXPND	-

When a SEQUENCE is decoded, MAD allocates new storage only for elements which have new attributes with respect to the class object. Otherwise it duplicates the pointer to the class object. Both the class and the copy are then marked as an alias by setting the *alias bit* MXALS. The class object is also marked as a class by setting the *class bit* MXCLS.

18.1 Check for Valid USE Command

```
CALL LNCHCK(COMAND,EFLAG)
```

checks that a working beam line has been set, and that it has not been deleted due to subsequent data changes. It has two arguments:

COMAND — A character string containing the current command name (input).

EFLAG — Logical error flag (output).

18.2 Dump Beam Line Definition

```
CALL LNDUMP(LLINE,LABEL)
```

writes a beam line definition on the ECHO file, using the arguments

LLINE Pointer to the definition to be dumped.

LABEL Name of the definition to be dumped.

18.3 Expand a Beam Line Reference

```
CALL LNREFE(LBANK,IBIAS,LSEQ,LSUP,ISUP)
```

Builds an expansion for a given beam line reference. The banks produced are:

LSDIR Directory indices for elements and lines belonging to the expansion.

LSFLG Flag words (Table 2.18).

LSNUM Element numbers (Table 2.19).

The other banks are built by other routines as required. The arguments of LNREFE are

LBANK Pointer to the command bank supporting the beam line reference bank (input).

IBIAS Attribute number of the beam line reference (input).

LSEQ Pointer to the new beam line expansion tree (output). For the working beam line this is LCSEQ.

LSUP Pointer to the bank which will support the expansion tree (input). For the working beam line this is LROOT.

ISUP Bias in the supporting bank for the pointer to the expansion tree (input). For the working beam line this is -MCSEQ, thus the link LQ(LROOT-MCSEQ) points to the working beam line.

Chapter 19. Matching Module "MT"

The MT module is the matching module. The methods are based on the MINUIT program, written by F. James [16]. There is also a method LMDIF lifted from the DIMAD program, originally written at SLAC by B. S. Garbov et al.

Table 19.1: Routines in the MT module

Name	Purpose	Section
MTACON	Add constraint to a position	19.1
MTBTIN	Find periodic solution of linear lattice functions	-
MTBTTK	Track linear lattice functions trough an element	-
MTCELL	Command routine for CELL command	-
MTCOND	Evaluate matching conditions, internal routine for MTEND and MTFCN	-
MTCONS	Command routine for CONSTRAINT command	-
MTCPLE	Command routine for COUPLE command	-
MTDERI	Find first and second derivatives of penalty function	-
MTEND	Command routine for END command	19.1
MTFCN	Compute matching functions	19.1
MTFIX	Command routine for FIX command	-
MTGETI	Get internal parameter values	-
MTHESS	Compute covariance matrix	-
MTINIT	Initialize MT module	-
MTLINE	Find minimum along a line in parameter space	-
MTLMDF	Command routine for LMDIF command	-
MTMAIN	Switch routine for MT module	19.2
MTMIGR	Command routine for MIGRAD command	-
MTMIG1	Minimize by MIGRAD method	-
MTMTCH	Command routine for MATCH command	-
MTPINI	Initialize calculation of target functions	19.1
MTPMOD	Delete precomputed transfer maps after parameter change	-
MTPRNT	Print-out for matching	-
MTPSDF	Ensure that covariance matrix is positive definite	-
MTPUTI	Put internal parameter values	-
MTRAZZ	Internal routine for MTSIM1	-
MTRMAT	Command routine for RMATRIX command	-
MTSIMP	Command routine for SIMPLEX command	-
MTSIM1	Minimize by SIMPLEX method	-
MTTMAT	Command routine for TMATRIX command	-
MTVARY	Command routine for VARY command	-
MTVFND	Look up a variable in the variable table	-
MTWEIG	Command routine for WEIGHT command	-
FDJAC	Internal routine for LMDIF	-
LMDIF	Minimize by LMDIF method	-
LMPAR	Internal routine for LMDIF	-

19.1 Changes Required to Add New Constraint Types

A new constraint type can easily be added to the present matching module. First, one has to link a command routine to decode the constraint to the switch routine MTMAIN. The command routine must link the constraint conditions to the working beam line by calling MTACON (possibly selecting the places via ENSRNG and/or ENSTYP. The actual evaluation of the constraint must be added to MTCOND. If the condition requires initialization, like the LINE=line condition for the CONSTRAINT command, this initialization must be added to MTPINI.

19.2 Changes Required to Add New Matching Methods

The command routine to execute a new matching method must be linked to the switch routine MTMAIN. It can get access to the target functions via

```
CALL MTFCN(NF,NX,X,FVEC,IFLAG)
```

with the arguments

NF	Number of target functions (input).
NX	Number of variable parameters (input).
X	Values of variable parameters (input). This must be dimensioned at NX.
FVEC	Values of the target functions (output). This must be dimensioned at NF.
IFLAG	Integer error flag (output). If this value is non-zero upon return, the matching method should discard this point in parameter space as illegal.

Chapter 20. Matrix Utilities "M66"

The M66 module contains utilities for operations on 6×6 matrices.

Table 20.1: Routines in the M66 module

Name	Purpose	Section
M66ADD	Add two matrices	20.1
M66BYV	Multiply matrix by vector	20.2
M66CPY	Copy a matrix	20.3
M66DIV	Multiply a matrix by inverse of another	20.4
M66EXP	Exponentiate a matrix	20.5
M66INV	Invert a symplectic matrix	20.6
M66MAK	Build matrix for F_2 polynomial	20.7
M66MPY	Multiply two matrices	20.9
M66MTR	Multiply matrix by transpose of another matrix	20.8
M66NRM	Return norm of a matrix	20.10
M66ONE	Build identity matrix	20.11
M66PRT	Print a matrix	20.12
M66REF	Reflect a symplectic matrix	20.13
M66SCL	Scale a matrix	20.14
M66STA	Determine if matrix is static	20.15
M66SUB	Subtract two matrices	20.16
M66TP	Transpose a matrix	20.17
M66TRM	Multiply matrix by transpose of another	20.18
M66ZRO	Set a matrix to zero	20.19

20.1 Add two Matrices

```
CALL M66ADD(A,B,C)
```

adds two 6×6 matrices: $C \leftarrow A + B$.

A — First summand A (input).

B — Second summand B (input).

C — Sum C (output).

20.2 Multiply Matrix by Vector

```
CALL M66BYV(A,B,C)
```

multiplies a 6×6 matrix A by a vector B: $C \leftarrow AB$.

A — Matrix A (input).

B — Vector B (input).

C — Result vector C (output).

20.3 Copy a Matrix

```
CALL M66CPY(A,B)
```

copies a 6×6 matrix: $B \leftarrow A$.

`A` Matrix A to be copied (input).

`B` Copied matrix B (output).

20.4 "Divide" two Matrices

```
CALL M66DIV(A,B,C,EFLAG)
```

multiplies a general 6×6 matrix by the inverse of another one: $C \leftarrow B^{-1}A$. If B cannot be inverted, `EFLAG` is returned as `.TRUE.`.

`A` Matrix A (input).

`B` Matrix B (input).

`C` Result C (output).

`EFLAG` Logical error flag (output).

20.5 Exponentiate a Matrix

```
CALL M66EXP(A,B,EFLAG)
```

exponentiates a matrix: $M \leftarrow e^{JS}$. The matrix A must be built by `M66MAK`, that is the equation $[F, Z] = AZ$ must be true for some quadratic homogeneous polynomial F.

`A` Input matrix A.

`M` Output matrix B.

`EFLAG` Logical error flag (output).

20.6 Invert a Symplectic Matrix

```
CALL M66INV(A,B)
```

inverts a symplectic 6×6 matrix: $B \leftarrow A^{-1}$.

`A` Symplectic matrix A to be inverted (input).

`B` Inverted matrix B (output).

20.7 Matrix for Poisson Bracket with Second-Order Polynomial

```
CALL M66MAK(F,A)
```

returns the matrix A such that $AZ = [F, Z]$, where F is a quadratic homogeneous polynomial in phase space coordinates.

`F` Second-order polynomial F (input), stored in the order shown in Appendix C.

`A` Result matrix A.

20.8 Multiply Two Matrices

```
CALL M66MTR(A,B,C)
```

multiplies two 6×6 matrices: $C \leftarrow AB$.

`A` First factor A (input).

`B` Second factor B (input).

`C` Product C (output).

20.9 Multiply Matrix by Transpose of Another Matrix

```
CALL M66MPY(A,B,C)
```

multiplies a matrix A by the transpose of another matrix B: $C \leftarrow AB^T$.

`A` First factor A (input).

`B` Second factor B to be transposed (input).

`C` Product C (output).

20.10 Return Norm of a Matrix

```
X = M66NRM(A)
```

Returns the norm of the 6×6 matrix A in X.

`A` Input matrix A.

20.11 Build Identity Matrix

```
CALL M66ONE(A)
```

sets the matrix A to the 6×6 identity matrix.

`A` Matrix A to be set to identity.

20.12 Print a Matrix

```
CALL M66PRT(A,IUNIT)
```

prints the 6×6 matrix A on logical unit IUNIT.

A — Matrix A to be printed (input).

IUNIT — Logical unit number to receive output (input).

20.13 Reflect a Symplectic Matrix

```
CALL M66REF(A,B)
```

reflects the symplectic 6×6 matrix A and returns the result in B. The result is equivalent to reflecting the beam line which produced the transfer matrix A.

A — Symplectic input matrix A.

B — Reflected matrix B (output).

20.14 Scale a Matrix

```
CALL M66SCL(S,A,B)
```

multiplies a 6×6 matrix A by a scalar S: $B \leftarrow SA$.

S — Scalar factor S (input).

A — Matrix factor A (input).

B — Product B (output).

20.15 Determine if a Matrix is Static

```
SFLAG = M66STA(A)
```

sets SFLAG to .TRUE. if the matrix A does not change the particle energy.

A — Input matrix A.

20.16 Subtract two Matrices

```
CALL M66SUB(A,B,C)
```

subtracts two 6×6 matrices: $C \leftarrow A - B$.

A — First term A (input).

B — Second term B (input).

C — Difference C (output).

20.17 Transpose a Matrix

```
CALL M66TP(A,B)
```

transposes a 6×6 matrix: $B \leftarrow A^T$.

`A` Input matrix A.

`B` Transposed matrix B (output).

20.18 Multiply a Matrix by the Transpose of Another One

```
CALL M66TRM(A,B,C)
```

transposes a 6×6 matrix and multiplies it with another one: $C \leftarrow A^T B$.

`A` First factor A to be transposed (input).

`B` Second factor B (input).

`C` Product C (output).

20.19 Set a Matrix to Zero

```
CALL M66ZRO(A)
```

sets a 6×6 matrix to zero: $A \leftarrow 0$.

`A` Matrix A to be set to zero.

Chapter 21. Polynomial Algebra "PA"

The PA module contains utilities for algebra on polynomials of three and six variables. Polynomials are stored in arrays, with the coefficients ordered as in Appendix C.

Table 21.1: Routines in the PA module

Name	Purpose	Section
	General Routines	
PAINIT	Initialize complete package	21.1
PAXIND	Internal routine to compute indexing	-
	Polynomials in three variables	
PA3DIF	Differentiate polynomial	21.2
PA3INI	Initialize	-
	Polynomials in six variables	
PA6ADD	Add two polynomials	21.3
PA6BRK	Poisson bracket of two polynomials	21.4
PA6CLR	Clear a polynomial to zero	21.5
PA6CPY	Copy a polynomial	21.6
PA6DIF	Differentiate polynomial	21.7
PA6INI	Initialize	-
PA6NRM	Find norm of a polynomial	21.8
PA6PRD	Product of two polynomials	21.9
PA6PRT	Print a polynomial	21.10
PA6SCL	Scale a polynomial	21.11
PA6SUB	Subtract two polynomials	21.12
PA6SUM	Scaled sum of two polynomials	21.13
PA6VAL	Value of a polynomial	21.14
PA6XFM	Transform arguments of a polynomial linearly	21.15

21.1 Initialize Polynomial Package

```
CALL PAINIT(N)
```

initializes the polynomial package for a given order N. It sets up ZEBRA banks with various tables.

21.2 Derive Polynomial in Three Variables

```
CALL PA3DIF(F,I,N,G)
```

derives a polynomial in three variables with respect to one variable: $G \leftarrow \partial F/\partial z_i$.

F — Coefficients of the polynomial F (input).

I — Variable i with respect to which to differentiate (input).

N — Order N of the polynomial (input):

$N > 0$: Homogeneous polynomial of order N.
$N < 0$: All orders up to N.

G — Coefficients for the derivative G, of order $N - 1$ (output).

21.3 Add Two Polynomials in Six Variables

```
CALL PA6ADD(F,G,N,H)
```

adds two polynomials in six variables: $H \leftarrow F + G$.

`F` Coefficients for the first term (input).

`G` Coefficients for the second term (input).

`N` Order N of the polynomials (input):

$N > 0$: Homogeneous polynomials of order N.

$N < 0$: All orders up to N.

`H` Coefficients for the sum (output).

21.4 Poisson Bracket of Two Polynomials in Six Variables

```
CALL PA6BRK(F,NF,G,NG,H)
```

Computes the Poisson bracket of two homogeneous polynomials in six variables: $H \leftarrow [F, G]$.

`F` Coefficients for the first operand F (input).

`NF` Order N_f of the polynomial F (input).

`G` Coefficients for the second operand G (input).

`NORD` Order N_g of the polynomial G (input).

`H` Coefficients for the product H of order $N_f + N_g - 2$ (output).

21.5 Clear a Polynomial in Six Variables

```
CALL PA6CLR(F,N)
```

clears a polynomials in six variables: $F \leftarrow 0$.

`F` Coefficients for the result (output).

`N` Order N of the polynomial (input):

$N > 0$: Homogeneous polynomial of order N.

$N < 0$: All orders up to N.

21.6 Copy a Polynomial in Six Variables

```
CALL PA6CPY(F,N,G)
```

copies a polynomials in six variables: $G \leftarrow F$.

F — Coefficients for the source (input).

N — Order N of the polynomial (input):

$N > 0$: Homogeneous polynomial of order N.
$N < 0$: All orders up to N.

G — Coefficients for the target (output).

21.7 Differentiate Polynomial in Six Variables

```
CALL PA6DIF(F,I,N,G)
```

derives a polynomial in six variables with respect to one variable: $G \leftarrow \partial F / \partial z_i$.

F — Coefficients of the polynomial F (input).

