EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN/SL/98-XX (AP)

The MAD Program
(Methodical Accelerator Design)
Version 9.3
User's Reference Manual

F. Christoph Iselin
with revisions by J.M. Jowett

Abstract

MAD is a tool for charged-particle optics in alternating-gradient accelerators and beam lines. It can handle very large and very small accelerators and solves various problems on such machines. MAD Version 9 is based on the CLASSIC class library, which was started in 1995 by an international collaboration.

The MAD program is under copyright by CERN.

The MAD framework makes it easy to add new features in the form of new `C++` classes. The authors of MAD hope that such classes will also be contributed and documented by others. The contributions of other authors are acknowledged in the relevant chapters.

This manual is constantly being updated and corrected so any printed copy is almost certainly out-of-date. The latest version of this manual and many other resources are available at the MAD Home Page

`http://wwwslap.cern.ch/mad/`

or

`http://home.cern.ch/mad/`

This page provides the channel for reports of bugs in the program, errors in the documentation and other comments.

Geneva, Switzerland
April 14, 2000

Contents

List of Tables

List of Figures

Chapter 1

Conventions

The accelerator and/or beam line to be studied is described as a sequence of beam elements placed sequentially along a reference or design orbit. The global reference orbit, also known as the design orbit (see Figure 1.2) is the path of a charged particle having the central design momentum of the accelerator through idealised magnets with no fringe fields.

1.1 Design or Reference Orbit

The reference orbit consists of a series of straight line segments and circular arcs. It is defined under the assumption that all elements are perfectly aligned along the design orbit. The accompanying tripod of the reference orbit spans a local curvilinear right handed coordinate system (x, y, s). The local s-axis is the tangent to the reference orbit. The two other axes are perpendicular to the reference orbit and are labelled x (in the bend plane) and y (perpendicular to the bend plane).

1.2 Closed Orbit

Due to various errors like misalignment errors, field errors, fringe fields etc., the closed orbit does not coincide with the design orbit. It also changes with the momentum error. The closed orbit is described with respect to the reference orbit, using the local reference system (see Figure 1.1). It is evaluated including any nonlinear effects.

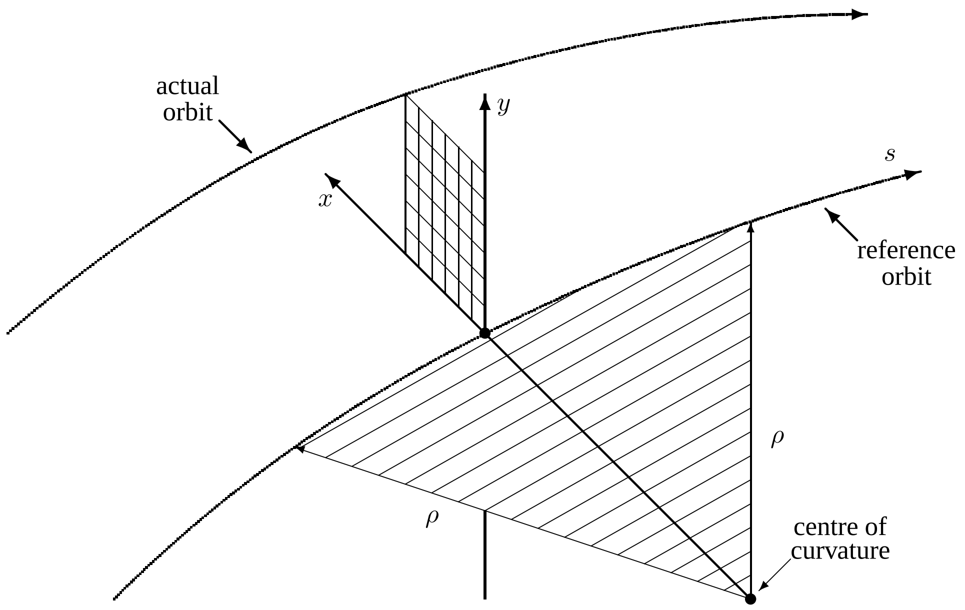

Figure 1.1: Local Reference System

MAD computes the betatron and synchrotron oscillations with respect to the closed orbit. Results are given in the local (x, y, s)-system defined by the reference orbit.

1.3 Global Reference System

The global reference orbit (see Figure 1.2) of the accelerator is uniquely defined by the sequence of physical elements. The local reference system (x, y, s) may thus be referred to a global Cartesian coordinate system (X, Y, Z). The positions between beam elements are numbered $0, \ldots, i, \ldots n$. The local reference system (x_i, y_i, s_i) at position i, i.e. the displacement and direction of the reference orbit with respect to the system (X, Y, Z) are defined by three displacements (X_i, Y_i, Z_i) and three angles $(\Theta_i, \Phi_i, \Psi_i)$.

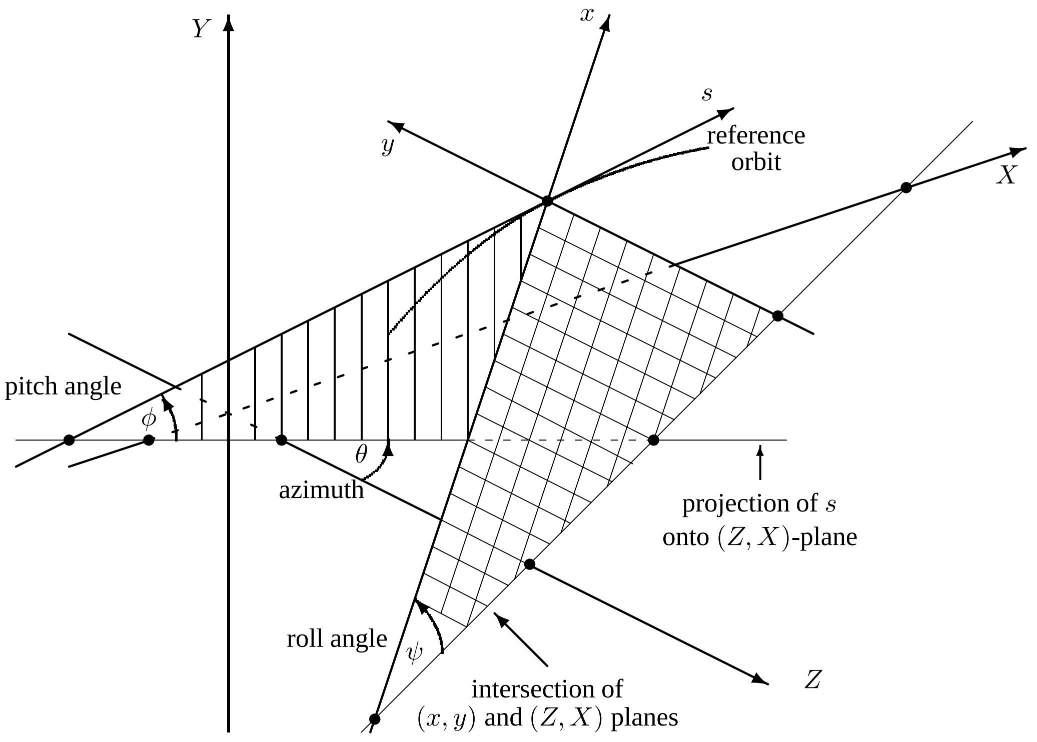
Y
x
s
y
reference orbit
X
pitch angle
ϕ
θ
azimuth
projection of s onto (Z, X)-plane
roll angle
ψ
intersection of (x, y) and (Z, X) planes
Z

Figure 1.2: Global Reference System

The above quantities are defined more precisely as follows:

X Displacement of the local origin in X-direction.

Y Displacement of the local origin in Y-direction.

Z Displacement of the local origin in Z-direction.

THETA Angle of rotation (azimuth) about the global Y-axis, between the global Z-axis and the projection of the reference orbit onto the (Z, X)-plane. A positive angle `THETA` forms a right-hand screw with the Y-axis.

PHI Pitch angle, i.e. the angle between the reference orbit and its projection onto the (Z, X)-plane. A positive angle `PHI` correspond to Y increasing with s. If only horizontal bends are present, the reference orbit remains in the (Z, X)-plane. In this case `PHI` is always zero.

PSI Roll angle about the local s-axis, i.e. the angle between the intersection (x, y)- and (Z, X)-planes and the local x-axis. A positive angle `PSI` forms a right-hand screw with the s-axis.

The angles `(THETA, PHI, PSI)` are **not** the Euler angles. The reference orbit starts at the origin and points by default in the direction of the positive Z-axis. The initial local axes (x, y, s) coincide with the global axes (X, Y, Z) in this order. The six quantities $(X_0, Y_0, Z_0, \Theta_0, \Phi_0, \Psi_0)$ thus all have zero initial values by default. The program user may however specify different initial conditions.

Internally the displacement is described by a vector V and the orientation by a unitary matrix W. The column vectors of W are the unit vectors spanning the local coordinate axes in the order (x, y, s). V and W have the values:

$$V = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, \qquad W = \Theta\Phi\Psi$$

where

$$\Theta = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}, \quad \Phi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}, \quad \Psi = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The reference orbit should be closed and it should not be twisted. This means that the displacement of the local reference system must be periodic with the revolution frequency of the accelerator, while the position angles must be periodic $\pmod{2\pi}$ with the revolution frequency. If `PSI` is not periodic $\pmod{2\pi}$, coupling effects are introduced. When advancing through a beam element, MAD computes V_i and W_i by the recurrence relations

$$V_i = W_{i-1}R_i + V_{i-1}, \qquad W_i = w_{i-1}S_i.$$

The vector R_i is the displacement and the matrix S_i is the rotation of the local reference system at the exit of the element i with respect to the entrance of the same element. The values of R_i and S_i are listed below for each physical element type.