I — Variable I with respect to which to differentiate (input).

N — Order N of the polynomial (input):

$N > 0$: Homogeneous polynomial of order N.
$N < 0$: All orders up to N.

G — Coefficients for the derivative G, of order $N - 1$ (output).

21.8 Find Norm of a Polynomial in Six Variables

```
X = PA6NRM(F,N)
```

returns the norm of a polynomials in six variables: $X \leftarrow ||F||$.

F — Coefficients for the polynomial (input).

NF — Order N_f of the polynomial F (input).

The norm of the polynomial is defined as the sum of the absolute values of its coefficients.

21.9 Multiply Two Polynomials in Six Variables

```
CALL PA6PRD(F,NF,G,NG,H)
```

subtracts two homogeneous polynomials in six variables: $H \leftarrow FG$.

F — Coefficients for the first factor (input).

NF — Order N_f of the polynomial F (input).

G — Coefficients for the second factor (input).

NG — Order N_g of the polynomial G (input).

H — Coefficients for the product of order $N_f + N_g$ (output).

21.10 Print a Polynomial in Six Variables

```
CALL PA6PRT(F,N,IUNIT)
```

prints a polynomial in six variables from order one to N.

F — Coefficients for the polynomial (input).

N — Order N of the polynomial (input).

IUNIT — Logical unit to receive output (input).

21.11 Scale a Polynomial in Six Variables

```
CALL PA6SCL(S,F,N,G)
```

scales a polynomial in six variables: $G \leftarrow SF$.

S — Scalar factor (input).

F — Coefficients for polynomial to be scaled (input).

N — Order N of the polynomial (input):

$N > 0$: Homogeneous polynomial of order N.

$N < 0$: All orders up to N.

G — Coefficients for scaled result G (output).

21.12 Subtract Two Polynomials in Six Variables

```
CALL PA6SUB(F,G,N,H)
```

subtracts two polynomials in six variables: $H \leftarrow F - G$.

F — Coefficients for the first term (input).

G — Coefficients for the second term (input).

N — Order N of the polynomials (input):

$N > 0$: Homogeneous polynomials of order N.

$N < 0$: All orders up to N.

H — Coefficients for the difference H (output).

21.13 Scaled Sum of Two Polynomials in Six Variables

```
CALL PA6SUM(S,F,N,G)
```

scales a polynomial in six variables and adds it to another one: $G \leftarrow G + SF$.

S — Scalar factor (input).

F — Coefficients for polynomial to be scaled (input).

N — Order N of the polynomials (input):

$N > 0$: Homogeneous polynomials of order N.
$N < 0$: All orders up to N.

G — Coefficients for the accumulated sum G (output).

21.14 Value of a Polynomial in Six Variables

```
V = PA6VAL(F,N,Z)
```

returns the value of a polynomial in six variables: $V \leftarrow F(Z)$.

F — Coefficients for the polynomial (input).

N — Order N of the polynomial (input).

Z — A phase-space vector (input).

21.15 Transform Arguments of a Polynomial in Six Variables

```
CALL PA6XFM(F,N,A,G)
```

transforms the arguments of a polynomial in six variables by a matrix: $G(Z) \leftarrow F(AZ)$.

F — Coefficients for the polynomial (input).

N — Order N of the polynomial (input).

A — Matrix for transformation (input).

G — Coefficients for the transformed polynomial G (output).

Chapter 22. Plot Module "PL"

The PL module implements plot commands.

Table 22.1: Routines in the PL module

Name	Purpose	Section
PLARWE	Tune plot (tune resonance lines)	22.1
PLCOLI	Enter variable formats, biasses in banks	-
PLCURV	Plot one curve	-
PLDUMP	Dump the plot bank (debugging)	22.2
PLELMA	Return 4×6 element matrix	-
PLGACN	Position curve annotation	-
PLGARW	Code tune plot constraints in Polish notation	22.3
PLGAXN	Return compound vertical axis annotation	-
PLGCMD	Return unpacked PLOT command parameters	-
PLGETN	Get variable names for labels etc.	-
PLGTBS	Return table information such as biasses	-
PLINTP	Interpolate variables plotted against s	22.4
PLMAIN	Plot module entry (PLOT, RESPLOT, SETPLOT)	-
PLPLOT	Plot one complete picture	22.5
PLPREP	Prepare plot banks for routine PLPLOT	22.6
PLPTIT	Prepare picture title	-
PLQCON	Check user constraints on tune plot	22.7
PLSCHM	Plot machine schema (magnet symbols)	-
PLPVAL	Calculate composite plot variables	-

22.1 Make Tune Plot

```
CALL PLARWE (IBK, ACTWIN)
```

Makes a plot of tune resonance lines

IBK — Array containing: the number of superperiods, the number of constraints N, the N constraints: minimum, maximum, step, length L, expression in Polish notation of length L, coded as 100001="+", 100002="−", 100003="∗", 100004="/", 100005=KX, 100006=KY, 100007=KS, all other simple integers.

ACTWIN — Active user window.

22.2 Dump Plot Bank

```
CALL PLDUMP
```

Dumps the contents of the plot bank.

22.3 Tune Plot Constraints

```
CALL PLGARW (MXB, MXC, QC, IR, NTOT, IBK, IERR)
```

Returns tune plot constraints in inverse Polish notation:

MXB	Maximum length of output array.
MXC	Maximum number of constraints.
QC	Constraints as read.
IR	Lower limit, upper limit, step of loop.
NTOT	Length occupied in output array.
IBK	Output array (format: see 22.1).
IERR	Error flag: 0=OK, else wrong expression.

22.4 Interpolate Twiss Variables

```
CALL PLINTP(IEP, NPNT, NMAX, STEP, VMIN, VMAX, IERR)
```

Interpolates variables plotted against *s*.

IEP	Row number of first element.
NPNT	Number of points
NMAX	Maximum number of points that should possibly appear.
STEP	Maximum distance between two successive horizontal values.
VMIN	Minima for 4 axes.
VMAX	Maxima for 4 axes.
IERR	Integer error flag: 0=OK, else wrong.

22.5 Plot one Complete Picture

```
CALL PLPLOT
```

Plots one complete picture entirely from the information contained in the plot bank structure PLMAIN. The format of this structure is given in the routine header of PLPLOT.

22.6 Prepare one Complete Picture

```
CALL PLPREP
```

Fills one complete picture into the plot bank structure PLMAIN. The format of this structure is given in the routine header of PLPLOT.

22.7 Check Tune Plot Constraints

```
CALL PLQCON (KX, KY, KXYS, IBK, ISELCT, IERR)
```

Checks the constraints on the tune resonances to be plotted

KX	Index of K_x in KXYS.
KY	Index of K_y in KXYS.
HXYS	Array containing K_x = KXYS(KX), K_y = KXYS(KY), K_s = KXYS(3).
IBK	Array containing constraints in inverse Polish notation (format: see 22.1).
ISELCT	Output flag: 1 if constraint is passed, else 0.
IERR	Integer error flag: 0=OK, else wrong.

Chapter 23. Print Utilities "PR"

The PR module contains the utilities for printing which may be useful for consistent page layout.

Table 23.1: Routines in the PR module

Name	Purpose	Section
PRLINE	Print line of dashes	23.1
PRPAGE	Print page header	23.2

23.1 Print a Line of Dashes over the Page

```
CALL PRLINE(IUNIT)
```

prints a line of 130 dashes across the page on logical unit IUNIT.

23.2 Print page Header

```
CALL PRPAGE(IUNIT)
```

prints the page header line, with title and version number, on logical unit IUNIT.

Chapter 24. Low-Level Reading Routines "RD"

The RD module contains the low-level decoding routines. These are not designed to be called directly, but to act as subroutines of the DC module.

Table 24.1: Routines in the RD module

Name	Purpose	Section
RDFAIL	Count fatal error and mark place on input line	-
RDFIND	Skip to next character occurring in a string	-
RDFORM	Mark delimiters of formal argument list	-
RDINIT	Initialize RD module	-
RDINT	Read an integer value	-
RDLINE	Read a card image	-
RDLOGC	Read a logical value	-
RDMARK	Mark place on input line	-
RDNUMB	Read a floating point value	-
RDSKIP	Skip characters in a string	-
RDSTAT	Read a complete statement into statement buffer	-
RDSTRG	Read a string value	-
RDTEST	Test and skip a character	-
RDWARN	Count warning message and mark place on input line	-
RDWORD	Read an alphanumeric string	-

Chapter 25. Survey Module "SU"

The SU module performs all actions related to SURVEY. Some routines are provided for use in misalignment operations.

Table 25.1: Routines in the SU module

Name	Purpose	Section
SUANGL	Convert rotation matrix to three angles	-
SUCOPY	Copy displacement and rotation matrix	-
SUELEM	Return displacement and rotation matrix for an element	25.1
SUHEAD	Print survey header	-
SUIDEN	Set displacement and rotation matrix to identity transform	-
SULINE	Return displacement and rotation matrix for a line	25.2
SUMAIN	Switch routine for SU module	-
SUMTRX	Build rotation matrix from three angles	-
SURVEY	Command routine for SURVEY command	-
SUTRAN	Transform displacement and rotation matrix for rotation	25.3
SUTRAK	Advance displacement and rotation matrix through element	-
SUUSER	Return displacement and rotation matrix for user-defined element	25.4

25.1 Displacement and Rotation for an Element

```
CALL SUELEM(ELEN,ALEN,V,W)
```

returns survey information about the current element, pointed at by LCELM.

ELEN Element length, measured as the sagitta (output).

ALEN Element length, measured as the arc length (output).

V Three-dimensional displacement (output).

W Rotation, represented as a 3-dimensional orthogonal matrix (output).

25.2 Displacement and Rotation for a Beam Line

```
CALL SULINE(IPOS,ELEN,ALEN,V,W)
```

returns survey information about the line ending at a given position in the working beam line.

IPOS Position counter in the working beam line, where the line in question ends (input).

ELEN Line length, measured as the sagitta (output).

ALEN Line length, measured as the arc length (output).

V Three-dimensional displacement (output).

W Rotation, represented as a 3-dimensional orthogonal matrix (output).

25.3 Transform Displacement and Rotation with a Rotation

```
CALL SUTRAN(W,V,WE)
```

transforms the rotation and the displacement of the element exit with respect to the element entrance from one reference to another.

`W,V` Rotation matrix W and the displacement vector V for the misalignment of an element.

`WE` Rotation W_e of the exit system with respect to entrance system.

Upon entry to the routine, W and V must be related to the element entrance. Upen exit the routine returns the same quantities related to the element exit:

$$W_2 = W_e^T W_1 W_e, \qquad V_2 = W_e^T V_1.$$

25.4 Displacement and Rotation for User-Defined Elements

```
CALL SUUSER(ELEN,ALEN,V,W)
```

returns survey information about user-defined elements. It should take the element parameters from the element bank, pointed at by `LCELM` (Section 3.1).

`ELEN` Element length, measured as the sagitta (output).

`ALEN` Element length, measured as the arc length (output).

`V` Three-dimensional displacement (output).

`W` Rotation, represented as a 3-dimensional orthogonal matrix (output).

Chapter 26. Formatted write routines "SV"

The SV module contains the command routine for the HELP, SHOW, and SAVE commands. Some of the routines may also be of use for building ad hoc saving commands.

Table 26.1: Routines in the SV module

Name	Purpose	Section
SVATTR	Write one attribute of a command	-
SVBANK	Write complete command	-
SVBEGN	Begin new line on SAVE file	-
SVCONT	Begin continuation line on SAVE file	-
SVDICT	Write complete keyword definition	-
SVDUMP	Write out last line of a command	-
SVEXPR	Write arithmetic expression in encoded form	-
SVHELP	Command routine for HELP command	-
SVINT	Write integer value	-
SVLINE	Write LINE command	-
SVLIST	Write beam line list	-
SVLITT	Write litteral string	-
SVMAIN	Command routine for SAVE command	-
SVNAME	Write alphanumeric string	-
SVPARM	Write PARAMETER command	-
SVREAL	Write real value	-
SVSEQ	Write SEQUENCE command	-
SVSHOW	Command routine for SHOW command	-
SVSUBR	Write SUBROUTINE definition	-
SVSTRG	Write quoted string	-
SVVREF	Write variable reference	-

Chapter 27. Dynamic Table Handler "TB"

The "TB" module contains routines to read and write dynamic tables of virtually unlimited size. On Cray computers the tables reside in the SSD (Solid State Device), on other computers on direct-access disk files. The tables may be read and written as a whole as TFS files [6].

Table 27.1: Routines in the TB module

Name	Purpose	Section
TBBUFF	Book table buffers	-
TBCHCK	Check table pointer for validity	-
TBCLOS	Close a dynamic table	27.1
TBCOL	Find named column in table	27.2
TBCREA	Create new dynamic table	27.3
TBDATA	Decode table field	-
TBDROP	Delete a dynamic table	27.4
TBDUMP	Dump tables to backing store	-
TBFILE	Open direct-access file for table manager	-
TBFORM	Decode table descriptor format	-
TBGDSC	Retrieve table descriptor	27.5
TBGET	Low-level table read	-
TBINIT	Initialize table handler	-
TBINPT	Simulate direct-access read	-
TBLIST	Command routine for TABLE command	-
TBNAME	Decode TFS string	-
TBOPEN	Open existing dynamic table	27.6
TBOUTP	Simulate direct-access write	-
TBPDSC	Add descriptor to table	27.7
TBPUT	Low-level table write	-
TBREAD	Read table buffer	-
TBRTFS	Read dynamic table in ASCII TFS format	27.8
TBSEG	Set table segment	27.9
TBSET	Set table row	27.10
TBWTFS	Write dynamic table in ASCII TFS format	27.11
TBWRIT	Write table buffer	-

Dynamic tables are organized in three dimensions:

column: Identified by name.

row: Identified by number.

segment: Identified by number.

As an example consider the table of lattice functions. Each column contains one given lattice function, Each row refers to one position in the machine, and each segment corresponds to one value of the energy error.

All items in a given column have the same data format, identified by its ZEBRA data type code (Table 2.1). Each table row resides in a ZEBRA buffer bank. At least one buffer bank must be allocated; for better efficiency a larger number may be allocated if possible. However, one can never allocate more buffers than available from the ZEBRA store. If more than one buffer bank is available, these are used in a circular way. Two buffers allow for example to access a table sequentially and to have two consecutive rows in store at any time.

Use of a table requires the following steps:

1. Create the table (TBCREA, Section 27.3).
2. For each segment of the table, set the segment number (TBSEG, Section 27.9).
3. For each row within the segment, set the row number (TBSET, Section 27.10). The user must provide a flag value to TBSET:

   1 (read); The buffer is filled by TBSET, but it will not be written when reused.