1.4 Local Reference Systems

1.4.1 Reference System for Straight Beam Elements

In straight elements the local reference system (see Figure 1.3) is simply translated by the length of the element along the local s-axis. This is true for

- Drift spaces (see Section 4.6)
- Quadrupoles (see Section 4.8)
- Sextupoles (see Section 4.9)
- Octupoles (see Section 4.10)
- Multipoles (see Section 4.10)
- Solenoids (see Section 4.12)
- RF cavities (see Section 4.14)
- Electrostatic separators (see Section 4.15)
- Closed orbit correctors (see Section 4.13)
- Beam position monitors (see Section 4.16)

The corresponding R, S are

$$R = \begin{pmatrix} 0 \\ 0 \\ L \end{pmatrix}, \qquad S = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

A rotation of the element about the S-axis has no effect on R and S, since the rotations of the reference system before and after the element cancel.

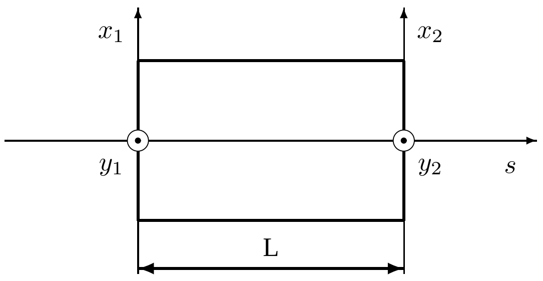

Figure 1.3: Reference System for Straight Beam Elements

1.4.2 Reference System for Bending Magnets

Both rectangular (see Figure 1.4) and sector (see Figure 1.5) bending magnets have a curved reference orbit. For both types of magnets

$$R = \begin{pmatrix} \rho(\cos\alpha - 1) \\ 0 \\ \rho\sin\alpha \end{pmatrix}, \qquad S = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{pmatrix},$$

where α is the bend angle. A positive bend angle represents a bend to the right, i.e. towards negative x values. For sector bending magnets, the bend radius is given by ρ, and for rectangular bending magnets it has the value

$$\rho = L/2\sin(\alpha/2).$$

If the magnet is rotated about the s-axis by an angle psi, R and S are transformed by

$$R^* = TR, \qquad S^* = TST^{-1}.$$

where T is the orthogonal rotation matrix

$$T = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The special value $\psi = \pi/2$ represents a bend down.

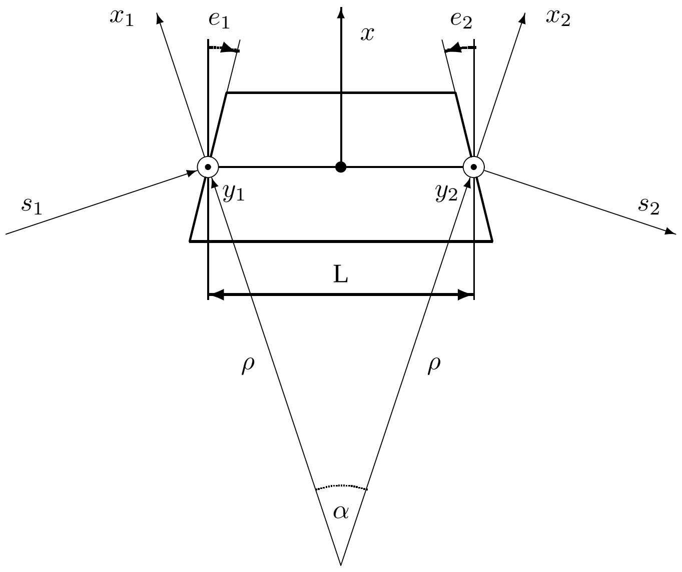

Figure 1.4: Reference System for a Rectangular Bending Magnet; the signs of pole-face rotations are positive as shown.

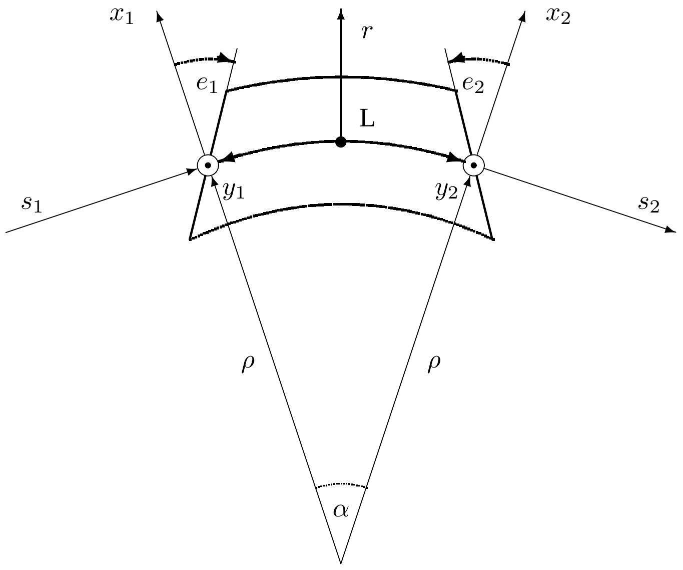

Figure 1.5: Reference System for a Sector Bending Magnet; the signs of pole-face rotations are positive as shown.

1.4.3 Rotation of the Reference System

For a rotation of the reference system by an angle ψ about the beam (s) axis (see Figure 1.6):

$$S = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

while for a rotation of the reference system by an angle θ about the vertical (y) axis (see Figure 1.7):

$$S = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}.$$

In both cases the displacement R is zero.

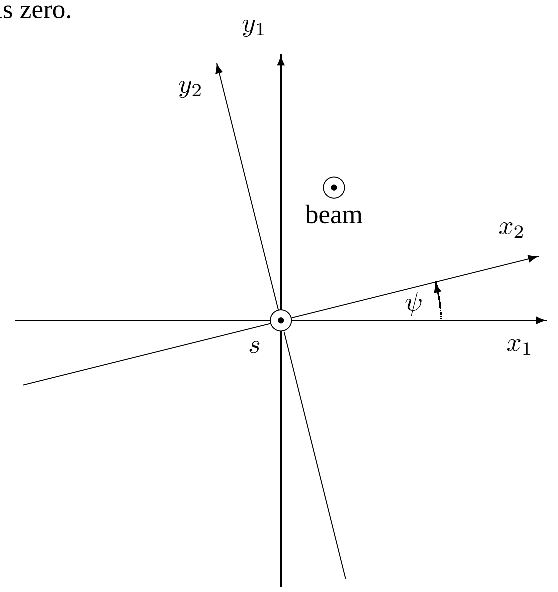

Figure 1.6: Reference System for a Rotation Around the s-Axis

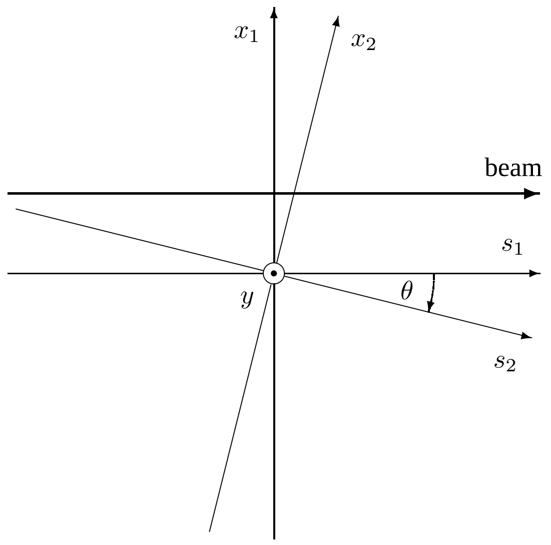

Figure 1.7: Reference System for a Rotation Around the y-Axis

1.4.4 Elements which do not Change the Local Reference

The following elements do not affect the reference orbit and are ignored for geometry calculations:

- Beam-beam interactions (see Section 4.19)
- Marker (see Section 4.5)

1.5 Sign Conventions for Magnetic Fields

The MAD program uses the following Taylor expansion for the normal and skewed field components respectively in the mid-plane y=0, described in [1]:

$$B_y(x,0) = \sum_{k=0}^{\infty} \frac{B_{kn} x^k}{k!}, \qquad B_x(x,0) = \sum_{k=0}^{\infty} \frac{B_{ks} x^k}{k!}.$$

Note the factorial in the denominator. The field coefficients have the following meaning:

B_{0n} Normal dipole field component. The component is positive, if the field points in positive y direction. A positive field bends a positively charged particle travelling in positive s-direction to the right.

B_{0s} Skew dipole field component. The component is positive, if the field points in negative y direction. A positive field bends a positively charged particle travelling in positive s-direction down.

B_{1n} Normal quadrupole field component $B_{1n} = \partial B_y/\partial x$. The component is positive, if B_y is positive on the positive x-axis. A positive value corresponds to horizontal focussing of a positively charged particle.

B_{1s} Skew quadrupole field component $B_{1s} = \partial B_x/\partial x$. The component is positive, if B_x is negative on the positive x-axis.

B_{2n} Normal sextupole field component $B_{2n} = \partial^2 B_y/\partial x^2$. The component is positive, if B_y is positive on the x-axis.

B_{2s} Skew sextupole field component $B_{2s} = \partial^2 B_x/\partial x^2$. The component is negative, if B_x is positive on the x-axis.

B_{3n} Normal octupole field component $B_{3n} = \partial^3 B_y/\partial x^3$. The component is positive, if B_y is positive on the positive x-axis.

B_{3s} Skew octupole field component $B_{3s} = \partial^3 B_x/\partial x^3$. The component is negative, if B_x is positive on the x-axis.