   2 (update): The buffer is filled by TBSET, and written to backing store when reused.

   3 (write): The buffer is cleared by TBSET, and written to backing store when reused.

   For correct function of the table manager the value of this flag is important. The safest mode of operation is "update", but it is also the slowest.
4. Close the table to release table buffers (TBCLOS, Section 27.1).

27.1 Close a Dynamic Table

```
CALL TBCLOS(LTAB)
```

closes the table identified by the pointer LTAB. This allows the table buffers to be released.

27.2 Find a Table Column and its Format by Name

```
CALL TBCOL(LTAB,COL,IFORM,IBIAS)
```

searches an open table for a given column name.

LTAB Pointer identifying the table (input).

COL Column name (MCNAM characters, input).

IFORM ZEBRA data type associated with the column format (output).

IBIAS Bias of the column in the buffer banks.

27.3 Create a Dynamic Table

```
CALL TBCREA(TNAM,NSEG,NROW,NCOL,CNAM,ICFRM,NBUF,LTAB)
```

creates a new dynamic table.

TNAM Name to be given to the table (MCNAM characters, input).

NSEG Number of segments to be allocated (input).

NROW Number of rows to be allocated (input).

NCOL Number of rows to be allocated (input).

CNAM Array of column names (NCOL words of MCNAM characters each, input).

ICFRM Array of ZEBRA format codes (NCOL integers, input).

NBUF Minimum number of buffers to be allocated (input). A large number of buffers improves the access speed, but it also occupies more space in the dynamic ZEBRA store.

LTAB Pointer to the created table (output), or zero in case of failure.

27.4 Delete a Dynamic Table

```
CALL TBDROP(LTAB)
```

deletes the dynamic table identified by the pointer LTAB.

27.5 Retrive a Table Descriptor

```
CALL TBGDSC(LTAB,DNAME,IFORM,IVAL,RVAL,SVAL)
```

retrieves a table descriptor value from a table.

LTAB Pointer identifying the table (input).

DNAME Descriptor name (MCNAM characters, input).

IFORM ZEBRA data type for descriptor (output).

IVAL Value of descriptor, if integer (output).

RVAL Value of descriptor, if real (output).

SVAL Value of descriptor, if string (output).

27.6 Open an Existing Dynamic Table

```
CALL TBOPEN(TNAM,NBUF,LTAB)
```

opens a dynamic table for reading or updating.

TNAM Name given to the table at its creation (MCNAM characters, input).

NBUF Minimum number of buffers to be allocated (input). A large number of buffers improves the access speed, but it also occupies more space in the dynamic ZEBRA store.

LTAB Pointer to the opened table (output), or zero in case of failure.

27.7 Add a Table Descriptor

```
CALL TBPDSC(LTAB,DNAME,IFORM,IVAL,RVAL,SVAL)
```

adds a table descriptor value to a table.

LTAB Pointer identifying the table (input).

DNAME Descriptor name (MCNAM characters, input).

IFORM ZEBRA data type for descriptor (input).

IVAL Value of descriptor, if integer (input).

RVAL Value of descriptor, if real (input).

SVAL Value of descriptor, if string (input).

27.8 Read a Table in ASCII TFS Format

```
CALL TBRTFS(TNAM,IUNIT)
```

reads a table in ASCII TFS format and leaves it open for reading.

TNAM A name to be given to the table (MCNAM characters, input).

IUNIT Logical unit number to read from (input).

27.9 Set Table Segment

```
CALL TBSEG(LTAB,ISEG,EFLAG)
```

changes the active segment number for a table. In the same time all buffers become empty; they must be reloaded before using.

LTAB Pointer identifying the table (input).

ISEG Desired segment number (input).

EFLAG Logical error flag (output).

27.10 Set Table Row

```
CALL TBSET(LTAB,IROW,IFLAG,LBUF)
```

gives access to the table buffer for a given row.

LTAB Pointer identifying the table (input).

IROW Desired Row number (input).

IFLAG Mode flag (input): 1: read, 2: update, 3: write. For correct function of the table manager this flag is important, since it controls when to fill or to dump buffers.

LBUF Pointer to the table buffer for the desired row (output).

27.11 Write a Table in ASCII TFS Format

```
CALL TBWTFS(TNAM,IUNIT)
```

opens an existing table and writes it in ASCII TFS format.

TNAM Name given to the table at its creation (MCNAM characters, input).

IUNIT Logical unit number to read from (input).

Chapter 28. Transport Maps "TM"

The TM module contains routines to compute and to operate on Transport maps [2]. A general transfer map represents the transformation of phase space from entrance to exit of an element. A Transport map is given by the Taylor series for the general transfer map, truncated at order 2:

$$z_i^{(2)} = \Delta z_i + \sum_{k=1}^{6} R_{ik} z_k^{(1)} + \sum_{k=1}^{6}\sum_{l=1}^{6} T_{ikl} z_k^{(1)} z_l^{(1)}, \qquad i = 1 \ldots 6.$$

Due to truncation a Transport map is symplectic only in exceptional cases. For algorithms refer to [1] and to the MAD Physicist's Reference Manual.

Table 28.1: Routines in the TM module

Name	Purpose	Section
TMALI1	TRANSPORT map for misalignment at entrance	-
TMALI2	TRANSPORT map for misalignment at exit	-
TMARB	TRANSPORT map for MATRIX	-
TMBB	TRANSPORT map for BEAMBEAM	-
TMBEND	TRANSPORT map for RBEND or SBEND	-
TMCAT	Concatenate to TRANSPORT maps	28.1
TMCLOR	Find closed orbit	28.14
TMCORR	TRANSPORT map for HKICK, VKICK, or KICK	-
TMDERI	Compute total derivative of R matrix w.r.t. energy error	28.2
TMDRF	TRANSPORT map for DRIFT	-
TMFOC	Internal routine for focussing matrix	-
TMFRNG	TRANSPORT map for fringing fields	-
TMFRST	Transfer matrix for one turn w.r.t. given orbit	28.3
TMINV	Invert a TRANSPORT map	28.4
TMMAP	TRANSPORT map for an element	28.5
TMMKSM	Multiply TRANSPORT map by its reflection	28.6
TMMULT	TRANSPORT map for MULTIPOLE	-
TMOCT	TRANSPORT map for OCTUPOLE	-
TMQUAD	TRANSPORT map for QUADRUPOLE	-
TMREFE	Transfer matrix for one turn w.r.t. ideal orbit	28.3
TMREFL	Reflect a TRANSPORT map	28.8
TMRF	TRANSPORT map for RFCAVITY	-
TMSCND	TRANSPORT map for one turn w.r.t. given orbit	28.9
TMSECT	TRANSPORT map for sector bend without fringe fields	-
TMSEP	TRANSPORT map for ELSEPARATOR	-
TMSEXT	TRANSPORT map for SEXTUPOLE	-
TMSOL	TRANSPORT map for SOLENOID	-
TMSROT	TRANSPORT map for SROT	-
TMSYMM	Fill in symmetric terms in T array	28.10
TMSYMP	Symplectify an R matrix	28.11
TMTILT	Conjugate a TRANSPORT map with a rotation around s-axis	28.12
TMTRAK	Refer TRANSPORT map to a given orbit	28.13
TMTURN	TRANSPORT map for one turn w.r.t. closed orbit	28.14
TMUSER	TRANSPORT map for user-defined elements	28.15
TMYROT	TRANSPORT map for YROT	-

28.1 Concatenate two TRANSPORT Maps

```
CALL TMCAT(FSEC,RB,TB,RA,TA,RD,TD)
```

concatenates two TRANSPORT maps.

`FSEC` If `.TRUE.` the second-order terms are also done (input).

`RB,TB` *Second* map in beam line order (input).

`RA,TA` *First* map in beam line order (input).

`RD,TD` Result of composition (output).

A similar routine also considers zero-order terms:

```
CALL TMCAT1(FSEC,EB,RB,TB,EA,RA,TA,ED,RD,TD)
```

concatenates two TRANSPORT maps.

`FSEC` If `.TRUE.` the second-order terms are also done (input).

`EB,RB,TB` *Second* map in beam line order (input).

`EA,RA,TA` *First* map in beam line order (input).

`ED,RD,TD` Result of composition (output).

The `Ex` variables conbtain the kicks.

28.2 Derivative of Transfer Matrix with Respect to $\delta p/p$

```
CALL TMDERI(TT,DISP,RTP)
```

computes the total derivative $dR_{ik}/d\delta = 2\sum_{l=1}^{6} T_{ikl}D_l$ of the first-order transfer matrix with respect to the relative energy error.

`TT` Second-order T array for the map to be derived (input).

`DISP` Dispersion vector D for the initial position (input).

`RT` Total derivative $dR/d\delta$ (output).

28.3 Transfer Matrix with Respect to Given Orbit

```
CALL TMFRST(LSEQ,EFLAG)
```

stores the the transfer matrix for a beam line expansion in `MAPTRN` (Section 3.5). It takes the initial orbit coordinates from `/OPTICO/ ORBITO(6)` (Section 3.6).

`LSEQ` Pointer to a beam line expansion (input),

`EFLAG` Returned as true, if an overflow occurs (output).

This routine uses the control flags in `/STFLAG/` (Section 3.11).

28.4 Invert a TRANSPORT Map

```
CALL TMINV(RS,TS,RD,TD)
```

inverts a TRANSPORT map.

RS,TS — TRANSPORT map to be inverted (input)

RD,TD — Inverted map (output).

28.5 TRANSPORT Map for Single Elements

```
CALL TMMAP(FSEC,FTRK,EL,FMAP)
```

stores the map and the kick for the current element in MAPELM (Section 3.4). It assumes that the routines UTBEAM and UTELEM have been called to set all relevant pointers.

FSEC — If .TRUE. the second-order terms are also done (input).

FTRK — If .TRUE. the orbit in /OPTIC1/ ORBIT(6) (Section 3.7) is advanced through the element (input).

EL — Length of the element in m (output).

FMAP — Logical flag telling whether the element affects the beam (output).

This routine uses the control flags in /STFLAG/ (Section 3.11).

28.6 Add Symmetric Part to TRANSPORT Map

```
CALL TMMKSM(FSEC)
```

completes the accumulated map in MAPTRN (Section 3.5) for a symmetric line. It does this by premultiplying the map by the reflection, obtained by TMREFL.

FSEC — If .TRUE. the second-order terms are also done (input).

Reflecting an element which is not longitudinally symmetric will not produce the desired effect. An accelerating cavity, for example, will become decelerating.

28.7 Transfer Matrix with Respect to Ideal Orbit

```
CALL TMREFE(LSEQ)
```

stores the transfer matrix for a beam line expansion in MAPTRN (Section 3.5). It ignores any effects which may displace the orbit and replaces RF cavities by drifts.

LSEQ — Pointer to the a line expansion input).

28.8 Reflect a TRANSPORT Map

```
CALL TMREFL(RS,TS,RD,TD)
```

reflects a TRANSPORT map, i. e. it calculates the map for its elements taken in reverse order.

`RS,TS` Map to be reflected (input).

`RD,TD` Reflected map (output).

Reflecting an element which is not longitudinally symmetric will not produce the desired effect. An accelerating cavity, for example, will become decelerating.

28.9 TRANSPORT Map with respect to Closed Orbit

```
CALL TMSCND(LSEQ)
```

stores the the TRANSPORT map for a beam line expansion in `MAPTRN` (Section 3.5). It takes the initial orbit coordinates from `/OPTICO/ ORBITO(6)` (Section 3.6).

`LSEQ` Pointer to a beam line expansion (input).

This routine uses the control flags in `/STFLAG/` (Section 3.11).

28.10 Make T Array Symmetric

The T arrays in a TRANSPORT map is symmetric with respect to its second and third indices. To make definition of such an array easier, on may fill in the values T_{ikl} for $1 \leq i \leq 6, 1 \leq k \leq l \leq 6$ only and call

```
CALL CALL TMSYMM(T)
```

to make the array symmetric.

28.11 Symplectify R Matrix

```
CALL TMSYMP(R)
```

Modifies its argument, a 6×6 matrix `R`, by a (hopefully) small amount to make it symplectic.

28.12 Conjugate a TRANSPORT Map with a Rotation around the s-Axis

```
CALL TMTILT(FSEC,TILT,EK,R,T)
```

Is the transformation required for tilted elements. It premultiplies the untilted map and kick with a rotation and postmultiplies it by the inverse rotation.

`FSEC` If `.TRUE.` the second-order terms are also done (input).

`TILT` Rotation angle in *rad*.

`EK` Six-dimensional kick Δz associated with the map (input/output).

`R,T` Map to be transformed (input/output).

28.13 Track Orbit Through a TRANSPORT Map

```
CALL TMTRAK(EK,RE,TE,ORB1,ORB2)
```

advances the orbit through a TRANSPORT map. It also modifies the transfer matrix in MAPELM to become the matrix with respect to the given orbit.

EK Kick in six dimensions (input).

RE,TE TRANSPORT map to be applied (input).

ORB1 Orbit before transformation (input).

ORB2 Orbit after transformation (output).

28.14 Closed Orbit and TRANSPORT Map

```
CALL TMTURN(LSEQ,DELTAT,DELTAP,MFLAG,EFLAG)
```

stores the closed orbit for a beam line expansion in /OPTICO/ ORBITO(6) (Section 3.6) and the TRANSPORT map in MAPTRN (Section 3.5).

LSEQ Pointer to a beam line expansion (input).

DELTAP (Average) relative energy error $\delta E/p_0c$ (input).

MFLAG If .TRUE. the monitor readings are updated (input).

DELTAT Path length difference ct in m (output).

EFLAG Logical flag telling whether the orbit could be found (output).

To improve speed the closed orbit and transport map for each energy error are kept in banks linked to the sequence bank. The first time TMTURN is called for a given energy error, it creates a corresponding bank by calling

```
CALL TMCLOR(LSEQ,DELTAT,DELTAP,SHOW,EFLAG)
```

LSEQ Pointer to the beam line expansion (input), normally LCSEQ (Section 2.10).

DELTAP (Average) relative energy error $\delta E/p_0c$ (input).

SHOW If .TRUE., a log is printed on the ECHO file during closed orbit search (input).

DELTAT Path length difference ct in m (output).

EFLAG If the closed orbit could not be found, this flag becomes true (output).

The banks containing the orbits and maps are dropped if anything changes in the beam line. These routines use the control flags in /STFLAG/ (Section 3.11).