All derivatives are taken on the x-axis. Using this expansion and the curvature h of the reference orbit, the longitudinal component of the vector potential for a magnet with mid-plane symmetry is to order 4:

$$\begin{aligned} A_s = & + B_{0n}\left(x - \frac{hx^2}{2(1+hx)}\right) & + & B_{1n}\left(\frac{1}{2}(x^2-y^2) - \frac{h}{6}x^3 + \frac{h^2}{24}(4x^4-y^4) + \ldots\right) \\ & + B_{2n}\left(\frac{1}{6}(x^3-3xy^2) - \frac{h}{24}(x^4-y^4) + \ldots\right) & + & B_{3n}\left(\frac{1}{24}(x^4-6x^2y^2+y^4) + \ldots\right) + \ldots \end{aligned}$$

Taking $\mathrm{curl}A$ in curvilinear coordinates, the field components can be computed as

$$\begin{aligned} B_x(x,y) = & + B_{1n}\left(y + \frac{h^2}{6}y^3 + \ldots\right) & & \\ & + B_{2n}\left(xy - \frac{h^3}{6}y^3 + \ldots\right) & + & B_{3n}\left(\frac{1}{6}(3x^2y - y^3) + \ldots\right) + \ldots \\ B_y(x,y) = & + B_{0n} & + & B_{1n}\left(x - \frac{h}{2}y^2 + \frac{h^2}{2}xy^2 + \ldots\right) \\ & + B_{2n}\left(\frac{1}{2}(x^2-y^2) - \frac{h}{2}xy^2 + \ldots\right) & + & B_{3n}\left(\frac{1}{6}(x^3 - 3xy^2) + \ldots\right) + \ldots \end{aligned}$$

One can easily verify that both $\mathrm{curl}B$ and $\mathrm{div}B$ are zero to the order of the B_3 term. Introducing the magnetic rigidity $B\rho$, the multipole coefficients are computed as

$$K_{kn} = eB_{kn}/p_0 = B_{kn}/B\rho, \qquad K_{ks} = eB_{ks}/p_0 = B_{ks}/B\rho.$$

Note that the K_k have the **same sign** as the corresponding field components B_k. The signs will be changed due to the sign of particle charges and the direction of travel of the beam.

1.6 Variables

For each variable the physical units are listed in square brackets.

1.6.1 Canonical Variables Describing Orbits

As from Version 9.01, MAD uses the following canonical variables to describe the motion of particles:

X Horizontal position x of the (closed) orbit, referred to the ideal orbit [m].

PX Horizontal canonical momentum of the (closed) orbit referred to the ideal orbit, divided by the reference momentum: `PX` $= p_x/p_0$.

Y Vertical position y of the (closed) orbit, referred to the ideal orbit [m].

PY Vertical canonical momentum of the (closed) orbit referred to the ideal orbit, divided by the reference momentum: `PY` $= p_y/p_0$.

T The negative time difference, multiplied by the instantaneous velocity of the particle [m]: `T` $= -v\delta(t)$. A positive `T` means that the particle arrives ahead of the **reference particle**. `T` describes the deviation of the particle from the orbit of a fictitious reference particle having the constant **reference momentum** p_s and the **reference velocity** v_s. v_s defines the revolution frequency. The velocities have the values

$$v = cp/\sqrt{p^2+m^2c^2}, \qquad v_s = cp_s/\sqrt{p_s^2+m^2c^2},$$

where c is the velocity of light, m is the particle rest mass, and p is the instantaneous momentum of the particle.

PT Momentum error, divided by the reference momentum: `PT` $= \delta p/p_s$. This value is only non-zero when synchrotron motion is present. It describes the deviation of the particle from the orbit of a particle with the reference momentum p_s.

The independent variable is:

S Arc length s along the reference orbit, [m].

The longitudinal variables have been changed with respect to previous versions of MAD, so as to be in line with the CLASSIC project. In the limit of fully relativistic particles ($\gamma \gg 1, v = c, pc = E$), the variables `T, PT` used here agree with the longitudinal variables used in [2]. This means that `T` becomes the negative path length difference, while `PT` is the fractional momentum error. The reference momentum must be constant in order to keep the system canonical.

1.6.2 Normalised Variables and other Derived Quantities

XN The normalised horizontal displacement [$\mathrm{m}^{1/2}$]: $\texttt{XN} = x_n = \Re(E_1^T SZ)$.

PXN The normalised horizontal transverse momentum [$\mathrm{m}^{1/2}$]: $\texttt{PXN} = p_{xn} = \Im(E_1^T SZ)$.

WX The horizontal Courant-Snyder invariant [m]: $\texttt{WX} = \sqrt{x_n^2 + p_{xn}^2}$.

PHIX The horizontal phase: $\texttt{PHIX} = -\arctan(p_{xn}/x_n)/2\pi$.

YN The normalised vertical displacement [$\mathrm{m}^{1/2}$]: $\texttt{YN} = y_n = \Re(E_2^T SZ)$.

PYN The normalised vertical transverse momentum [$\mathrm{m}^{1/2}$]: $\texttt{PYN} = p_{yn} = \Im(E_2^T SZ)$.

WY The vertical Courant-Snyder invariant [m]: $\texttt{WY} = \sqrt{y_n^2 + p_{yn}^2}$.

PHIY The vertical phase: $\texttt{PHIY} = -\arctan(p_{yn}/y_n)/2\pi$.

TN The normalised longitudinal displacement [$\mathrm{m}^{1/2}$]: $\texttt{TN} = t_n = \Re(E_3^T SZ)$.

PTN The normalised longitudinal transverse momentum [$\mathrm{m}^{1/2}$]: $\texttt{PTN} = p_{tn} = Im(E_3^T SZ)$.

WT The longitudinal invariant [m]: $\texttt{WT} = \sqrt{t_n^2 + p_{tn}^2}$.

PHIT The longitudinal phase: $\texttt{PHIT} = +\arctan(p_{tn}/t_n)/2\pi$.

in the above formulas Z is the phase space vector $Z = (x, p_x, y, p_y, t, p_t)^T$. The matrix S is the "symplectic unit matrix"

$$S = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix},$$

and the vectors E_i are the three complex eigenvectors. The superscript T denotes the transpose of a vector or matrix.

1.7 Physical Units

Throughout the computations MAD uses international units (see Table 1.1), as defined by SI (Système International).

quantity	dimension
Length	m (metres)
Angle	rad (radians)
Quadrupole coefficient	m^{-2}
Multipole coefficient, 2n poles	m^{-n}
Electric voltage	MV (Megavolts)
Electric field strength	MV/m
Frequency	MHz (Megahertz)
Phase angles	2π
Particle energy	GeV
Particle mass	GeV/c^2
Particle momentum	GeV/c
Beam current	A (Amperes)
Particle charge	e (elementary charges)
Impedances	MΩ (Megohms)
Emittances	π m mrad
RF power	MW (Megawatts)
Higher mode loss factor	V/pc

Table 1.1: Physical Units

Chapter 2

Command Format

2.1 Statements and Comments

Input to MAD is free format, and the line length is not limited. During reading, input lines are normally printed on the echo file, but this feature can be turned off for long input files. The input is broken up into tokens (words, numbers, delimiters etc.), which form a sequence of commands, also known as statements. Each statement must be terminated by a semicolon (`;`), and long statements can be continued on any number of input lines. White space, like blank lines, spaces, tabs, and newlines are ignored between tokens. Comments can be introduced with two slashes (`//`) and any characters following the slashes on the same line are ignored.

The C convention for comments (`/* ... */`) is also accepted. The comment delimiters `/*` and `*/` can be nested; thus whole section of input can be "commented out".

In the following descriptions, words in `lower case` stand for syntactic units which are to be replaced by actual text. `UPPER CASE` is used for keywords or names. These must be entered as shown. Ellipses (`...`) are used to indicate repetition.

The general format for a command is

```
keyword,attribute,...,attribute;
label:keyword,attribute,...,attribute;
```

It has three parts:

1. The `label` is required for a definition statement. Its must be an identifier (see Section 2.2) and gives a name to the stored command.

2. The `keyword` identifies the action desired. It must be an identifier (see Section 2.2).

3. Each `attribute` is entered in one of the forms

   ```
   attribute-name
   attribute-name=attribute-value
   attribute-name:=attribute-value
   ```

   and serves to define data for the command, where:

   - The `attribute-name` selects the attribute, it must be an identifier (see Section 2.2).
   - The `attribute-value` gives it a value (see Section 2.3). When the attribute value is a constant or an expression preceded by the delimiter `=` it is evaluated immediately and the result is assigned to the attribute as a constant. When the attribute value is an expression preceded by the delimiter `:=` the expression is retained and re-evaluated whenever one of its operands changes.

   Each attribute has a fixed attribute type (see Section 2.3). The `attribute-value` can only be left out for logical attributes, this implies a `true` value.

When a command has a `label`, MAD keeps the command in memory. This allows repeated execution of the same command by entering its label only:

```
label;
```

or to re-execute the command with modified attributes:

```
label,attribute,...,attribute;
```

If the label of such a command appears together with new attributes, MAD makes a copy of the stored command, replaces the attributes entered, and then executes the copy:

```
QF:QUADRUPOLE,L=1,K1=0.01; // first definition of QF
QF,L=2;                    // redefinition of QF, new length

MATCH;
...
LMD:LMDIF,CALLS=10;           // first execution of LMD
LMD;                          // re-execute LMD with the same attributes
LMD,CALLS=100,TOLERANCE=1E-5; // re-execute LMD with new attributes
ENDMATCH;
```

2.2 Identifiers or Labels

An identifier refers to a keyword, an element, a beam line, a variable, an array, etc.

A label begins with a letter, followed by an arbitrary number of letters, digits, periods (`.`), underscores (`_`). Other special characters can be used in a label, but the label must then be enclosed in single or double quotes. It makes no difference which type of quotes is used, as long as the same are used at either end. The preferred form is double quotes. The use of non-numeric characters is however strongly discouraged, since it makes it difficult to subsequently process a MAD output with another program.

When a name is not quoted, it is converted to upper case; the resulting name must be unique. An identifier can also be generated from a string expression (see Section 2.4). Abbreviations are no longer allowed as from Version 9.01.