28.15 TRANSPORT Map for User-Defined Element

```
CALL TMUSER(FSEC,EK,RE,TE)
```

returns the TRANSPORT map for an user-defined element. It should take the element parameters from the element bank, pointed at by LCELM (Section 3.1).

FSEC — If .TRUE., the second-order terms must also be returned (input).

EK — Kick in six dimensions (output).

RE,TE — The TRANSPORT map for the element (output).

Chapter 29. "TAPE3" Routines "TP"

The TP module provides utilities for output for the TAPE option of SURVEY and TWISS commands. As these routines are considered as obsolete, they are not documented.

Table 29.1: Routines in the TP module

Name	Purpose	Section
TPELEM	Write data for current element	-
TPHEAD	Write header for TAPE option file	-

Chapter 30. Tracking Module "TR"

The TR module contains the switch and command routines for tracking. These routines are not documented here, as any change may seriously affect tracking speed.

Table 30.1: Routines in the TR module

Name	Purpose	Section
TRBEGN	Command routine for TRACK command	-
TRCLOS	Close machine-independent binary track file	-
TRDSP1	Displace rays due to misalignments at entrance	-
TRDSP2	Displace rays due to misalignments at exit	-
TREND	Command routine for ENDTRACK command	-
TREXEC	Bookkeeping for tracking over many turns	-
TRFILE	Write tracks on machine-independent binary track file	-
TFFLOW	Test for overflow rays	-
TRHEAD	Write header on machine-independent binary track file	-
TRMAIN	Switch routine for TR module	-
TRKILL	Remove rays lost by packing surviving rays	-
TRNOIS	Command routine for NOISE command	-
TRNRES	Reset parameters which were changed due to noise	-
TRNSET	Add noise to parameter values	-
TRPELM	Print rays after an element	-
TRPTRN	Print rays after a turn	-
TRRUN	Command routine for RUN command	-
TRSAVE	Command routine for TSAVE command	-
TRSTRT	Command routine for START command	-
TRTBLE	Save current turn in track table	-
TRTURN	Bookkeeping for tracking over one turn	-

Chapter 31. TRANSPORT Tracking Routines "TT"

The TT module contains routines for tracking by the TRANSPORT method. The algorithms are straight-forward.

Table 31.1: Routines in the TT module

Name	Purpose	Section
TTBB	Track through a BEAMBEAM interaction	-
TTCORR	Track through a HKICK, VKICK, or KICK	-
TTDRF	Track through a DRIFT or similar element	-
TTELEM	Track through an element	31.1
TTMULT	Track through a MULTIPOLE	-
TTOCT	Track through a OCTUPOLE	-
TTQUAD	Track through a QUADRUPOLE	-
TTRF	Track through a RFCAVITY	-
TTSEXT	Track through a SEXTUPOLE	-
TTSOL	Track through a SOLENOID	-
TTSROT	Track through a SROT	-
TTTRAK	Apply TRANSPORT map on a set of rays	31.2
TTUSER	Track through a user-defined element	31.3
TTYROT	Track through a YROT	-

31.1 Track through an Element by TRANSPORT Method

```
CALL TTELEM(ITURN,IORD,ISUP,IPOS,SUML,TRACK,NUMBER,NTRACK)
```

advances a set of rays through the current element. It assumes that UTBEAM and UTELEM have been called to set the relevant pointers, or that equivalent operations have been performed before the call. Most elements are tracked by the TRANSPORT method; however some elements (e. g. thin multipoles) have their ad hoc method.

ITURN Number of current turn, for messages (input).

IORD Not used, present for compatibility with LMELEM.

ISUP Number of current superperiod, for messages (input).

SUML Accumulated length (input/output).

TRACK Rays to be tracked (input/output).

NUMBER Ray numbers for the rays in TRACK (input/output).

NTRACK Number of rays in TRACK.

The orbits are compacted when an orbit is lost. Thus the n^{th} orbit is stored in TRACK(*,i), and its number is found in NUMBER(i).

31.2 Apply TRANSPORT Map to a Set of Rays

```
CALL TTTRAK(D,R,T,TRACK,NTRACK)
```

advances a set of rays through a given TRANSPORT map:

$$z_i^{(2)} = \Delta z_i + \sum_{k=1}^{6} R_{ik} z_k^{(1)} + \sum_{k=1}^{6}\sum_{l=1}^{6} T_{ikl} z_k^{(1)} z_l^{(1)}, \qquad for \quad i = 1 \ldots 6.$$

It has the arguments:

D	Kicks ΔZ (input).
R	First-order terms R (input).
T	Second-order terms T (input).
TRACK	Set of rays (input/output).
NTRACK	Number of rays (input).

The rays are stored as explained in Section 31.1.

31.3 Track through User-Defined Elements

```
CALL TTUSER(EL,TRACK,NTRACK)
```

may be replaced by the user to implement tracking through user-defined elements. It should take the element parameters from the element bank, pointed at by LCELM (Section 3.1).

EL	Element length (output).
TRACK	Set of rays (input/output).
NTRACK	Number of rays (input).

The rays are stored as explained in Section 31.1.

Chapter 32. Twiss Routines "TW"

The TW module contains the switch and command routines for the commands of the Twiss group. These include also derived commands, like IBS. For algorithms refer to the MAD Physicist's Manual.

Table 32.1: Routines in the TW module

Name	Purpose	Section
TWBTGO	Bookkeeping routine for TWISS,-COUPLE	-
TWBTIN	Initial lattice functions for TWISS,-COUPLE	-
TWBTPR	Print lattice function for TWISS,-COUPLE	-
TWBTSV	Save lattice functions in table for TWISS,-COUPLE	-
TWBTTK	Track lattice functions for TWISS,-COUPLE	-
TWBTTP	Write lattice functions on file for TWISS,-COUPLE	-
TWCHGO	Bookkeeping routine for TWISS,CHROM	-
TWCHPR	Print lattice functions for TWISS,CHROM	-
TWCHTP	Write lattice functions on file for TWISS,CHROM	-
TWCLOG	Compute Coulomb logarithm for IBS	-
TWCPGO	Bookkeeping routine for TWISS,COUPLE	-
TWCPIN	Initial lattice functions for TWISS,COUPLE	-
TWCPPR	Print lattice functions for TWISS,COUPLE	-
TWCPTK	Track lattice functions for TWISS,COUPLE	-
TWDISP	Track dispersion	-
TWFILL	Fill in given initial values	-
TWIBS	Command routine for IBS	-
TWISS	Command routine for TWISS	-
TWMAIN	Switch routine for TW module	-
TWOPGO	Bookkeeping routine for OPTICS	-
TWOPSV	Save lattice functions in table for OPTICS	-
TWOPTC	Command routine for OPTICS	-
TBSBET	Command routine for SAVEBETA	-
TWSINT	Bjorken/Mtingwa integrals	-
TWSMSV	Save TWISS summary data	-
TWSUMM	Compute TWISS summary data	-

Chapter 33. Utility Routines "UT"

The utilities in the UT module include routine to store data into and to retrieve data from banks, and pattern matching routines.

Table 33.1: Routines in the UT module

Name	Purpose	Section
UTBEAM	Retrieve current working beam line and range	33.1
UTCLRC	Clear occurrence counts in data bank directory	33.2
UTDASH	Internal routine for UTPATT	-
UTELEM	Retrieve data and pointers for current element	33.3
UTGFLT	Retrieve floating-point data	33.4.1
UTGINT	Retrieve integer data	33.4.2
UTGLOG	Retrieve logical data	33.4.3
UTGNAM	Retrieve alphanumeric data	33.4.4
UTGPOS	Retrieve position reference	33.6
UTGRNG	Retrieve range reference	33.6
UTGSTR	Retrieve string data	33.4.5
UTGTYP	Retrieve MAD data types	33.5
UTLENG	Find last non-blank character in a string	33.7
UTLOOK	Search for a name in a table, accepting abbreviations	33.8
UTMTCH	Internal routine for UTMTPT	-
UTMTPT	Match name to pattern	33.10
UTOCNM	Generate unique name from occurrence counter	33.9
UTPATT	Built pattern for pattern matcher	33.10
UTPFLT	Store floating-point data	33.11.1
UTPINT	Store integer data	33.11.2
UTPLOG	Store logical data	33.11.3
UTPNAM	Store alphanumeric data	33.11.4

33.1 Retrieve Current Working Beam Line and Range

```
CALL UTBEAM(LSEQ,IRG1,IRG2,SYMM,NSUP,LINNAM,RNGNAM)
```

retrieves the data for the current working range. The structure of the main beam line expansion is explained in Section 2.10.

LSEQ Pointer to a beam line expansion (input).

IRG1 Start position of the range set by the latest USE command (output).

IRG2 End position of the range set by the latest USE command (output).

SYMM Symmetry flag from the latest USE command (output).

NSUP Number of superperiods from the latest USE command (output).

LINNAM Name of the expanded beam line (output).

RNGNAM Range specification in printable form (output).

33.2 Clear Occurrence Counters In Data Bank Directory

```
CALL UTCLRC
```

is used internally by MAD to clears all occurrence counters in the data bank directory.

33.3 Retrieve Data and Pointers for Current Element

```
CALL UTELEM(LSEQ,IPOS,IFLAG,ELMNAM,IOCC,IENUM)
```

LSEQ	Pointer to a beam line expansion (input).
IPOS	Position number (IRG1 $\leq$ IPOS $\leq$ IRG2, output).
IFLAG	Flag word from the *position flags bank* (output, Table 2.18).
ELMNAM	Element name (output).
IOCC	Occurrence number for the element or line ELMNAM (output).
IENUM	The physical element counter for this position (output). Starts at one for the first element of the line (not of the range), incremented for each element encountered.

The subroutine UTELEM also sets the following reference pointers in common block REFER:

LCELM	Current element.
LCALI	Misalignments for the current element (see Section 2.10.2). This pointer is zero, if there is no misalignment.
LCFLD	Field errors for the current element (see Section 2.10.2). This pointer is zero, if there is no field error.
LCCOM	Information specific to the current orbit corrector or monitor (see Section 2.10.3). This pointer is zero, if the current element is not a corrector or monitor, or if no orbit correction has been made yet.

As an example take a routine which lists all physical elements:

```
      SUBROUTINE LIST
+CALL IMPLICIT
+CALL SEQFLAG           required for MCODE
+CALL REFER             required for LCSEQ, LCALI, and LCFLD
      CHARACTER*(MCNAM) ELMNAM, LINNAM
      CHARACTER*40      RNGNAM
      LOGICAL           SYMM

      CALL UTBEAM(LCSEQ, IPOS1, IPOS2, SYMM, NSUP, LINNAM, RNGNAM)
      WRITE (IFILE, 910) LINNAM, RNGNAM

      DO 90 IPOS = IPOS1, IPOS2
        CALL UTELEM(LCSEQ, IPOS, IFLAG, ELMNAM, IOCC, IENUM)
        ICODE = JBYT(IFLAG,1,MCODE)
        IF (ICODE .EQ. 1) THEN
```

```
          WRITE (IFILE, 920) IENUM, IOCC, ELMNAM
          IF (LCALI .NE. 0) WRITE (IFILE, 930)
          IF (LCFLD .NE. 0) WRITE (IFILE, 940)
        ENDIF
   90 CONTINUE

  910 FORMAT(' Listing line: ',A,' Range: ',A)
  920 FORMAT(' Physical element number ',I5,' is occurrence ',I5,
     +       ' of name ',A)
  930 FORMAT(' This element is misaligned.')
  940 FORMAT(' This element has field errors.')
      END
```

33.4 Fetch Attributes from Command or Definition Banks

The routines in this section all retrieve data without requiring detailed knowledge of the precise bank structure. They all have three input arguments:

`LBANK` Pointer to a command or definition bank (input).

`I1,I2` Range of attributes to be retrieved (input).

`V` Vector with dimension `(I2-I1+1)` (output).

of the proper type. For $I1 \leq i \leq I2$, the routines copies attribute i into `V(`i`-I1+1)`, if it has the expected type and has been set. Otherwise `V(`i`-I1+1)` remains unchanged. One may thus fill `V` with default values prior to calling a routine. Refer to the code for command routines for examples.

33.4.1 Real or Deferred Values

```
CALL UTGFLT(LBANK,I1,I2,V)
```

The output vector `V` is of type `REAL` (single precision version) or `DOUBLE PRECISION` (double precision version). If an attribute is a deferred expression, `UTGFLT` generates a new value by calling `EXEVAL`.

33.4.2 Integer Values

```
CALL UTGINT(LBANK,I1,I2,V)
```

The output vector `V` is of type `INTEGER`.

33.4.3 Logical Values

```
CALL UTGLOG(LBANK,I1,I2,V)
```

The output vector `V` is of type `LOGICAL`.

33.4.4 Name Values

```
CALL UTGNMT(LBANK,I1,I2,V)
```

The output vector `V` is of type `CHARACTER*(MCNAM)`.

33.4.5 String Values

```
CALL UTGSTR(LBANK,I1,I2,V)
```

The output vector V is of type CHARACTER*(MCSTR).

33.5 Data Type Flags

```
CALL UTGTYP(LBANK,ITYPE)
```

returns an array of MAD data types for the attributes of a command.

LBANK — Pointer to the command bank (input).

ITYPE — INTEGER array which will contain the MAD data type of each attribute (if the attribute was entered) or zero (if not entered). ITYPE must be dimensioned at least by the number of command attributes.

33.6 Ranges and Positions

```
CALL UTGRNG(LRNG,LSEQ,IRG1,IRG2,EFLAG)
```

returns the positions for the end points of a range.

LRNG — Pointer to a range reference bank (Table 2.11, input).

LSEQ — Pointer to a beam line expansion (Section 2.10, input).

IRG1,IRG2 — Start and end positions of the range (output).

EFLAG — Logical error flag (output).

The two positions are searched for by two calls to

```
CALL UTGPOS(LRNG,LSEQ,IEND,IPOS,EFLAG)
```

LRNG — Pointer to the range reference bank (input).

LSEQ — Pointer to a beam line expansion (input).

IEND — Offset in the range reference bank: 0 for start, 3 for end (input).

IPOS — Position found (output).

EFLAG — Logical error flag (output).

33.7 Find Last Non-Blank Character in a Name

```
CALL UTLENG(WORD,LENG)
```

takes a character variable WORD of any length and returns in LENG the position of its last non-blank character.

33.8 Look up Name in a Table

```
PARAMETER (NDICT = ...)
CHARACTER*(MCNAM) DICT(NDICT)
CALL UTLOOK(WORD,DICT,NDICT,IDICT)
```

looks up a character variable WORD of any length in the dictionary vector DICT. If found, IDICT is set to the position, otherwise it is set to zero. The routine accepts abbreviations, provided they have at least characters and are unique:

```
PARAMETER (NDICT = ...)
CHARACTER*(MCNAM) DICT(NDICT)
CHARACTER*(MCNAM) WORD
CALL UTLENG(WORD,LENG)
CALL UTLOOK(WORD(1:LENG),DICT,NDICT,IDICT)
```

finds WORD='TI' in the dictionary, if there is exactly one dictionary entry beginning with 'TI'.

33.9 Generate Unique Name

```
CALL UTOCNM(OLD,IOCC,NEW)
```

constructs a unique name of the form "xxxxxxxx[iiiiii]", where "xxxxxxxx" are the first eight characters of the character variable OLD, and "iiiiii" is the encoded value of the integer IOCC. The result is returned in NEW, which must be of type CHARACTER*(16).

33.10 Pattern Matching

Commands which use pattern matching (SAVE and REMOVE) should first convert the pattern to an internal data structure:

```
CALL UTPATT(PATT,LPATT)
```

PATT — Pattern string (input).

LPATT — Pointer to the generated data structure (output). This pointer should reside in a structural link area.

The test whether a name matches the pattern is then made by

```
CALL UTMTPT(LPATT,NAME,FOUND)
```

LPATT — Pointer returned by UTPATT (input).

NAME — Name to be matched (input).

FOUND — Logical flag (output is .TRUE. on success).

33.11 Store Attributes in Command or Definition Banks

The routines in this section all store data without requiring detailed knowledge of the precise bank structure.

`LBANK` Pointer to a command or definition bank (input).

`I1,I2` Range of attributes to be stored (input).

`V` Vector with dimension `(I2-I1+1)` of the proper type (output).

For $I1 \leq i \leq I2$ the value `V(`i`-I1+1)` is copied into the i^{th} attribute of the bank, provided this attribute has the expected type, and the attribute is marked as set. Refer to the code for command routines for examples.

33.11.1 Real Values

```
CALL UTPFLT(LBANK,I1,I2,V)
```

The input vector `V` is of type `REAL` (single precision version) or `DOUBLE PRECISION` (double precision version). If an attribute stored was defined previously by an expression, the expression is erased.

33.11.2 Integer Values

```
CALL UTPINT(LBANK,I1,I2,V)
```

The input vector `V` is of type `INTEGER`.

33.11.3 Logical Values

```
CALL UTPLOG(LBANK,I1,I2,V)
```

The input vector `V` is of type `LOGICAL`.

33.11.4 Name Values

```
CALL UTPNAM(LBANK,I1,I2,V)
```

The input vector `V` is of type `CHARACTER*(MCNAM)`.

Chapter 34. Miscellaneous Routines in MAD

Table 34.1: Miscellaneous Utilities in MAD

Name	Purpose	Section
	Random Generators:	
FRNDM	Uniform distribution in (0,1)	-
GRNDM	Gaussian distribution with $\sigma = 1$	-
INIT55	Initialize random generators	-
IRNDM	Integer in range (1,999999999)	-
IRNGEN	Get next number in sequence	-
	Linear Algebra and Minimization:	
FDJAC2	Internal routine for LMDIF	-
HQR2	Eigenvalues of a real general matrix	-
HTLSQ	Least squares fit by Householder transforms	-
LMDIF	Minimization by LMDIF method	-
LMPAR	Internal routine for LMDIF	-
ORTHES	Transform matrix to Hessenberg form by Householder transforms	-
ORTRAN	Accumulate several Householder transforms	-
QRFAC	Factorize by QR method	-
QRSOLV	Solve linear equations after QR factorization	-
SOLVER	Solve linear equations	-
SYMEIG	Eigenvalues of real symmetric matrix	-
SYMSOL	Solve linear equations with symmetric matrix	-
VDOT	Dot product of two vectors	-
VMOD	Modulus of a vector	-
	Other Routines:	
CHINIT	Initialize character code table, called by AAINIT	-
ERRF	Complex error function, used for BEAMBEAM element	-
FACTOR	Factorial function	-
	User-Replaceable Routines:	
USERCM	User-defined commands	-
USERDF	User-defined definitions	-
USER0	User-defined random generator, 0 arguments	-
USER1	User-defined random generator, 1 argument	-
USER2	User-defined random generator, 2 arguments	-
	ZEBRA Services, see ZEBRA manual:	
ZABEND	Abnormal end of program	-
ZEND	Normal end of program	-
ZTELUS	User-defined program error exit	-

Part III

CERN Library Routines Called by MAD

Chapter 35. ZEBRA Routines

Table 35.1: ZEBRA Routines Called by MAD

Name	Purpose	Section
DZSHOW	Dump ZEBRA bank	35.1
FZENDI	End ZEBRA input	35.2
FZENDO	End ZEBRA output	35.3
FZFILE	Declare file to ZEBRA	35.4
FZIN	ZEBRA input	35.5
FZOUT	ZEBRA output	35.6
LZFIND	Search linear list for word	35.7
LZLAST	Find last bank in linear list	35.8
LZLONG	Search linear list for word string	35.9
MZBOOK	Book new bank	35.10
MZCOPY	Copy bank or structure	35.11
MZDROP	Drop bank or structure	35.12
MZEBRA	Initialize ZEBRA system	35.13
MZEND	End of ZEBRA processing	35.14
MZFLAG	Flag bank or structure	35.15
MZGARB	Garbage collection	35.16
MZLINK	Declare link area	35.17
MZNEED	Test for available space	35.18
MZPUSH	Change bank size	35.19
MZSTOR	Initialize ZEBRA store	35.20
MZVERS	Print ZEBRA version	35.21
MZWIPE	Wipe out division	35.22
MZWORK	Allocate working space	35.23
NZBANK	Find number of banks in sinear list	35.24
ZFATAL	Fatal termination	35.25
ZFATAM	Fatal termination with message	35.26
ZPHASE	Switch processing phase	35.27
ZSHUNT	Change bank linkage	35.28
ZTOPSY	Reverse order of linear list	35.29

35.1 Dump ZEBRA Bank

```
CALL DZSHOW(CHTEXT,ISTOR,LBANK,CHOPT,ILINK1,ILINK2,IDATA1,IDATA2)
```

Displays the contents of a bank or a data structure. The output format is controlled by the I/O characteristics of the structure.

CHTEXT Character variable printed to identify the output.

ISTOR Index of the store containing the bank or data structure.

LBANK Pointer to the bank or structure to be displayed.

CHOPT Option string. A selection of the following:

'B' Print the single bank at LBANK (default).

'D' Print the bank contents down (in five columns).

'S' Print the bank contents sideways (in lines of 10 elements).

'L' Print the whole linear structure supported by LBANK.

'V' Print the vertical structure supported by LBANK.

'Z' Print the structure in hexadecimal format.

ILINK1,ILINK2 First and last link number to be printed for each bank. When both are zero, all links are displayed.

IDATA1,IDATA2 First and last data word to be printed for each bank. When both are zero, all data words are displayed.

35.2 End ZEBRA Input

```
CALL FZENDI(IUNIT,CHOPT)
```

Ends one or several input files.

IUNIT Logical unit number. If zero, all FZ input files.

CHOPT Option string. A selection of the following:

'0' Switch file to output; needed after positioning by reading.

'Q' Quiet, suppress printing of file statistics.

'R' Final rewind.

'T' Terminate, drop control bank for this file and print statistics.

'U' Unload file.

35.3 End ZEBRA Output

```
CALL FZENDO(IUNIT,CHOPT)
```

Terminates one or several output files.

IUNIT Logical unit number. If zero, all FZ output files.

CHOPT Option string. A selection of the following:

'E' Write end of file (unless done).

'E2' Write end of data (unless done).

'F' Flush the buffer only.

'I' Switch to input, write end of data and rewind, if not yet done. Cancel the output permission.

'Q' Quiet, suppress printing of file statistics.

'R' Rewind, unless done.

'T' Terminate; write end of run, drop control bank for this file and print statistics.

'U' Unload file.

35.4 Declare File to ZEBRA

```
CALL FZFILE(IUNIT,LREC,CHOPT)
```

Declares a file to ZEBRA.

`IUNIT` Logical unit number.

`LREC` Record length. Zero gives system defaults.

`CHOPT` Option string. A selection of the following:

file medium:	default:	Disk.
	'M'	Memory.
	'T'	Magnetic tape.
file format:	default:	Native format.
	'A'	Exchange format, ASCII mapping.
	'X'	Exchange format, binary.
data format:	default for disk or tape files: Same as file format.	
	default for memory: Native format.	
	'N'	Native format.
	'X'	Exchange data format.
processing direction:	default:	Input only.
	'I'	Input enabled.
	'O'	Output enabled.
	'IO'	Input and output enabled.
end of file:	'0'	No hardware file marks.
	'1'	Hardware file mark only for level 2 end of file.
	'2'	Hardware file marks for both level 1 and 2 end of files.
various options:	'R'	Initial rewind.
	'Q'	Quiet, print error messages only.
	'P'	Permissive, enable error returns.

35.5 ZEBRA Input

```
CALL FZIN(IUNIT,IDIV,LSUP,IBIAS,CHOPT,NUH,IUHEAD)
```

Reads a complete data structure from a file.

`IUNIT` Logical unit number.

`IDIV` Index of the division to receive the data structure.

`LSUP,IBIAS` See the description of `MZBOOK` in Section 35.10.

`CHOPT` Option string. Only default options are used in MAD.

`NUH` Size of the user header (not used in MAD).

`IUHEAD` The user header (not used in MAD).

35.6 ZEBRA Output

```
CALL FZOUT(IUNIT,IDIV,LENTRY,IEVENT,CHOPT,IOD,NUH,IUHEAD)
```

Writes a data structure to a file.

IUNIT Logical unit number.

IDIV Index of the store or division containing the data structure.

LENTRY Pointer to the data structure to be written.

IEVENT Start-of-event flag; always 1.

CHOPT Option string.

IOD Format descriptor for the user header (not used in MAD).

NUH The size of the user header (not used in MAD).

IUHEAD The user header (not used in MAD).

35.7 Search Linear List for Word

```
L=FUNCTION LZFIND(ISTOR,LSUP,IWORD,M)
```

searches the linear list pointed at by LSUP for the first bank containing IWORD in position M.

35.8 Find Last Bank in Linear List

```
INTEGER FUNCTION LZLAST(ISTOR,LSUP)
```

searches the linear list pointed at by LSUP for its last bank.

35.9 Search Linear List for Word String

```
INTEGER FUNCTION LZLONG(ISTOR,LSUP,N,ITEXT,M)
```

searches the linear list pointed at by LSUP for the first bank containing ITEXT in positions N to N+M-1.

35.10 Book New Bank

```
CALL MZBOOK(IDIV,L,LSUP,IBIAS,IDH,NL,NS,ND,NIO,NZERO)
```

books a new bank.

IDIV The division number, 2 for long-lived banks, 1 for temporary banks.

L Pointer to the new bank (output).

LSUP,IBIAS The new bank will be linked to link IBIAS of the bank at LSUP. If IBIAS is 1, LSUP must be a structural link in a link area which will be made to point at the new bank.

IDH Hollerith name of the bank.

NL Total number of links to be allocated.

NS Number of structural links (out of the total number) to be allocated.

ND Number of data words to be allocated.

NIO ZEBRA data code (Section 2.1) for the bank.

NZERO If zero, the bank is preset to zeros; if positive, the first NZERO words are cleared; if negative, no presetting is done.

35.11 Copy Bank or Structure

```
CALL MZCOPY(IDIV1,LENTRY,IDIV2,LSUP,IBIAS,CHOPT)
```

copies a data structure

IDIV1,LENTRY Structure to be copied resides in division IDIV1 and is pointed at by LENTRY.

IDIV1,LSUP,IBIAS Structure is copied to division IDIV2, and will be linked to link IBIAS of the bank at LSUP.

CHOPT Option string. In MAD this is always 'S' (single bank).

35.12 Drop Bank or Structure

```
CALL MZDROP(ISTOR,L,CHOPT)
```

drops a data structure.

ISTOR Store number. In MAD this is always zero.

L Pointer to the bank to be dropped.

CHOPT Option string. A selection of:

'L'	Drop linear structure.
'V'	Drop only vertical dependents.
default:	Drop bank and its vertical dependents.

35.13 Initialize ZEBRA System

```
CALL MZEBRA(I)
```

Initializes the ZEBRA system.

I The value -2 suppresses logging.

35.14 End of ZEBRA Processing

```
CALL MZEND
```

Prints ZEBRA statistics about the usage of all divisions.

35.15 Flag Bank or Structure

```
CALL MZFLAG(ISTOR,L,IBIT,CHOPT)
```

Sets a status bit in work IQ(L) for all banks in a structure.

ISTOR — Store number. This is always zero in MAD.

L — Pointer to the structure to be flagged.

IBIT — Number of the bit to be set. This may be interrogated for a bank by the function call JBIT(IQ(L),IBIT).

CHOPT — Option string. The same letters are accepted as for MZDROP. Additionally the character 'Z' causes the selected bit to be cleared to zero.

35.16 Garbage Collection

```
CALL MZGARB(IGARB,IWIPE)
```

causes garbage collection in division IGARB and division IWIPE is wiped out.

35.17 Declare Link Area

```
CALL MZLINK(ISTOR,NAME,LAREA,LREF,LREFL)
```

declares a link area (in a COMMON block) to ZEBRA. If this call is omitted, ZEBRA will not update the links in this area in case of garbage collection.

ISTOR — Store number. This is always zero in MAD.

NAME — Name of the link area.

LAREA — First word in the area, also the first link of this area.

LREF — First reference link, if any; otherwise the last structural link.

LREFL — Last reference link, if any; otherwise this parameter is LAREA.

Examples:

```
*    Mixed link area:
     COMMON /LAMIX/ LS1, ..., LSN, LR1, ..., LRN
     CALL MZLINK(0, '/LAMIX/', LS1, LR1, LRN)

*    All structural links:
     COMMON /LASTR/ LS1, ..., LSN
```

```
      CALL MZLINK(0, '/LASTR/', LS1, LSN, LS1)

*     All reference links:
      COMMON /LAREF/ LR1, ..., LRN
      CALL MZLINK(0, '/LAREF/', LR1, LR1, LRN)
```

35.18 Test for Available Space

```
CALL MZNEED(IDIV,NNEED,CHOPT)
```

Tests if the required number of words is available in a division.