2.3 Command Attribute Types

An object attribute is referred to by the syntax

```
object-name->attribute-name
```

If the attribute is an array (see Section 2.14), one of its components is found by the syntax

```
object-name->attribute-name[index]
```

The following types of command attributes are available in MAD-9:

- String (see Section 2.4),
- Logical (see Section 2.5),
- Real expression (see Section 2.6),
- Deferred expression (see Section 2.8.5),
- Place (see Section 2.9.1),
- Range (see Section 2.9.2),
- Constraint (see Section 2.10),
- Variable Reference (see Section 2.11)
- Regular expression (see Section 2.12),
- Token list (see Section 2.13).
- Array (see Section 2.14) of

  - Logical (see Section 2.14.1),
  - Real (see Section 2.14.2),
  - String (see Section 2.14.3),
  - Token lists (see Section 2.14.4),

See also:

- Operators (see Table 2.4),
- Functions (see Table 2.5),
- Array functions (see Table 2.6),
- Real functions of arrays (see Table 2.8),
- Operand (see Section 2.8),
- Random generators (see Section 2.8.5).

2.4 String Attributes

A string attribute makes alphanumeric information available, e.g. a title, file name, element class name, or an option. It can contain any characters, enclosed in single (`'`) or double (`"`) quotes. However, if it contains a quote, this character must be doubled. Strings can be concatenated using the & operator (see Table 2.1). An operand in a string can also use the function `STRING` (see Table 2.2). String values can occur in string arrays (see Section 2.14).

Table 2.1: String Operator in MAD-9

Operator	Meaning	result type	operand types
`X & Y`	concatenate the strings `X` and `Y`. String concatenations are always evaluated immediately when read.	string	string,string

Table 2.2: String Function in MAD-9

Function	Meaning	result type	argument type
`STRING(X)`	return string representation of the value of the numeric expression `X`	string	real

Examples:

```
TITLE,"This is a title for the program run ""test""";
CALL,FILE="save";

X=1;
TWISS,LINE=LEP&STRING(X+1);
```

The second example converts the value of the expression “X+1” to a string and appends it to “LEP”, giving the string “LEP2”.

2.5 Logical Expressions

Many commands in MAD require the setting of logical values (flags) to represent the on/off state of an option. A logical value is represented by one of the values `TRUE` or `FALSE`, or by a logical expression. A logical expression can occur in logical arrays (see Section 2.14.1).

A logical expression has the same format and operator precedence as a logical expression in C. It is built from logical operators (see Table 2.3) and logical operands:

```
relation      ::= "TRUE" |
                  "FALSE" |
                  real-expr rel-operator real-expr

rel-operator  ::= "==" | "!=" | "<" | ">" | ">=" | "<="

and-expr      ::= relation | and-expr "&&" relation

logical-expr  ::= and-expr | logical-expr "||" and-expr
```

Table 2.3: Logical Operators in MAD-9

Operator	Meaning	result type	operand type
`X < Y`	true, if X is less than Y	logical	real,real
`X <= Y`	true, if X is not greater than Y	logical	real,real
`X > Y`	true, if X is greater than Y	logical	real,real
`X >= Y`	true, if X is not less than Y	logical	real,real
`X == Y`	true, if X is equal to Y	logical	real,real
`X != Y`	true, if X is not equal to Y	logical	real,real
`X && Y`	true, if both X and Y are true	logical	logical,logical
`X || Y`	true, if at least one of X and Y is true	logical	logical,logical

Example:

```
OPTION,ECHO=TRUE; // output echo is desired
```

When a logical attribute is not entered, its default value is always `false`. When only its name is entered, the value is set to TRUE:

```
OPTION,ECHO;      // same as above
```

Example of a logical expression:

```
X>10 && Y<20 || Z==15
```

2.6 Real Expressions

To facilitate the definition of interdependent quantities, any real value can be entered as an arithmetic expression. When a value used in an expression is redefined by the user or changed in a matching process, the expression is re-evaluated. Expression definitions may be entered in any order. MAD evaluates them in the correct order before it performs any computation. At evaluation time all operands used must have values assigned. A real expression can occur in real arrays (see Section 2.14.2).

A real expression is built from operators (see Table 2.4) and operands (see Section 2.8):

```
real-ref  ::= real-variable |
              real-array "[" index "]" |
              object "->" real-attribute |
              object "->" real-array-attribute "[" index "]" |

table-ref ::= table "@" place "->" column-name

primary   ::= literal-constant |
              symbolic-constant |
              "#" |
              real-ref |
              table-ref |
              function-name "(" arguments ")" |
              (real-expression)

factor    ::= primary |
              factor "^" primary

term      ::= factor |
              term "*" factor |
              term "/" factor

real-expr ::= term |
              "+" term |
              "-" term |
              real-expr "+" term |
              real-expr "-" term |
```

It may contain functions (see Table 2.5), Parentheses indicate operator precedence if required. Constant sub-expressions are evaluated immediately, and the result is stored as a constant.

2.7 Operators

An expression can be formed using operators (see Table 2.4) and functions (see Table 2.5) acting on operands (see Section 2.8).

Table 2.4: Real Operators in MAD-9

Operator	Meaning	result type	operand type(s)
	Real operators with one operand		
`+ X`	unary plus, returns `X`	real	real
`- X`	unary minus, returns the negative of `X`	real	real
	Real operators with two operands		
`X + Y`	add `X` to `Y`	real	real,real
`X - Y`	subtract `Y` from `X`	real	real,real
`X * Y`	multiply `X` by `Y`	real	real,real
`X / Y`	divide `X` by `Y`	real	real,real
`X ^ Y`	power, return `X` raised to the power `Y` ($Y > 0$)	real	real,real

Care must be used when an ordinary expression contains a random generator. It may be re-evaluated at unpredictable times, generating a new value. However, the use of a random generator in an assignment expression is safe. Examples:

```
D:DRIFT,L=0.01*RANF();    // a drift space with random length,
                          // may change during execution.
P=EVAL(0.001*TGAUSS(X));  // Evaluated once and stored as a constant.
```

2.8 Operands in Expressions

A real expression may contain the operands listed in the follolwing subsections.

2.8.1 Literal Constants

Numerical values are entered like FORTRAN constants. Real values are accepted in INTEGER or REAL format. The use of a decimal exponent, marked by the letter `D` or `E`, is permitted.
Examples:

```
1, 10.35, 5E3, 314.1592E-2
```

Table 2.5: Real Functions in MAD-9

Function	Meaning	result type	argument type(s)
Real functions with no arguments			
`RANF()`	random number, uniform distribution in [0,1)	real	-
`GAUSS()`	random number, Gaussian distribution with $\sigma = 1$	real	-
`USER0()`	random number, user-defined distribution	real	-
`SI()`	arc length from start of ring to the entry of the current element. This function is only available in the `EALIGN` command (see Section 10.3)	real	-
`SC()`	arc length from start of ring to the centre of the current element. This function is only available in the `EALIGN` command (see Section 10.3)	real	-
`SO()`	arc length from start of ring to the exit of current the element. This function is only available in the `EALIGN` command (see Section 10.3)	real	-
Real functions with one argument			
`TRUNC(X)`	truncate `X` towards zero (discard fractional part)	real	real
`ROUND(X)`	round `X` to nearest integer	real	real
`FLOOR(X)`	return largest integer not greater than `X`	real	real
`CEIL(X)`	return smallest integer not less than `X`	real	real
`SIGN(X)`	return sign of `X` (+1 for `X` positive, -1 for `X` negative, 0 for `X` zero)	real	real
`SQRT(X)`	return square root of `X`	real	real
`LOG(X)`	return natural logarithm of `X`	real	real
`EXP(X)`	return exponential to the base e of `X`	real	real
`SIN(X)`	return trigonometric sine of `X`	real	real
`COS(X)`	return trigonometric cosine of `X`	real	real
`ABS(X)`	return absolute value of `X`	real	real
`TAN(X)`	return trigonometric tangent of `X`	real	real
`ASIN(X)`	return inverse trigonometric sine of `X`	real	real
`ACOS(X)`	return inverse trigonometric cosine of `X`	real	real
`ATAN(X)`	return inverse trigonometric tangent of `X`	real	real
`TGAUSS(X)`	random number, Gaussian distribution with σ=1, truncated at `X`	real	real
`USER1(X)`	random number, user-defined distribution with one parameter	real	real
`EVAL(X)`	evaluate the argument immediately and transmit it as a constant	real	real
Real functions with two arguments			
`ATAN2(X,Y)`	return inverse trigonometric tangent of `Y/X`	real	real,real
`MAX(X,Y)`	return the larger of `X, Y`	real	real,real
`MIN(X,Y)`	return the smaller of `X, Y`	real	real,real
`MOD(X,Y)`	return the largest value less than `Y` which differs from `X` by a multiple of `Y`	real	real,real
`USER2(X,Y)`	random number, user-defined distribution with two parameters	real	real,real

2.8.2 Symbolic constants

MAD recognises a few build-in built-in mathematical and physical constants (see Table 2.7). Their names must not be used for user-defined labels. Additional symbolic constants may be defined (see Section 3.4.2) to simplify their repeated use in statements and expressions.

Table 2.6: Real Functions of Arrays in MAD-9

Function	Meaning	result type	operand type
`VMAX(X,Y)`	return largest array component	real	real array
`VMIN(X,Y)`	return smallest array component	real	real array
`VRMS(X,Y)`	return rms value of an array	real	real array
`VABSMAX(X,Y)`	return absolute largest array component	real	real array

Table 2.7: Predefined Symbolic Constants

MAD name	Mathematical symbol	Value	Unit
PI	π	3.1415926535898	1
TWOPI	2π	6.2831853071796	1
DEGRAD	$180/\pi$	57.295779513082	deg/rad
RADDEG	$\pi/180$	.017453292519943	rad/deg
E	e	2.7182818284590	1
EMASS	m_e	.51099906e-3	GeV
PMASS	m_p	.93827231	GeV
CLIGHT	c	299792458	m/s

2.8.3 Variable labels

Often a set of numerical values depends on a common variable parameter. Such a variable must be defined as a global variable (see Section 3.4.1) defined by one of

```
X=expression;
X:=expression;
VECTOR X=vector-expression;
VECTOR X:=vector-expression;
```

When such a variable is used in an expression, MAD uses the current value of the variable. When the value is a constant or an expression preceded by the delimiter `=` it is evaluated immediately and the result is assigned to the variable as a constant. When the value is an expression preceded by the delimiter `:=` the expression is retained and re-evaluated whenever one of its operands changes. Example:

```
L=1.0;
X:=L;
D1:DRIFT,L:=X;
D2:DRIFT,L:=2.0-X;
```

When the value of `X` is changed, the lengths of the drift spaces are recalculated as `X` and `2-X` respectively.

2.8.4 Element or command attributes

In arithmetic expressions the attributes of physical elements or commands can occur as operands. They are named respectively by

```
element-name->attribute-name
command-name->attribute-name
```

If they are arrays, they are denoted by

```
element-name->attribute-name[index]
command-name->attribute-name[index]
```

Values are assigned to attributes in element definitions or commands.
Example:

```
D1:DRIFT,L=1.0;
D2:DRIFT,L=2.0-D1->L;
```

`D1->L` refers to the length `L` of the drift space `D1`.

2.8.5 Deferred Expressions and Random Values

Definition of random machine imperfections requires evaluation of expressions containing random functions. These are not evaluated like other expressions before a command begins execution, but sampled as required from the distributions indicated when errors are generated. Such an expression is known as a **deferred expression**. Its value cannot occur as an operand in another expression.
Example:

```
ERROR:EALIGN,CLASS=QUADRUPOLE,DX=SIGMA*GAUSS();
```

All elements in range are assigned independent random displacements sampled from a Gaussian distribution with standard deviation `SIGMA`. The quantity `ERROR->DX` must not occur as an operand in another expression.

2.8.6 Table References

Values can be extracted from a table with the syntax

```
table-name "@" place "->" column-name
```

Here `table-name` denotes a table (see Section 7), `place` denotes a table row (see Section 2.9.1), and `column-name` denotes the name of a column in the table.
Example:

```
TWISS@#E->BETX
```

denotes the horizontal beta function at the end of table `TWISS`.

2.9 Element Selection

Many MAD commands allow for the possibility to process or display a subset of the elements occurring in a beam line or sequence.

2.9.1 Element Selection

A `place` denotes a single element, or the position **following** that element. It can be specified by one of the choices

object-name[index] The name verb'object-name' is the name of an element, line or sequence, and the integer `index` is its occurrence count in the beam line. If the element is unique, `[index]` can be omitted.

#S denotes the position before the first physical element in the **full** beam line. This position can also be written `#0`.

#E denotes the position after the last physical element in the **full** beam line.

Either form may be qualified by one or more beam line names, as described by the formal syntax:

```
place ::= element-name |
          element-name "[" integer "]" |
          "#S" |
          "#E" |
          line-name "::" place
```

An omitted index defaults to one. Examples: assume the following definitions:

```
M: MARKER;
S: LINE=(C,M,D);
L: LINE=(A,M,B,2*S,A,M,B);
   SURVEY,LINE=L
```

The line `L` is equivalent to the sequence of elements

```
A,M,B,C,M,D,C,M,D,A,M,B
```

Some possible `place` definitions are:

C[1] The first occurrence of element `C`.

#S The beginning of the line `L`.

M[2] The second marker `M` at top level of line `L`, i. e. the marker between second `A` and the second `B`.

#E The end of line `L`

S[1]::M[1] The marker `M` nested in the first occurrence of `S`.

2.9.2 Range Selection

A `range` in a beam line (see Section 5.1) is selected by the following syntax:

```
range ::= place |
          place "/" place
```

This denotes the range of elements from the first`place` to the second `place`. Both positions are included. A few special cases are worth noting:

- When `place1` refers to a `LINE` (see Section 5.1). the range starts at the **beginning** of this line.
- When `place2` refers to a `LINE` (see Section 5.1). the range ends at the **ending** of this line.
- When both `place` specifications refer to the same object, then the second can be omitted. In this case, and if `place` refers to a `LINE` (see Section 5.1) the range contains the whole of the line.

Examples: Assume the following definitions:

```
M: MARKER;
S: LINE=(C,M,D);
L: LINE=(A,M,B,2*S,A,M,B);
```

The line L is equivalent to the sequence of elements

```
A,M,B,C,M,D,C,M,D,A,M,B
```

Examples for `range` selections:

#S/#E The full range or `L`.

A[1]/A[2] `A[1]` through `A[2]`, both included.

S::M/S[2]::M From the marker `M` nested in the first occurrence of `S`, to the marker `M` nested in the second occurrence of `S`.

S[1]/S[2] Entrance of first occurrence of `S` through exit of second occurrence of `S`.

2.10 Constraints

In matching it is desired to specify equality constraints, as well as lower and upper limits for a quantity. MAD accepts the following form of constraints:

```
constraint          ::= array-expr constraint-operator array-expr

constraint-operator ::= "==" | "<" | ">"
```

2.11 Variable Names

A variable name can have one of the formats:

```
variable name ::= real variable |
                  object"->"real attribute
```

The first format refers to the value of the `global variable` (see Section 3.4.1), the second format refers to a named `attribute` of the named `object`. `object` can refer to an element or a command

2.12 Regular Expressions

Some commands allow selection of items via a `regular-expression`. Such a pattern string **must** be enclosed in single or double quotes; and the case of letters is significant. The meaning of special characters follows the standard UNIX usage: utility:

. Stands for a single arbitrary character,

[letter...letter] Stands for a single character occurring in the bracketed string, Example: "`[abc]`" denotes the choice of one of `a,b,c`.

[character-character] Stands for a single character from a range of characters, Example: "`[a-zA-Z]`" denotes the choice of any letter.

***** Allows zero or more repetitions of the preceding item, Example: "`[A-Z]*`" denotes a string of zero or more upper case letters.

\character Removes the special meaning of `character`, Example: "`\*`" denotes a literal asterisk.

All other characters stand for themselves. The pattern

```
"[A-Za-z][A-Za-z0-9_']*"
```

illustrates all possible unquoted identifier formats (see Section 2.2). Since identifiers are converted to lower case, after reading they will match the pattern

```
"[a-z][a-z0-9_']*"
```

Examples for pattern use:

```
SELECT,PATTERN="D.."
SAVE,PATTERN="K.*QD.*\.R1"
```

The first command selects all elements whose names have exactly three characters and begin with the letter `D`. The second command saves definitions beginning with the letter `K`, containing the string `QD`, and ending with the string `.R1`. The two occurrences of `.*` each stand for an arbitrary number (including zero) of any character, and the occurrence `\.` stands for a literal period.

2.13 Token List

In some special commands `LIST` (see Section 7.9) an attribute cannot be parsed immediately, since some information may not yet be available during parsing. Such an attribute is entered as a "token list", and it is parsed again when the information becomes available. Token lists can occur in token list arrays (see Section 2.14.4).
Example:

```
LIST,COLUMN={X:12:6,Y:12:6};
```

where `X:12:6` and `Y:12:6` are two token lists, and {`X:12:6,Y:12:6`} is a token list array.

2.14 Arrays

An attribute array is a set of values of the same `attribute type` (see Section 2.3). Normally an array is entered as a list in braces:

```
{value,...,value}
```

The list length is only limited by the available storage. If the array has only one value, the braces () can be omitted:

```
value
```

2.14.1 Logical Arrays

For the time being, logical arrays can only be given as a list. The formal syntax is:

```
logical-array ::= "{" logical-list "}"

logical-list  ::= logical-expr |
                  logical-list "," logical-expr
```

Example:

```
{true,true,a==b,false,x>y && y>z,true,false}
```

2.14.2 Real Arrays

Real arrays have the following syntax:

```
array-ref     ::= array-variable |
                  object "->" array-attribute |

table-ref     ::= "ROW" "(" table "," place ")" |
                  "ROW" "(" table "," place "," column-list ")"
                  "COLUMN" "(" table "," column ")" |
                  "COLUMN" "(" table "," column "," range ")"

columns       ::= column |
                  "{" column-list "}"

column-list   ::= column |
                  column-list "," column

column        ::= string

real-list     ::= real-expr |
                  real-list "," real-expr

index-select  ::= integer |
                  integer "," integer |
                  integer "," integer "," integer

array-primary ::= "{" real-list "}" |
                  "TABLE" "(" index-select "," real-expr ")" |
                  array-ref |
                  table-ref |
                  array-function-name "(" arguments ")" |
                  (array-expression)

array-factor  ::= array-primary |
                  array-factor "ˆ" array-primary
```

```
array-term    ::= array-factor |
                  array-term "*" array-factor |
                  array-term "/" array-factor

array-expr    ::= array-term |
                  "+" array-term |
                  "-" array-term |
                  array-expr "+" array-term |
                  array-expr "-" array-term |
```

Table 2.8: Real Array Functions in MAD-9 (acting component-wise)