`IDIV` Number of the division; 2 for long-lived data, 1 for temporary.

`NNEED` Number of words needed.

`CHOPT` Option string. If there are not enough words, and the option contains `'G'`, garbage collection is performed.

The variable `IQUEST(11)` will contain the number of words which would remain *after allocating* `NNEED` words.

35.19 Change Bank Size

The size of a bank can be increased or decreased by using

```
CALL MZPUSH(ISTOR,L,INCNL,INCND,CHOPT)
```

`ISTOR` Store number. This is always zero in MAD.

`L` The pointer to the bank whose size is to be changed. If the bank must be moved, `L` is updated automatically.

`INCNL` The number of links to be added (positive) or removed (negative). If the bank has only structural links, the new links will be structural as well, otherwise they will be reference links.

`INCND` The number of data words to be added (positive) or removed (negative).

`CHOPT` Option string. A selection of:

`'R'`	No reference links point into the abandoned bank region.
`'I'`	Only the supporting structural link, the link passed in `L`, and the reverse links in the first level dependents point to this bank.
default	Any link may point to this bank.

35.20 Initialize ZEBRA Store

```
      PARAMETER          (MEMMIN = 100 000)
      PARAMETER          (MEMLEN = 500 000)
      COMMON /MEMORY/    FENCE, LQ(MWFLT*MEMLEN)
      SAVE               /MEMORY/
      INTEGER            IQ(MWFLT*MEMLEN)
      REAL               FENCE(2), Q(MWFLT*MEMLEN)
      DIMENSION          DQ(MEMLEN)
      EQUIVALENCE        (IQ(1), Q(1), DQ(1), LQ(9))
      EQUIVALENCE        (LROOT, LQ(1)), (LLUMP, LQ(2))
      ...
      CALL MZSTOR(ISTOR,CHNAME,CHOPT,FENCE,LQ(1),IQ(LR),LQ(LW),
     +            LQ(LIM2),LQ(LAST))
```

is called oncy in the initialization phase. It defines the working store as follows:

`ISTOR` Store number. This is returned as zero in the first call.

`CHNAME` Name of the store for debugging.

`CHOPT` Option string. If this is `'Q'`, the log level for the store is set to minimum logging.

`FENCE` Safety area preceding the store to protect against reference to `LQ(0)`, `LQ(-1)` etc.

`LQ(1)` First word of the dynamic store, first permanent structural link.

`LQ(LR)` MAD uses `IQ` in this position, thus reserving eight structural links, `LQ(1)` through `LQ(8)`. First permanent reference link (no links reserved in MAD).

`LQ(LW)` First word of the working area. MAD uses `IQ` in this position, thus reserving no reference links. The working space is used as explained in Section 1.3.5).

`LQ(LIM2)` Lowest position of the upper limit of division 2, to protect divisions 1 and 2 from being squeezed out of existence by divisions created later. This parameter has no effect in MAD, as no other divisions are ever created.

`LQ(LAST)` Last word of the dynamic store.

35.21 Print ZEBRA Version

```
CALL MZVERS
```

Prints the version of ZEBRA used.

35.22 Wipe Out Division

```
CALL MZWIPE(IDIV)
```

Wipes out all banks contained in division `IDIV`. MAD uses this to wipe out working space allocated in division 1. Note that division 1 is also wiped out by changing the working space limit with a call to `MZWORK`.

35.23 Allocate Working Space

```
CALL MZWORK(ISTOR,DQ(IF),DQ(IL),IFLAG)
```

is used to reserve or release working space. MAD uses this to move the top-of-stack pointer of the working stack. Note that a call to MZWORK also wipes out division 1. The parameters of MZWORK are:

ISTOR — Store number. Always zero in MAD.

DQ(IF) — First word of working space. MAD uses DQ(1) in this position, thus the links reserved by CALL MZSTOR are not changed.

DQ(IL) — Last word of working space.

IFLAG — Flag word with one of the values:

0	Define a new working space,
1	Vary length of the working space, keep links common to the old and new setup intact.
2	Vary last word of working space, don't change first word.
-1	Reset working space to null, i. e. no links and no data words.

35.24 Find Number of Banks in Linear List

```
N=NZBANK(ISTOR,LINK)
```

Returns the number of banks in a linear structure:

ISTOR — Store number. Always zero in MAD.

LINK — Pointer to first bank of the structure.

The variable N will contain the number of banks in the list.

35.25 Fatal Termination

```
CALL ZFATAL
```

is called by ZEBRA when an unrecoverable error occurs. It kills the current job.

35.26 Fatal Termination with Message

```
CALL ZFATAM(MSG)
```

is called by MAD when an unrecoverable error occurs. It kills the current job after printing the message in the character string MSG.

35.27 Switch Processing Phase

```
CALL ZPHASE(IPHASE)
```

Tells ZEBRA in which phase the program is. The meaningful values for IPHASE are:

zero Initializing,

positive Running,

negative Cleaning up.

MAD does not use further conventions.

35.28 Change Bank Linkage

```
CALL ZSHUNT(ISTOR,LSH,LSUP,JBIAS,IFLAG)
```

Changes the linking of a bank:

ISTOR	Store number. Always zero in MAD.
LSH	Pointer to the bank to be relinked.
LSUP	Pointer to the new supporting bank.
JBIAS	Number of the new supporting structural link.
IFLAG	If zero, only a single bank is relinked, otherwise the whole linear structure pointed at by LSH is relinked.

35.29 Reverse Order of Linear List

```
CALL ZTOPSY(ISTOR,LLS)
```

reverses the order of banks in a linear structure, sitting in store ISTOR and pointed at by LLS.

Chapter 36. Other Routines External to MAD

36.1 GX Package, High-Level Plot Routines

The high-level plotting routines used by MAD are listed in table 36.1. They provide an easy interface to the GKS system, and may be replaced for driving a different plotting system. For documentation refer to the GXPLOT Manual [11].

Table 36.1: GXPLOT Routines called by MAD

Name	Purpose	Section
GXASKU	Ask user for plot options	-
GXCLOS	Close terminal work station	-
GXCLRW	Clear open workstations, set picture name	-
GXCUBI	Calculate third-order natural spline	-
GXCUBV	Calculate value of third-order spline	-
GXEOPN	Set logical unit number for a file	-
GXFRM1	Plot a frame with several axes, return windows	-
GXINIT	Initialize plot package	-
GXOPEN	Open terminal workstation	-
GXPL	Plot polyline with clipping	-
GXPLT1	Plot smoothed polyline (spline) with clipping	-
GXPMSW	Plot software marker symbol	-
GXPNBL	Find first and last non-blank in a string	-
GXQAXS	Inquire axis parameters	-
GXQCRV	Inquire curve set parameters	-
GXQRVP	Inquire viewport ratio	-
GXQVAR	Inquire selected variable value	-
GXREST	Restore GKS settings	-
GXSAVE	Save GKS settings	-
GXSAXS	Set axis parameters	-
GXSCRV	Set curve set parameters	-
GXSDEF	Set undefined variables/restore defaults	-
GXSVAR	Set selected variable value	-
GXSVPT	Set workstation viewport	-
GXSWND	Set window	-
GXTERM	Terminate plot package	-
GXTX	Plot text, including greek characters etc.	-
GXWAIT	Wait for user input while diplaying a frame	-

36.2 GKS Plotting Routines

The low-level plotting routines used in the standard distribution of MAD are listed in table 36.2. For documentation refer to the GKS Manual [10].

Table 36.2: GKS Routines Called by MAD

Name	Purpose	Section
GPL	Plot points	-
GSCHH	Set character height	-
GSCHXP	Set character expansion factor	-
GSLN	Set ...	-
GSLWSC	Set ...	-
GSMK	Set mark character	-
GSTXAL	Set text alignment	-

36.3 EPIO Routines, Machine-Independent Binary I/O

The routines for machine-independen binary I/O used by MAD are listed in table 36.3. For documentation refer to the EPIO Manual [13].

Table 36.3: EPIO Routines Called by MAD

Name	Purpose	Section
BL032W	Convert 32 bit to native word	-
CTOIBM	Convert floating point to IBM format	-
EPEND	Terminate of machine-independent I/O	-
EPINIT	Initialize machine-independent I/O	-
EPOUTL	Write a record	-
EPSETW	Set selected control word	-
STOASC	Convert string to ASCII	-

36.4 Miscellaneous Routines

MAD uses a few minor routines from the CERN KERNLIB library. These are listed in table 36.4. For documentation refer to the relevant CERN program library document [4].

Table 36.4: Miscellaneous CERN Library Routines Called by MAD

Package	Name	Purpose	Section
F122	VZERO	Fill an array with zeroes	-
	VBLANK	Fill an array with Hollerith blanks	-
	VFILL	Fill an array with given word	-
M101	SORTZV	Sort array	-
M409	UCTOH	Copy character to Hollerith	-
	UHTOC	Copy Hollerith to character	-
M421	JBIT	Fetch bit from a word	-
	JBYT	Fetch byte from a word	-
	SBIT0	Clear a bit in a word	-
	SBIT1	Set a bit in a word	-
	SBYT	Store a byte in a word	-
V300	UZERO	Clear an array to fill	-
V301	UCOPY	Copy n words	-
V304	IUCOMP	Look up a word in a table	-
Z007	DATIMH	Date and time in Hollerith	-
	TIMEL	Time left for execution	-
	TIMEX	Time used for execution	-
Z044	INTRAC	Identify interactive job	-
Z100	JOBNAM	Retrieve job name	-

Appendix A. An Example of a New Element: Wiggler

Appendix B. An Example of a New Module: Part List

Appendix C. Indexing Scheme for Symbolic Polynomials

The polynomial manipulation routines of MAD use a storage scheme for polynomials invented by Giorgelli [9]. The monomials are ordered by their order; and within the order in lexicographical sequence. The tables below list the monomials in this order.

C.1 First-Order Terms

index	variables	index	variables	index	variables	index	variables	index	variables
1	x	2	p_x	3	y	4	p_y	5	t
6	p_t								

C.2 Second-Order Terms

index	variables	index	variables	index	variables	index	variables	index	variables
7	x^2	8	xp_x	9	xy	10	xp_y	11	xt
12	xp_t	13	p_x^2	14	p_xy	15	p_xp_y	16	p_xt
17	p_xp_t	18	y^2	19	yp_y	20	yt	21	yp_t
22	p_y^2	23	p_yt	24	p_yp_t	25	t^2	26	tp_t
27	p_t^2								

C.3 Third-Order Terms

index	variables	index	variables	index	variables	index	variables	index	variables
28	x^3	29	x^2p_x	30	x^2y	31	x^2p_y	32	x^2t
33	x^2p_t	34	xp_x^2	35	xp_xy	36	xp_xp_y	37	xp_xt
38	xp_xp_t	39	xy^2	40	xyp_y	41	xyt	42	xyp_t
43	xp_y^2	44	xp_yt	45	xp_yp_t	46	xt^2	47	xtp_t
48	xp_t^2	49	p_x^3	50	p_x^2y	51	$p_x^2p_y$	52	p_x^2t
53	$p_x^2p_t$	54	p_xy^2	55	p_xyp_y	56	p_xyt	57	p_xyp_t
58	$p_xp_y^2$	59	p_xp_yt	60	$p_xp_yp_t$	61	p_xt^2	62	p_xtp_t
63	$p_xp_t^2$	64	y^3	65	y^2p_y	66	y^2t	67	y^2p_t
68	yp_y^2	69	yp_yt	70	yp_yp_t	71	yt^2	72	ytp_t
73	yp_t^2	74	p_y^3	75	p_y^2t	76	$p_y^2p_t$	77	p_yt^2
78	p_ytp_t	79	$p_yp_t^2$	80	t^3	81	t^2p_t	82	tp_t^2
83	p_t^3								

C.4 Fourth-Order Terms

index	variables	index	variables	index	variables	index	variables	index	variables
84	x^4	85	x^3p_x	86	x^3y	87	x^3p_y	88	x^3t
89	x^3p_t	90	$x^2p_x^2$	91	x^2p_xy	92	$x^2p_xp_y$	93	x^2p_xt
94	$x^2p_xp_t$	95	x^2y^2	96	x^2yp_y	97	x^2yt	98	x^2yp_t
99	$x^2p_y^2$	100	x^2p_yt	101	$x^2p_yp_t$	102	x^2t^2	103	x^2tp_t
104	$x^2p_t^2$	105	xp_x^3	106	xp_x^2y	107	$xp_x^2p_y$	108	xp_x^2t
109	$xp_x^2p_t$	110	xp_xy^2	111	xp_xyp_y	112	xp_xyt	113	xp_xyp_t

114	$xp_xp_y^2$	115	xp_xp_yt	116	$xp_xp_yp_t$	117	xp_xt^2	118	xp_xtp_t
119	$xp_xp_t^2$	120	xy^3	121	xy^2p_y	122	xy^2t	123	xy^2p_t
124	xyp_y^2	125	xyp_yt	126	xyp_yp_t	127	xyt^2	128	$xytp_t$
129	xyp_t^2	130	xp_y^3	131	xp_y^2t	132	$xp_y^2p_t$	133	xp_yt^2
134	xp_ytp_t	135	$xp_yp_t^2$	136	xt^3	137	xt^2p_t	138	xtp_t^2
139	xp_t^3	140	p_x^4	141	p_x^3y	142	$p_x^3p_y$	143	p_x^3t
144	$p_x^3p_t$	145	$p_x^2y^2$	146	$p_x^2yp_y$	147	p_x^2yt	148	$p_x^2yp_t$
149	$p_x^2p_y^2$	150	$p_x^2p_yt$	151	$p_x^2p_yp_t$	152	$p_x^2t^2$	153	$p_x^2tp_t$
154	$p_x^2p_t^2$	155	p_xy^3	156	$p_xy^2p_y$	157	p_xy^2t	158	$p_xy^2p_t$
159	$p_xyp_y^2$	160	p_xyp_yt	161	$p_xyp_yp_t$	162	p_xyt^2	163	p_xytp_t
164	$p_xyp_t^2$	165	$p_xp_y^3$	166	$p_xp_y^2t$	167	$p_xp_y^2p_t$	168	$p_xp_yt^2$
169	$p_xp_ytp_t$	170	$p_xp_yp_t^2$	171	p_xt^3	172	$p_xt^2p_t$	173	$p_xtp_t^2$
174	$p_xp_t^3$	175	y^4	176	y^3p_y	177	y^3t	178	y^3p_t
179	$y^2p_y^2$	180	y^2p_yt	181	$y^2p_yp_t$	182	y^2t^2	183	y^2tp_t
184	$y^2p_t^2$	185	yp_y^3	186	yp_y^2t	187	$yp_y^2p_t$	188	yp_yt^2
189	yp_ytp_t	190	$yp_yp_t^2$	191	yt^3	192	yt^2p_t	193	ytp_t^2
194	yp_t^3	195	p_y^4	196	p_y^3t	197	$p_y^3p_t$	198	$p_y^2t^2$
199	$p_y^2tp_t$	200	$p_y^2p_t^2$	201	p_yt^3	202	$p_yt^2p_t$	203	$p_ytp_t^2$
204	$p_yp_t^3$	205	t^4	206	t^3p_t	207	$t^2p_t^2$	208	tp_t^3
209	p_t^4								

C.5 Fifth-Order Terms

index	variables	index	variables	index	variables	index	variables	index	variables
210	x^5	211	x^4p_x	212	x^4y	213	x^4p_y	214	x^4t
215	x^4p_t	216	$x^3p_x^2$	217	x^3p_xy	218	$x^3p_xp_y$	219	x^3p_xt
220	$x^3p_xp_t$	221	x^3y^2	222	x^3yp_y	223	x^3yt	224	x^3yp_t
225	$x^3p_y^2$	226	x^3p_yt	227	$x^3p_yp_t$	228	x^3t^2	229	x^3tp_t
230	$x^3p_t^2$	231	$x^2p_x^3$	232	$x^2p_x^2y$	233	$x^2p_x^2p_y$	234	$x^2p_x^2t$
235	$x^2p_x^2p_t$	236	$x^2p_xy^2$	237	$x^2p_xyp_y$	238	x^2p_xyt	239	$x^2p_xyp_t$
240	$x^2p_xp_y^2$	241	$x^2p_xp_yt$	242	$x^2p_xp_yp_t$	243	$x^2p_xt^2$	244	$x^2p_xtp_t$
245	$x^2p_xp_t^2$	246	x^2y^3	247	$x^2y^2p_y$	248	x^2y^2t	249	$x^2y^2p_t$
250	$x^2yp_y^2$	251	x^2yp_yt	252	$x^2yp_yp_t$	253	x^2yt^2	254	x^2ytp_t
255	$x^2yp_t^2$	256	$x^2p_y^3$	257	$x^2p_y^2t$	258	$x^2p_y^2p_t$	259	$x^2p_yt^2$
260	$x^2p_ytp_t$	261	$x^2p_yp_t^2$	262	x^2t^3	263	$x^2t^2p_t$	264	$x^2tp_t^2$
265	$x^2p_t^3$	266	xp_x^4	267	xp_x^3y	268	$xp_x^3p_y$	269	xp_x^3t
270	$xp_x^3p_t$	271	$xp_x^2y^2$	272	$xp_x^2yp_y$	273	xp_x^2yt	274	$xp_x^2yp_t$
275	$xp_x^2p_y^2$	276	$xp_x^2p_yt$	277	$xp_x^2p_yp_t$	278	$xp_x^2t^2$	279	$xp_x^2tp_t$
280	$xp_x^2p_t^2$	281	xp_xy^3	282	$xp_xy^2p_y$	283	xp_xy^2t	284	$xp_xy^2p_t$
285	$xp_xyp_y^2$	286	xp_xyp_yt	287	$xp_xyp_yp_t$	288	xp_xyt^2	289	xp_xytp_t
290	$xp_xyp_t^2$	291	$xp_xp_y^3$	292	$xp_xp_y^2t$	293	$xp_xp_y^2p_t$	294	$xp_xp_yt^2$
295	$xp_xp_ytp_t$	296	$xp_xp_yp_t^2$	297	xp_xt^3	298	$xp_xt^2p_t$	299	$xp_xtp_t^2$
300	$xp_xp_t^3$	301	xy^4	302	xy^3p_y	303	xy^3t	304	xy^3p_t
305	$xy^2p_y^2$	306	xy^2p_yt	307	$xy^2p_yp_t$	308	xy^2t^2	309	xy^2tp_t
310	$xy^2p_t^2$	311	xyp_y^3	312	xyp_y^2t	313	$xyp_y^2p_t$	314	xyp_yt^2
315	xyp_ytp_t	316	$xyp_yp_t^2$	317	xyt^3	318	xyt^2p_t	319	$xytp_t^2$
320	xyp_t^3	321	xp_y^4	322	xp_y^3t	323	$xp_y^3p_t$	324	$xp_y^2t^2$
325	$xp_y^2tp_t$	326	$xp_y^2p_t^2$	327	xp_yt^3	328	$xp_yt^2p_t$	329	$xp_ytp_t^2$
330	$xp_yp_t^3$	331	xt^4	332	xt^3p_t	333	$xt^2p_t^2$	334	xtp_t^3
335	xp_t^4	336	p_x^5	337	p_x^4y	338	$p_x^4p_y$	339	p_x^4t

340	$p_x^4 p_t$	341	$p_x^3 y^2$	342	$p_x^3 y p_y$	343	$p_x^3 y t$	344	$p_x^3 y p_t$
345	$p_x^3 p_y^2$	346	$p_x^3 p_y t$	347	$p_x^3 p_y p_t$	348	$p_x^3 t^2$	349	$p_x^3 t p_t$
350	$p_x^3 p_t^2$	351	$p_x^2 y^3$	352	$p_x^2 y^2 p_y$	353	$p_x^2 y^2 t$	354	$p_x^2 y^2 p_t$
355	$p_x^2 y p_y^2$	356	$p_x^2 y p_y t$	357	$p_x^2 y p_y p_t$	358	$p_x^2 y t^2$	359	$p_x^2 y t p_t$
360	$p_x^2 y p_t^2$	361	$p_x^2 p_y^3$	362	$p_x^2 p_y^2 t$	363	$p_x^2 p_y^2 p_t$	364	$p_x^2 p_y t^2$
365	$p_x^2 p_y t p_t$	366	$p_x^2 p_y p_t^2$	367	$p_x^2 t^3$	368	$p_x^2 t^2 p_t$	369	$p_x^2 t p_t^2$
370	$p_x^2 p_t^3$	371	$p_x y^4$	372	$p_x y^3 p_y$	373	$p_x y^3 t$	374	$p_x y^3 p_t$
375	$p_x y^2 p_y^2$	376	$p_x y^2 p_y t$	377	$p_x y^2 p_y p_t$	378	$p_x y^2 t^2$	379	$p_x y^2 t p_t$
380	$p_x y^2 p_t^2$	381	$p_x y p_y^3$	382	$p_x y p_y^2 t$	383	$p_x y p_y^2 p_t$	384	$p_x y p_y t^2$
385	$p_x y p_y t p_t$	386	$p_x y p_y p_t^2$	387	$p_x y t^3$	388	$p_x y t^2 p_t$	389	$p_x y t p_t^2$
390	$p_x y p_t^3$	391	$p_x p_y^4$	392	$p_x p_y^3 t$	393	$p_x p_y^3 p_t$	394	$p_x p_y^2 t^2$
395	$p_x p_y^2 t p_t$	396	$p_x p_y^2 p_t^2$	397	$p_x p_y t^3$	398	$p_x p_y t^2 p_t$	399	$p_x p_y t p_t^2$
400	$p_x p_y p_t^3$	401	$p_x t^4$	402	$p_x t^3 p_t$	403	$p_x t^2 p_t^2$	404	$p_x t p_t^3$
405	$p_x p_t^4$	406	y^5	407	$y^4 p_y$	408	$y^4 t$	409	$y^4 p_t$
410	$y^3 p_y^2$	411	$y^3 p_y t$	412	$y^3 p_y p_t$	413	$y^3 t^2$	414	$y^3 t p_t$
415	$y^3 p_t^2$	416	$y^2 p_y^3$	417	$y^2 p_y^2 t$	418	$y^2 p_y^2 p_t$	419	$y^2 p_y t^2$
420	$y^2 p_y t p_t$	421	$y^2 p_y p_t^2$	422	$y^2 t^3$	423	$y^2 t^2 p_t$	424	$y^2 t p_t^2$
425	$y^2 p_t^3$	426	$y p_y^4$	427	$y p_y^3 t$	428	$y p_y^3 p_t$	429	$y p_y^2 t^2$
430	$y p_y^2 t p_t$	431	$y p_y^2 p_t^2$	432	$y p_y t^3$	433	$y p_y t^2 p_t$	434	$y p_y t p_t^2$
435	$y p_y p_t^3$	436	$y t^4$	437	$y t^3 p_t$	438	$y t^2 p_t^2$	439	$y t p_t^3$
440	$y p_t^4$	441	p_y^5	442	$p_y^4 t$	443	$p_y^4 p_t$	444	$p_y^3 t^2$
445	$p_y^3 t p_t$	446	$p_y^3 p_t^2$	447	$p_y^2 t^3$	448	$p_y^2 t^2 p_t$	449	$p_y^2 t p_t^2$
450	$p_y^2 p_t^3$	451	$p_y t^4$	452	$p_y t^3 p_t$	453	$p_y t^2 p_t^2$	454	$p_y t p_t^3$
455	$p_y p_t^4$	456	t^5	457	$t^4 p_t$	458	$t^3 p_t^2$	459	$t^2 p_t^3$
460	$t p_t^4$	461	p_t^5						

C.6 Sixth-Order Terms

index	variables	index	variables	index	variables	index	variables	index	variables
462	x^6	463	$x^5 p_x$	464	$x^5 y$	465	$x^5 p_y$	466	$x^5 t$
467	$x^5 p_t$	468	$x^4 p_x^2$	469	$x^4 p_x y$	470	$x^4 p_x p_y$	471	$x^4 p_x t$
472	$x^4 p_x p_t$	473	$x^4 y^2$	474	$x^4 y p_y$	475	$x^4 y t$	476	$x^4 y p_t$
477	$x^4 p_y^2$	478	$x^4 p_y t$	479	$x^4 p_y p_t$	480	$x^4 t^2$	481	$x^4 t p_t$
482	$x^4 p_t^2$	483	$x^3 p_x^3$	484	$x^3 p_x^2 y$	485	$x^3 p_x^2 p_y$	486	$x^3 p_x^2 t$
487	$x^3 p_x^2 p_t$	488	$x^3 p_x y^2$	489	$x^3 p_x y p_y$	490	$x^3 p_x y t$	491	$x^3 p_x y p_t$
492	$x^3 p_x p_y^2$	493	$x^3 p_x p_y t$	494	$x^3 p_x p_y p_t$	495	$x^3 p_x t^2$	496	$x^3 p_x t p_t$
497	$x^3 p_x p_t^2$	498	$x^3 y^3$	499	$x^3 y^2 p_y$	500	$x^3 y^2 t$	501	$x^3 y^2 p_t$
502	$x^3 y p_y^2$	503	$x^3 y p_y t$	504	$x^3 y p_y p_t$	505	$x^3 y t^2$	506	$x^3 y t p_t$
507	$x^3 y p_t^2$	508	$x^3 p_y^3$	509	$x^3 p_y^2 t$	510	$x^3 p_y^2 p_t$	511	$x^3 p_y t^2$
512	$x^3 p_y t p_t$	513	$x^3 p_y p_t^2$	514	$x^3 t^3$	515	$x^3 t^2 p_t$	516	$x^3 t p_t^2$
517	$x^3 p_t^3$	518	$x^2 p_x^4$	519	$x^2 p_x^3 y$	520	$x^2 p_x^3 p_y$	521	$x^2 p_x^3 t$
522	$x^2 p_x^3 p_t$	523	$x^2 p_x^2 y^2$	524	$x^2 p_x^2 y p_y$	525	$x^2 p_x^2 y t$	526	$x^2 p_x^2 y p_t$
527	$x^2 p_x^2 p_y^2$	528	$x^2 p_x^2 p_y t$	529	$x^2 p_x^2 p_y p_t$	530	$x^2 p_x^2 t^2$	531	$x^2 p_x^2 t p_t$
532	$x^2 p_x^2 p_t^2$	533	$x^2 p_x y^3$	534	$x^2 p_x y^2 p_y$	535	$x^2 p_x y^2 t$	536	$x^2 p_x y^2 p_t$
537	$x^2 p_x y p_y^2$	538	$x^2 p_x y p_y t$	539	$x^2 p_x y p_y p_t$	540	$x^2 p_x y t^2$	541	$x^2 p_x y t p_t$
542	$x^2 p_x y p_t^2$	543	$x^2 p_x p_y^3$	544	$x^2 p_x p_y^2 t$	545	$x^2 p_x p_y^2 p_t$	546	$x^2 p_x p_y t^2$
547	$x^2 p_x p_y t p_t$	548	$x^2 p_x p_y p_t^2$	549	$x^2 p_x t^3$	550	$x^2 p_x t^2 p_t$	551	$x^2 p_x t p_t^2$
552	$x^2 p_x p_t^3$	553	$x^2 y^4$	554	$x^2 y^3 p_y$	555	$x^2 y^3 t$	556	$x^2 y^3 p_t$
557	$x^2 y^2 p_y^2$	558	$x^2 y^2 p_y t$	559	$x^2 y^2 p_y p_t$	560	$x^2 y^2 t^2$	561	$x^2 y^2 t p_t$
562	$x^2 y^2 p_t^2$	563	$x^2 y p_y^3$	564	$x^2 y p_y^2 t$	565	$x^2 y p_y^2 p_t$	566	$x^2 y p_y t^2$