Function	Meaning	result type	argument type
`TRUNC(X)`	truncate `X` towards zero (discard fractional part)	real array	real array
`ROUND(X)`	round `X` to nearest integer	real array	real array
`FLOOR(X)`	return largest integer not greater than `X`	real array	real array
`CEIL(X)`	return smallest integer not less than `X`	real array	real array
`SIGN(X)`	return sign of `X` (+1 for `X` positive, -1 for `X` negative, 0 for `X` zero)	real array	real array
`SQRT(X)`	return square root of `X`	real array	real array
`LOG(X)`	return natural logarithm of `X`	real array	real array
`EXP(X)`	return exponential to the base e of `X`	real array	real array
`SIN(X)`	return trigonometric sine of `X`	real array	real array
`COS(X)`	return trigonometric cosine of `X`	real array	real array
`ABS(X)`	return absolute value of `X`	real array	real array
`TAN(X)`	return trigonometric tangent of `X`	real array	real array
`ASIN(X)`	return inverse trigonometric sine of `X`	real array	real array
`ACOS(X)`	return inverse trigonometric cosine of `X`	real array	real array
`ATAN(X)`	return inverse trigonometric tangent of `X`	real array	real array
`TGAUSS(X)`	random number, Gaussian distribution with σ=1, truncated at `X`	real array	real array
`USER1(X)`	random number, user-defined distribution with one parameter	real array	real array
`EVAL(X)`	evaluate the argument immediately and transmit it as a constant	real array	real array

Example:

```
{a,a+b,a+2*b}
```

There are also three functions allowing the generation of real arrays:

TABLE Generate an array of expressions:

```
TABLE(n2,expression)       // implies TABLE(1:n2:1,expression)
TABLE(n1:n2,expression)    // implies TABLE(n1:n2:1,expression)
TABLE(n1:n2:n3,expression)
```

These expressions all generate an array with `n2` components. The components selected by `n1:n2:n3` are filled from the given `expression`; a C pseudo-code for filling is

```
int i;
for (i = n1; i <= n2; i += n3) a[i] = expression(i);
```

In each generated expression the special character hash sign (#) is replaced by the current value of the index `i`.

Example:

```
    // an array with 9 components, evaluates to {1,4,7,10,13}:
    table(5:9:2,3*#+1) // equivalent to {0,0,0,0,16,0,22,0,28}
```

ROW Generate a table row:

```
    ROW(table,place)             // implies all columns
    ROW(table,place,column list)
```

This generates an array containing the named (or all) columns in the selected place.

COLUMN Generate a table column:

```
    COLUMN(table,column)         // implies all rows
    COLUMN(table,column,range)
```

This generates an array containing the selected (or all) rows of the named column.

2.14.3 String Arrays

String arrays can only be given as lists of single values. For permissible values String values (see Section 2.4).
Example:

```
{A, "xyz", A & STRING(X)}
```

2.14.4 Token List Arrays

Token list arrays are always lists of single token lists.
Example:

```
{X:12:8,Y:12:8}
```

Chapter 3

Control Statements

3.1 Getting Help

3.1.1 HELP Command

A user who is uncertain about the attributes of a command should try the command `HELP`, which has three formats:

```
HELP;                  // Give help on the "HELP" command
HELP,NAME=label;       // List function and attribute types of "label"
HELP,label;            // Shortcut for the second format
```

`label` is an `identifier` (see Section 2.2). If it is non-blank, MAD prints the function of the object `label` and lists its attribute types. Entering `HELP` alone displays help on the `HELP` command.
Examples:

```
HELP;
HELP,NAME=TWISS;
HELP,TWISS;
```

3.1.2 SHOW Command

The `SHOW` statement displays the current attribute values of an object. It has three formats:

```
SHOW;                  // Give help on the "SHOW" command
SHOW,NAME=pattern;     // Show names matching of "pattern"
SHOW,pattern;          // Shortcut for the second format
```

`pattern` is an `regular expression` (see Section 2.12). If it is non-blank, MAD displays all object names matching the `pattern`. Entering `SHOW` alone displays help on the `SHOW` command. If the `OPTION,MAD8` (see Section 3.3) is active, the format is MAD-8 format, otherwise it is MAD-9 format. Examples:

```
SHOW;
SHOW,NAME="QD.*\.L*";
SHOW,"QD.*\.L*";
```

3.1.3 WHAT Command

The `WHAT` statement displays all object names matching a given regular expression. It has three formats:

```
WHAT;                    // Give help on the "WHAT" command
WHAT,NAME=label;         // Show definition of "label"
WHAT,label;              // Shortcut for the second format
```

`label` is an `identifier` (see Section 2.2). If it is non-blank, MAD displays the object `label` in a format similar to the input statement that created the object. Entering `WHAT` alone displays help on the `WHAT` command. Examples:

```
WHAT;
WHAT,NAME=QD;
WHAT,QD;
```

3.2 STOP Statement

The statement

```
STOP;
```

terminates execution of the MAD program, or, when the statement occurs in a `CALL` file (see Section 3.8.1), returns to the calling file. Any statement following the `STOP` statement is ignored.

3.3 OPTION Statement

The `OPTION` command controls global command execution and sets a few global quantities:

```
OPTION,ECHO=logical,INFO=logical,TRACE=logical,VERIFY=logical,
       WARN=logical,ADDERROR=logical,SEED=real,TELL=logical;
```

The first five logical flags activate or deactivate execution options:

ECHO Controls printing of an echo of input lines on the standard error file.

INFO If this option is turned off, MAD suppresses all information messages.

MAD8 When the MAD8 option is on, all output of the `SHOW` (see Section 3.1.2), `SAVE` (see Section 3.8.2), `ESAVE` (see Section 10.6), and `MAKESEQ` (see Section 3.8.3) is written in MAD-8 format. MAD-9 can thus serve as a translator from MAD-9 to MAD-8 format.

TRACE When the TRACE option is on, MAD writes additional trace information on the standard error file for each executable command. This information includes the command name and elapsed CPU time before and after the command.

VERIFY If this option is on, MAD gives a message for each undefined variable or element in a beam line.

WARN If this option is turned off, MAD suppresses all warning messages.

ADDERROR If this logical flag is set, an `EALIGN`, `EFIELD`, or `EFCOMP`, causes the errors to be added on top of existing ones. If it is not set, new errors overwrite any previous definitions.

SEED Selects a particular sequence of random values. A SEED value is an integer in the range [0...999999999] (default: 123456789). SEED can be an expression. See also: random values (see Section 2.8.5).

The last attribute requests listing of the current settings:

TELL If true, the current settings are listed.

Examples:

```
OPTION,ECHO=FALSE,TELL;
OPTION,SEED=987456321
```

Table 3.1: Default Settings for Options

`ECHO`	= `true`	`INFO`	= `true`	`TRACE`	= `false`	`VERIFY`	= `false`
`WARN`	= `true`	`ADDERROR`	= `false`	`SEED`	= 123456789	`TELL`	= `false`

3.4 Parameter Statements

3.4.1 Variable Definitions

MAD-9 recognises several types of variables.

Real Scalar Variables

```
REAL variable-name=real-expression;
```

The keyword `REAL` is optional. For backward compatibility the program also accepts the form

```
variable-name:=real-expression;
```

This statement creates a new global variable `variable-name` and discards any old variable with the same name. Its value depends on all quantities occurring in `real-expression` (see Section 2.6). Whenever an operand changes in `real-expression`, a new value is calculated. The definition may be thought of as a mathematical equation. However, MAD is not able to solve the equation for a quantity on the right-hand side.

An assignment in the sense of the FORTRAN or C languages can be achieved by using the `EVAL` function (see Section 3.4.4).

A reserved variable is the value `P0` which is used as the global reference momentum for normalising all magnetic field coefficients. Example:

```
REAL GEV=100;
P0=GEV;
```

Circular definitions are not allowed:

```
X=X+1;    // X cannot be equal to X+1
A=B;
B=A;      // A and B are equal, but of unknown value
```

However, redefinitions by assignment are allowed:

```
X=EVAL(X+1);
```

Real Vector Variables

```
REAL VECTOR variable-name=vector-expression;
```

The keyword `REAL` is optional. This statement creates a new global variable `variable-name` and discards any old variable with the same name. Its value depends on all quantities occurring in `vector-expression` (see Section 2.14) on the right-hand side. Whenever an operand changes in `vector-expression`, a new value is calculated. The definition may be thought of as a mathematical equation. However, MAD is not able to solve the equation for a quantity on the right-hand side.
Example:

```
REAL VECTOR A = TABLE(10, #);
REAL VECTOR B = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

Circular definitions are not allowed, but redefinitions by assignment are allowed.

Logical Variables

```
BOOL variable-name=logical-expression;
```

This statement creates a new global variable `variable-name` and discards any old variable with the same name. Its value depends on all quantities occurring in `logical-expression` (see Section 2.5). Whenever an operand changes in `logical-expression`, a new value is calculated. The definition may be thought of as a mathematical equation. However, MAD is not able to solve the equation for a quantity on the right-hand side.
Example:

```
BOOL FLAG = X != 0;
```

Circular definitions are not allowed, but redefinitions by assignment are allowed.

3.4.2 Symbolic Constants

MAD recognises a few build-in built-in mathematical and physical constants. (see Table 2.7) Additional constants can be defined by the command

```
REAL CONST label:CONSTANT=<real-expression>;
```

which defines a constant with the name `label`. The keyword `REAL` is optional, and `label` must be unique. An existing symbolic constant can never be redefined. The `real-expression` is evaluated at the time the `CONST` definition is read, and the result is stored as the value of the constant.
Example:

```
CONST IN=0.0254; // conversion of inches to metres
```

3.4.3 Vector Values

A vector of expressions is established by a statement

```
REAL VECTOR vector-name=vector-expression;
```

The keyword `REAL` is optional. It creates a new global vector `vector-name` and discards any old vector with the same name. Its value depends on all quantities occurring in `vector-expression` (see Section 2.14). Whenever an operand changes in `vector-expression`, a new value is calculated. The definition may be thought of as a mathematical equation. However, MAD is not able to solve the equation for a quantity on the right-hand side.
Example:

```
VECTOR A_AMPL={2.5e-3,3.4e-2,0,4.5e-8};
VECTOR A_ON=TABLE(10,1);
```

Circular definitions are not allowed.

3.4.4 Assignment to Variables

A value is assigned to a variable or vector by using the function `EVAL(real-expression)`. When seen, this function is immediately evaluated and replaced by the result treated like a constant.