567	$x^2yp_ytp_t$	568	$x^2yp_yp_t^2$	569	x^2yt^3	570	$x^2yt^2p_t$	571	$x^2ytp_t^2$
572	$x^2yp_t^3$	573	$x^2p_y^4$	574	$x^2p_y^3t$	575	$x^2p_y^3p_t$	576	$x^2p_y^2t^2$
577	$x^2p_y^2tp_t$	578	$x^2p_y^2p_t^2$	579	$x^2p_yt^3$	580	$x^2p_yt^2p_t$	581	$x^2p_ytp_t^2$
582	$x^2p_yp_t^3$	583	x^2t^4	584	$x^2t^3p_t$	585	$x^2t^2p_t^2$	586	$x^2tp_t^3$
587	$x^2p_t^4$	588	xp_x^5	589	xp_x^4y	590	$xp_x^4p_y$	591	xp_x^4t
592	$xp_x^4p_t$	593	$xp_x^3y^2$	594	$xp_x^3yp_y$	595	xp_x^3yt	596	$xp_x^3yp_t$
597	$xp_x^3p_y^2$	598	$xp_x^3p_ytp_t$	599	$xp_x^3p_yp_t$	600	$xp_x^3t^2$	601	$xp_x^3tp_t$
602	$xp_x^3p_t^2$	603	$xp_x^2y^3$	604	$xp_x^2y^2p_y$	605	$xp_x^2y^2t$	606	$xp_x^2y^2p_t$
607	$xp_x^2yp_y^2$	608	$xp_x^2yp_ytp_t$	609	$xp_x^2yp_yp_t$	610	$xp_x^2yt^2$	611	$xp_x^2ytp_t$
612	$xp_x^2yp_t^2$	613	$xp_x^2p_y^3$	614	$xp_x^2p_y^2tp_t$	615	$xp_x^2p_y^2p_t$	616	$xp_x^2p_ytp_t$
617	$xp_x^2p_ytp_t$	618	$xp_x^2p_yp_t^2$	619	$xp_x^2t^3$	620	$xp_x^2t^2p_t$	621	$xp_x^2tp_t^2$
622	$xp_x^2p_t^3$	623	xp_xy^4	624	$xp_xy^3p_y$	625	xp_xy^3t	626	$xp_xy^3p_t$
627	$xp_xy^2p_y^2$	628	$xp_xy^2p_ytp_t$	629	$xp_xy^2p_yp_t$	630	$xp_xy^2t^2$	631	$xp_xy^2tp_t$
632	$xp_xy^2p_t^2$	633	$xp_xyp_y^3$	634	$xp_xyp_y^2tp_t$	635	$xp_xyp_y^2p_t$	636	$xp_xyp_ytp_t$
637	$xp_xyp_ytp_t$	638	$xp_xyp_yp_t^2$	639	xp_xyt^3	640	$xp_xyt^2p_t$	641	$xp_xytp_t^2$
642	$xp_xyp_t^3$	643	$xp_xp_y^4$	644	$xp_xp_y^3tp_t$	645	$xp_xp_y^3p_t$	646	$xp_xp_y^2tp_t$
647	$xp_xp_y^2tp_t$	648	$xp_xp_y^2p_t^2$	649	$xp_xp_ytp_t$	650	$xp_xp_yt^2p_t$	651	$xp_xp_ytp_t^2$
652	$xp_xp_yp_t^3$	653	xp_xt^4	654	$xp_xt^3p_t$	655	$xp_xt^2p_t^2$	656	$xp_xtp_t^3$
657	$xp_xp_t^4$	658	xy^5	659	xy^4p_y	660	xy^4t	661	xy^4p_t
662	$xy^3p_y^2$	663	$xy^3p_ytp_t$	664	$xy^3p_yp_t$	665	xy^3t^2	666	xy^3tp_t
667	$xy^3p_t^2$	668	$xy^2p_y^3$	669	$xy^2p_y^2tp_t$	670	$xy^2p_y^2p_t$	671	$xy^2p_ytp_t$
672	$xy^2p_ytp_t$	673	$xy^2p_yp_t^2$	674	xy^2t^3	675	$xy^2t^2p_t$	676	$xy^2tp_t^2$
677	$xy^2p_t^3$	678	xyp_y^4	679	$xyp_y^3tp_t$	680	$xyp_y^3p_t$	681	$xyp_y^2tp_t$
682	$xyp_y^2tp_t$	683	$xyp_y^2p_t^2$	684	xyp_ytp_t	685	$xyp_yt^2p_t$	686	$xyp_ytp_t^2$
687	$xyp_yp_t^3$	688	xyt^4	689	xyt^3p_t	690	$xyt^2p_t^2$	691	$xytp_t^3$
692	xyp_t^4	693	xp_y^5	694	$xp_y^4tp_t$	695	$xp_y^4p_t$	696	$xp_y^3tp_t$
697	$xp_y^3tp_t$	698	$xp_y^3p_t^2$	699	$xp_y^2tp_t$	700	$xp_y^2t^2p_t$	701	$xp_y^2tp_t^2$
702	$xp_y^2p_t^3$	703	xp_ytp_t	704	$xp_yt^3p_t$	705	$xp_yt^2p_t^2$	706	$xp_ytp_t^3$
707	$xp_yp_t^4$	708	xt^5	709	xt^4p_t	710	$xt^3p_t^2$	711	$xt^2p_t^3$
712	xtp_t^4	713	xp_t^5	714	p_x^6	715	p_x^5y	716	$p_x^5p_y$
717	p_x^5t	718	$p_x^5p_t$	719	$p_x^4y^2$	720	$p_x^4yp_y$	721	p_x^4yt
722	$p_x^4yp_t$	723	$p_x^4p_y^2$	724	$p_x^4p_ytp_t$	725	$p_x^4p_yp_t$	726	$p_x^4t^2$
727	$p_x^4tp_t$	728	$p_x^4p_t^2$	729	$p_x^3y^3$	730	$p_x^3y^2p_y$	731	$p_x^3y^2t$
732	$p_x^3y^2p_t$	733	$p_x^3yp_y^2$	734	$p_x^3yp_ytp_t$	735	$p_x^3yp_yp_t$	736	$p_x^3yt^2$
737	$p_x^3ytp_t$	738	$p_x^3yp_t^2$	739	$p_x^3p_y^3$	740	$p_x^3p_y^2tp_t$	741	$p_x^3p_y^2p_t$
742	$p_x^3p_yt^2$	743	$p_x^3p_ytp_t$	744	$p_x^3p_yp_t^2$	745	$p_x^3t^3$	746	$p_x^3t^2p_t$
747	$p_x^3tp_t^2$	748	$p_x^3p_t^3$	749	$p_x^2y^4$	750	$p_x^2y^3p_y$	751	$p_x^2y^3t$
752	$p_x^2y^3p_t$	753	$p_x^2y^2p_y^2$	754	$p_x^2y^2p_ytp_t$	755	$p_x^2y^2p_yp_t$	756	$p_x^2y^2t^2$
757	$p_x^2y^2tp_t$	758	$p_x^2y^2p_t^2$	759	$p_x^2yp_y^3$	760	$p_x^2yp_y^2tp_t$	761	$p_x^2yp_y^2p_t$
762	$p_x^2yp_yt^2$	763	$p_x^2yp_ytp_t$	764	$p_x^2yp_yp_t^2$	765	$p_x^2yt^3$	766	$p_x^2yt^2p_t$
767	$p_x^2ytp_t^2$	768	$p_x^2yp_t^3$	769	$p_x^2p_y^4$	770	$p_x^2p_y^3tp_t$	771	$p_x^2p_y^3p_t$
772	$p_x^2p_y^2t^2$	773	$p_x^2p_y^2tp_t$	774	$p_x^2p_y^2p_t^2$	775	$p_x^2p_yt^3$	776	$p_x^2p_yt^2p_t$
777	$p_x^2p_ytp_t^2$	778	$p_x^2p_yp_t^3$	779	$p_x^2t^4$	780	$p_x^2t^3p_t$	781	$p_x^2t^2p_t^2$
782	$p_x^2tp_t^3$	783	$p_x^2p_t^4$	784	p_xy^5	785	$p_xy^4p_y$	786	p_xy^4t
787	$p_xy^4p_t$	788	$p_xy^3p_y^2$	789	$p_xy^3p_ytp_t$	790	$p_xy^3p_yp_t$	791	$p_xy^3t^2$
792	$p_xy^3tp_t$	793	$p_xy^3p_t^2$	794	$p_xy^2p_y^3$	795	$p_xy^2p_y^2tp_t$	796	$p_xy^2p_y^2p_t$
797	$p_xy^2p_yt^2$	798	$p_xy^2p_ytp_t$	799	$p_xy^2p_yp_t^2$	800	$p_xy^2t^3$	801	$p_xy^2t^2p_t$
802	$p_xy^2tp_t^2$	803	$p_xy^2p_t^3$	804	$p_xyp_y^4$	805	$p_xyp_y^3tp_t$	806	$p_xyp_y^3p_t$
807	$p_xyp_y^2t^2$	808	$p_xyp_y^2tp_t$	809	$p_xyp_y^2p_t^2$	810	$p_xyp_yt^3$	811	$p_xyp_yt^2p_t$
812	$p_xyp_ytp_t^2$	813	$p_xyp_yp_t^3$	814	p_xyt^4	815	$p_xyt^3p_t$	816	$p_xyt^2p_t^2$

817	$p_x y t p_t^3$	818	$p_x y p_t^4$	819	$p_x p_y^5$	820	$p_x p_y^4 t$	821	$p_x p_y^4 p_t$
822	$p_x p_y^3 t^2$	823	$p_x p_y^3 t p_t$	824	$p_x p_y^3 p_t^2$	825	$p_x p_y^2 t^3$	826	$p_x p_y^2 t^2 p_t$
827	$p_x p_y^2 t p_t^2$	828	$p_x p_y^2 p_t^3$	829	$p_x p_y t^4$	830	$p_x p_y t^3 p_t$	831	$p_x p_y t^2 p_t^2$
832	$p_x p_y t p_t^3$	833	$p_x p_y p_t^4$	834	$p_x t^5$	835	$p_x t^4 p_t$	836	$p_x t^3 p_t^2$
837	$p_x t^2 p_t^3$	838	$p_x t p_t^4$	839	$p_x p_t^5$	840	y^6	841	$y^5 p_y$
842	$y^5 t$	843	$y^5 p_t$	844	$y^4 p_y^2$	845	$y^4 p_y t$	846	$y^4 p_y p_t$
847	$y^4 t^2$	848	$y^4 t p_t$	849	$y^4 p_t^2$	850	$y^3 p_y^3$	851	$y^3 p_y^2 t$
852	$y^3 p_y^2 p_t$	853	$y^3 p_y t^2$	854	$y^3 p_y t p_t$	855	$y^3 p_y p_t^2$	856	$y^3 t^3$
857	$y^3 t^2 p_t$	858	$y^3 t p_t^2$	859	$y^3 p_t^3$	860	$y^2 p_y^4$	861	$y^2 p_y^3 t$
862	$y^2 p_y^3 p_t$	863	$y^2 p_y^2 t^2$	864	$y^2 p_y^2 t p_t$	865	$y^2 p_y^2 p_t^2$	866	$y^2 p_y t^3$
867	$y^2 p_y t^2 p_t$	868	$y^2 p_y t p_t^2$	869	$y^2 p_y p_t^3$	870	$y^2 t^4$	871	$y^2 t^3 p_t$
872	$y^2 t^2 p_t^2$	873	$y^2 t p_t^3$	874	$y^2 p_t^4$	875	$y p_y^5$	876	$y p_y^4 t$
877	$y p_y^4 p_t$	878	$y p_y^3 t^2$	879	$y p_y^3 t p_t$	880	$y p_y^3 p_t^2$	881	$y p_y^2 t^3$
882	$y p_y^2 t^2 p_t$	883	$y p_y^2 t p_t^2$	884	$y p_y^2 p_t^3$	885	$y p_y t^4$	886	$y p_y t^3 p_t$
887	$y p_y t^2 p_t^2$	888	$y p_y t p_t^3$	889	$y p_y p_t^4$	890	$y t^5$	891	$y t^4 p_t$
892	$y t^3 p_t^2$	893	$y t^2 p_t^3$	894	$y t p_t^4$	895	$y p_t^5$	896	p_y^6
897	$p_y^5 t$	898	$p_y^5 p_t$	899	$p_y^4 t^2$	900	$p_y^4 t p_t$	901	$p_y^4 p_t^2$
902	$p_y^3 t^3$	903	$p_y^3 t^2 p_t$	904	$p_y^3 t p_t^2$	905	$p_y^3 p_t^3$	906	$p_y^2 t^4$
907	$p_y^2 t^3 p_t$	908	$p_y^2 t^2 p_t^2$	909	$p_y^2 t p_t^3$	910	$p_y^2 p_t^4$	911	$p_y t^5$
912	$p_y t^4 p_t$	913	$p_y t^3 p_t^2$	914	$p_y t^2 p_t^3$	915	$p_y t p_t^4$	916	$p_y p_t^5$
917	t^6	918	$t^5 p_t$	919	$t^4 p_t^2$	920	$t^3 p_t^3$	921	$t^2 p_t^4$
922	$t p_t^5$	923	p_t^6						

Bibliography

[1] K. L. Brown, *A First- and Second-Order Matrix Theory for the Design of Beam Transport Systems and Charged Particle Spectrometers.* SLAC-75, Revision 3, 1972.

[2] K. L. Brown, D. C. Carey, Ch. Iselin,and F. Rothacker, *TRANSPORT — A Computer Program for Designing Charged Particle Beam Transport Systems.* CERN 73-16, revised as CERN 80-4, CERN, 1980.

[3] R. Brun, J. Zoll, *ZEBRA User Guide*, CERN Program Library Q100.

[4] CERN Program Library, Collection of Short Writeups. Available from the CERN Program Library.

[5] A. Chao. Evaluation of beam distribution parameters in an electron storage ring. *Journal of Applied Physics*, 50:595–598, 1979.

[6] Ph. Defert, Ph. Hofmann, and R. Keyser. *The Table File System, the C Interfaces.* LAW Note 9, CERN, 1989.

[7] M. Donald and D. Schofield. *A User's Guide to the HARMON Program.* LEP Note 420, CERN, 1982.

[8] A. Dragt. *Lectures on Nonlinear Orbit Dynamics, 1981 Summer School on High Energy Particle Accelerators, Fermi National Accelerator Laboratory, July 1981.* American Institute of Physics, 1982.

[9] Giorgelli, *Comp. Phys. Comm.*, **16**, (1979), pg. 331.

[10] *The Graphical Kernel System (GKS).* ISO, Geneva, July 1985. International Standard ISO 7942.

[11] H. Grote. *GXPLOT User's Guide and Reference Manual.* LEP TH Note 57, CERN, 1988.

[12] H. Grote, F. C. Iselin, *The MAD Program (Methodical Accelerator Design) Version 8.10, User's Reference Manual*, CERN/SL/90-13 (AP), Rev. 3.

[13] H. Grote, I. McLaren, EP Standard Format Input/Output Package, CERN Program Library Long Write-Up I101.

[14] H. Grote, F. C. Iselin, *The MAD Program (Methodical Accelerator Design) Version 8.10, Physicist's Reference Manual*, to be published.

[15] F. Ch. Iselin, *Lie Transformations and Transport Equations for Combined-Function Dipoles*, Particle Accelerators, 1985, **17**, 143-155.

[16] F. James, *MINUIT long write-up.* CERN Program Library D506.

[17] L. C. Teng, *Concerning n-Dimensional Coupled Motion.* FN 229, FNAL, 1971.

Index