```
variable-name=EVAL(real-expression);
```

This statement acts like a FORTRAN or C assignment. The `real-expression` or `vector-expression` is **evaluated**, and the result is assigned as a constant to the variable or vector on the left-hand side. Finally the expression is discarded. The `EVAL` function can also be used within an expression, e.g.,

```
vector-name=TABLE(range,EVAL(real-expression));
vector-name={...,EVAL(real-expression),...);
```

A sequence like the following is permitted:

```
...                 // some definitions
X=0;                // create variable X with value zero
WHILE (X <= 0.10) {
  TWISS,LINE=...;   // uses X=0, 0.01, ..., 0.10
  X=EVAL(X+.01);    // increment variable X by 0.01
                    // CANNOT use: X=X+.01;
}
```

3.4.5 VALUE: Output of Expressions

The statement

```
VALUE,VALUE=expression-vector;
```

evaluates a set of expressions using the most recent values of any operands and prints the results on the standard error file.
Example:

```
A=4;
VALUE,VALUE=TABLE(5,#*A);
P1=5;
P2=7;
VALUE,VALUE={P1,P2,P1*P2-3};
```

These commands give the results:

```
value: {0*A,1*A,2*A,3*A,4*A} = {0,4,8,12,16}
value: {P1,P2,P1*P2-3} = {5,7,32}
```

This commands serves mainly for printing one or more quantities which depend on matched attributes. It also allows the use of MAD as a programmable calculator. One may also tabulate functions.

3.5 DOOM: Interact with the DOOM Data Base

MAD can now be interfaced to the DOOM (MAD Object-Oriented Database) data base; this allows the user to transmit most objects known by MAD to and from user-written programs. We assume that MAD-9 has been launched with:

```
mad9 -db data-base-name
```

It will then open the data base `data-base-name` and read all data stored therein. Whenever the user requests it, or when a `STOP` statement is executed, MAD will write all modified objects to the data base and (temporarily) close it. Another program can then access the data base and optionally modify it. MAD regains access to the data base by explicitely re-opening it.

Interaction with the data base is done with the command

```
DOOM,OPEN=logical,CLOSE=logical,SHUT=logical,NOUPDATE=logical,
     DEBUG=integer;
```

which has the following attributes:

CLOSE Close the database for good. Afterwards one may continue with MAD without affecting the database contents. At the `STOP;` command a `DOOM,CLOSE;` is executed by default.

SHUT Close the database temporarily before a launching a user program by a `SYSTEM` call (see Section 3.6.2).

OPEN Reopen the database after `DOOM,CLOSE;` or `DOOM,SHUT;` and possible execution of a system call. The first `DOOM,OPEN;` is performed automatically when MAD-9 is launched with a data base name and need not be requested explicitly.

NOUPDATE After this command, the database is not updated any more, be it at a `DOOM,SHUT;`, `DOOM,CLOSE;`, or at the `STOP;` command. This can come handy when you do not want to spoil the contents of the database. You can revoke the effect (i.e. enable updating) with

```
DOOM,NOUPDATE=false;
```

DEBUG Enables debugging, if the `<integer>` is non-zero.

Example:

```
// Open the data base by default when launching MAD.
// Read some input.
...
DOOM,CLOSE; // Allow access to another program
// Do something which does not require access to the data base
...
SYSTEM,"..."; // Use the data base from outside MAD
// Do another thing which does not require access to the data base
...
DOOM,OPEN; // Re-gain access to the data base
// Modify something.
...
STOP; // Data base is closed by default.
```

3.6 Miscellaneous Commands

3.6.1 ECHO Statement

The `ECHO` statement has two formats:

```
ECHO,MESSAGE=message;
ECHO,message;           // shortcut
```

`message` is a string value (see Section 2.4). It is immediately transmitted to the `ECHO` stream.

3.6.2 SYSTEM: Execute System Command

During an interactive MAD session the command `SYSTEM` can be used to execute operating system commands. After execution of the system command, successful or not, control returns to MAD. At present this command is only available under UNIX or VM/CMS. It has two formats:

```
SYSTEM,CMD=string;
SYSTEM,string;         // shortcut
```

The string (see Section 2.4) `string` must be a valid operating system command.

3.6.3 SYSTEM Command under UNIX

Most UNIX commands can be issued directly.
Example:

```
SYSTEM,"ls -l"
```

causes a listing of the current directory in long form on the terminal.

3.6.4 SYSTEM Command under VM/CMS

If the system command refers to an EXEC file the string must begin with the word EXEC. Synonyms or abbreviations are not always accepted. It is recommended to use standard CMS command names and to spell them out in full.
Examples:

```
SYSTEM,"ERASE TESTDATA MAD A"
SYSTEM,"EXEC FILELIST * * D"
SYSTEM,"XEDIT TESTDATA MAD A"
```

3.7 TITLE Statement

The `TITLE` statement has three formats:

```
TITLE,STRING=page-header;   // define new page header
TITLE,page-header;          // shortcut for first format
TITLE,STRING="";            // clear page header
```

`page-header` is a string value (see Section 2.4). It defines the page header which will be used as a title for subsequent output pages. Before the first `TITLE` statement is encountered, the page header is empty. It can be redefined at any time.

3.8 File Handling

3.8.1 CALL Statement

The `CALL` command has two formats:

```
CALL,FILE=file-name;
CALL,file-name;
```

`file-name` is a string (see Section 2.4). The statement causes the input to switch to the named file. Input continues on that file until a `STOP` or an end of file is encountered. Example:

```
CALL,FILE="structure";
CALL,"structure";
```

3.8.2 SAVE Statement

The `SAVE` command has two formats:

```
SAVE,FILE=file-name
```

`file-name` is a string (see Section 2.4). The command causes all beam element, beam line, and parameter definitions to be written on the named file. If the `OPTION,MAD8` (see Section 3.3) is active, the format is MAD-8 format, otherwise it is MAD-9 format. The file may be read again in the same run. Examples:

```
SAVE,FILE="structure";
SAVE,"structure";
```

3.8.3 MAKESEQ Statement

A file containing a machine sequence can be generated in MAD-9 by the command

```
MAKESEQ,LINE=string,NAME=string,FILE=string;
```

The named beam line (see Section 5.1) or sequence (see Section 5.2) is written as a flat `SEQUENCE` (see Section 5.2) with the given name on the named file. If the `OPTION,MAD8` (see Section 3.3) is active, the format is MAD-8 format, otherwise it is MAD-9 format. All required elements and parameters are also written. All expressions are evaluated and only their values appear in the output. The command has the following attributes:

LINE The line for which a flat sequence is to be written.

NAME The name to be given to the sequence written.

FILE The name of the file to receive the output.

3.9 IF: Conditional Execution

Conditional execution can be requested by an `IF` statement. It allows usages similar to the C language `if` statement:

```
IF (logical) statement;
IF (logical) statement; ELSE statement;
IF (logical) { statement-group; }
IF (logical) { statement-group; } ELSE { statement-group; }
```

Note that all statements must be terminated with semicolons (`;`), but there is no semicolon after a closing brace. The statement or group of statements following the `IF` is executed if the condition is satisfied. If the condition is false, and there is an `ELSE`, the statement or group following the `ELSE` is executed.

3.10 WHILE: Repeated Execution

Repeated execution can be requested by a `WHILE` statement. It allows usages similar to the C language `while` statement:

```
WHILE (logical) statement;
WHILE (logical) { statement-group; }
```

Note that all statements must be terminated with semicolons (`;`), but there is no semicolon after a closing brace. The condition is re-evaluated in each iteration. The statement or group of statements following the `WHILE` is repeated as long as the condition is satisfied. Of course some variable(s) must be changed within the `WHILE` group to allow the loop to terminate.

3.11 MACRO: Macro Statements (Subroutines)

Subroutine-like commands can be defined by a `MACRO` statement. It allows usages similar to C language function call statements. A macro is defined by one of the following statements:

```
name(formals): MACRO { token-list }
name(): MACRO { token-list }
```

A macro may have formal arguments, which will be replaced by actual arguments at execution time. An empty formals list is denoted by `()`. Otherwise the `formals` consist of one or more names, separated by commas. The `token-list` consists of input tokens (strings, names, numbers, delimiters etc.) and is stored unchanged in the definition.

A macro is executed by one of the statements:

```
name(actuals);
name();
```

Each actual consists of a set of tokens which replaces all occurrences of the corresponding formal name. The actuals are separated by commas. Example:

```
// macro definitions:
SHOWIT(X): MACRO {
   SHOW, NAME = X;
}
DOIT(): MACRO {
   DYNAMIC,LINE=RING,FILE="DYNAMIC.OUT";
}

// macro calls:
SHOWIT(PI);
DOIT();
```

Chapter 4

Physical Elements and Markers

4.1 Element Input Format

All physical elements are defined by statements of the form

```
label:keyword,attribute,...,attribute
```

Example:

```
QF: QUADRUPOLE,L=1.8,K1=0.015832;
```

where

label Is the name to be given to the element (in the example QF), it is an `identifier` (see Section 2.2).

keyword Is a `keyword` (see Section 2.2), it is an element type keyword (in the example `QUADRUPOLE`),

attribute normally has the form

```
attribute-name=attribute-value
```

attribute-name selects the attribute from the list defined for the element type `keyword` (in the example `L` and `K1`). It must be an `identifier` (see Section 2.2)

attribute-value gives it a `value` (see Section 2.3) (in the example `1.8` and `0.015832`).

Omitted attributes are assigned a default value, normally zero.

4.2 Element Classes

The concept of element classes solves the problem of addressing instances of elements in the accelerator in a convenient manner. It also helps defining portions of the machine which are shared physically between two or more beam lines.

This concept will first be illustrated by an example. All the quadrupoles in the accelerator form a class `QUADRUPOLE`. Let us define three subclasses for the focussing quadrupoles, the defocussing quadrupoles, and the skewed quadrupoles:

```
MQF:QUADRUPOLE,L=LQM,K1=KQD; // Focusing quadrupoles
MQD:QUADRUPOLE,L=LQM,K1=KQF; // Defocusing quadrupoles
MQT:QUADRUPOLE,L=LQT;        // Skewed quadrupoles
```

These classes can be thought of as new keywords which define elements with specified default attributes. We now use them to define the real quadrupoles:

```
QD1:MQD; // Defocusing quadrupoles
QD2:MQD;
QD3:MQD;
...
QF1:MQF; // Focusing quadrupoles
```

```
QF2:MQF;
QF3:MQF;
...
QT1:MQT,K1S=KQT1; // Skewed quadrupoles
QT2:MQT,K1S=KQT2;
...
```

These quadrupoles inherit all unspecified attributes from their class. In this way, the user can build up a hierarchy of objects with a rather economic input structure.

The full power of the class concept is revealed when object classes are used to select (see Section 7.2) instances of elements for printing. Example:

```
SELECT,CLASS=QUADRUPOLE;  // Select all quadrupoles
PRINT,CLASS=MQT;          // Select skewed quadrupoles
```

More formally, for each element keyword MAD maintains a class of elements with the same name. A defined element becomes itself a class which can be used to define new objects, which will become members of this class. A new object inherits all attributes from its class; but its definition may override some of those values by new ones. All class and object names can be used in range selections, providing a powerful mechanism to specify positions for matching constraints and printing.

When an object is used in a beam line, MAD automatically makes a copy of that element, unless the object is defined as `SHARED`. The user can later attach imperfections to all copies individually. Example:

```
QF:QUADRUPOLE,...;           // Define the class QF
L:LINE=(...,QF,...,QF,...)   // Each QF is distinct
QF1:QF,...;                  // QF1 is derived from QF
```

On the contrary, if an object is defined as shared using the keyword `SHARED`, its use in more than one beam line implies the same object occurs in all those beam lines. Example:

```
SHARED QF:QUADRUPOLE,...;   // Define shared quadrupole QF
L1:LINE=(...,QF,...,QF,...) // All the QF's are the same ...
L2:LINE=(...,QF,...)        // ... also in this line.
```

4.3 Common Attributes for all Elements

The following attributes are allowed on all elements:

TYPE A `string value` (see Section 2.4). It specifies an “engineering type” and can be used for element selection.

APERTURE A real vector with an arbitrary length which describes the element aperture. It is ignored by MAD-9, but it can be used in other programs.

other All elements can have arbitrary additional attributes which are not defined below. Such attributes must have a name different from all defined attributes and single real values.

Only the `TYPE` attribute is used by MAD-9, but the `SAVE` command (see Section 3.8.2) saves all attributes unless `OPTION,MAD8` (see Section 3.3) is active.

4.4 Thin Lenses

All multipole-like elements (`RBEND, SBEND, QUADUPOLE, SEXTUPOLE, OCTUPOLE, MULTIPOLE`) can have a finite or a zero length. For finite length the multipole coefficients are the coefficients per unit length; for zero length they are interpreted as the integrated strength. The `SAVE` command (see Section 3.8.2) converts thin elements to thin multipoles in MAD-8 format, if `OPTION,MAD8` (see Section 3.3) is active.

4.5 Markers

```
label: MARKER,TYPE=string,APERTURE=vector;
```

The simplest element which can occur in a beam line is the `MARKER`. It has no effect on the beam, but it allows one to identify a position in the beam line, for example to apply a matching constraint. A marker has no attributes:
Example:

```
M27:MARKER;
```

4.6 Drift Spaces

```
label:DRIFT,TYPE=string,APERTURE=real-vector,L=real;
```

A DRIFT space has one real attribute:

L The drift length (default: 0 m)

Examples:

```
DR1:DRIFT,L=1.5;
DR2:DRIFT,L=DR1->L,TYPE=DRF;
```

The length of `DR2` will always be equal to the length of `DR1`. The reference system for a drift space is a Cartesian coordinate system (see Figure 1.3).

4.7 Bending Magnets

Two different type keywords are recognised for bending magnets, they are distinguished only by the reference system used:

RBEND is a rectangular bending magnet. It has parallel pole faces and is based on a Cartesian reference system (see Figure 1.4).

SBEND is a sector bending magnet. Its pole faces meet at the centre of curvature of the curved reference system (see Figure 1.5).

They are defined by the commands:

```
SBEND,TYPE=string,APERTURE=real-vector,L=real,ANGLE=real,
       K0=real,K1=real,K2=real,K3=real,K0s=real,K1S=real,K2S=real,K3S=real
       E1=real,E2=real,H1=real,H2=real,HGAP=real,FINT=real;
RBEND,TYPE=string,APERTURE=real-vector,L=real,ANGLE=real,
       K0=real,K1=real,K2=real,K3=real,K0S=real,K1S=real,K2S=real,K3S=real,
       E1=real,E2=real,H1=real,H2=real,HGAP=real,FINT=real;
```

For both types, the following attributes are permitted:

L The length of the magnet (default: 0 m). For a rectangular magnet the length is measured along a straight line, while for a sector magnet it is the arc length of the reference orbit. A thin dipole (see Section 4.4) is described with length zero. In this case all fields are the integrated fields.

ANGLE The **geometric** bend angle (default: 0 rad). It is this attribute only which determines the geometry of the magnet. A positive bend angle bends the reference axis to the right, i.e. towards negative x values.

K0 The normal dipole component $K_0 = \frac{1}{B\rho}B_y$. If this value is not given, it is taken as `ANGLE/L`. A positive value bends positive particles to the right (towards negative x).

K0S The skew dipole component $K_{0s} = \frac{1}{B\rho}B_x$. The default is $0\mathrm{m}^{-2}$. The component is positive for a bend up.

K1 The normal quadrupole component $K_1 = \frac{1}{B\rho}\frac{\partial B_y}{\partial x}$. The default is $0\mathrm{m}^{-2}$. The component is positive, if B_y is positive on the positive x-axis. This implies horizontal focusing of positively charged particles which travel in positive s direction.

K1S The skew quadrupole component $K_{1s} = \frac{1}{B\rho}\frac{\partial B_x}{\partial x}$. The default is $0\mathrm{m}^{-2}$. The component is negative, if B_x is positive on the positive x-axis.

K2 The normal sextupole component $K_2 = \frac{1}{B\rho}\frac{\partial^2 B_y}{\partial x^2}$. The default is $0\mathrm{m}^{-3}$. The component is positive, if B_y is positive on the positive x-axis.

K2S The skew sextupole component $K_{2s} = \frac{1}{B\rho}\frac{\partial^2 B_x}{\partial x^2}$. The default is $0\mathrm{m}^{-3}$. The component is negative, if B_x is positive on the positive x-axis.

K3 The normal sextupole component $K_3 = \frac{1}{B\rho}\frac{\partial^3 B_y}{\partial x^3}$. The default is $0\mathrm{m}^{-4}$. The component is positive, if B_y is positive on the positive x-axis.

K3S The skew sextupole component $K_{3s} = \frac{1}{B\rho}\frac{\partial^3 B_x}{\partial x^3}$. The default is $0\mathrm{m}^{-4}$. The component is negative, if B_x is positive on the positive x-axis.

E1 The rotation angle for the entrance pole face (default: 0 rad).

E2 The rotation angle for the exit pole face (default: 0 rad).

H1 The curvature of the entrance pole face (default: $0\mathrm{m}^{-1}$).

H2 The curvature of the exit pole face (default: $0\ \mathrm{m}^{-1}$). A positive pole face curvature induces a negative sextupole component; i.e. for positive `H1` and `H2` the centres of curvature of the pole faces are placed inside the magnet.

FINT The field integral (default =0).

HGAP The half gap of the magnet (default: 0 m).

The pole face rotation angles are referred to the magnet model for a `RBEND` (see Figure 1.4) and `SBEND` (see Figure 1.5) respectively. The quantities `FINT` and `HGAP` specify the finite extent of the fringe fields as defined in [1] as follows:

$$\mathtt{FINT} = \int_0^\infty \frac{B_y(s)(B_0 - B_y(s))}{B_0^2 g} ds, \qquad \mathtt{HGAP} = 2g.$$

The default values of zero corresponds to the hard-edge approximation, i.e. a rectangular field distribution. For other approximations, enter the correct value of the half gap, and one of the following values for `FINT`:

Typical values for `FINT`	
Linear Field drop-off	1/6
Clamped ”Rogowski” fringing field	0.4
Unclamped ”Rogowski” fringing field	0.7
”Square-edged” non-saturating magnet	0.45

A reasonable average value for `FINT` is 0.5. All thes dipole examples have the same bend angle:

```
BR:RBEND,L=5.5,ANGLE=+0.001;       // Deflection to the right
BR:RBEND,L=5.5,K0=+0.001/5.5;      // Deflection to the right
                                   // This magnet has a straight reference
BL:SBEND,L=5.5,ANGLE=-0.001;       // Deflection to the left
BL:SBEND,L=5.5,K0=-0.001/5.5;      // Deflection to the left
                                   // This magnet has a straight reference
```

4.8 Quadrupoles

```
label:QUADRUPOLE,TYPE=string,APERTURE=real-vector,L=real,K1=real,K1S=real;
```

A `QUADRUPOLE` has three real attributes:

L The quadrupole length (default: 0 m). A thin quadrupole (see Section 4.4) is defined by setting the length to zero.

K1 The normal quadrupole component $K_1 = \frac{1}{B\rho}\frac{\partial B_y}{\partial x}$. The default is $0\mathrm{m}^{-2}$. The component is positive, if B_y is positive on the positive x-axis. This implies horizontal focusing of positively charged particles which travel in positive s direction.

K1S The skew quadrupole component $K_{1s} = \frac{1}{B\rho}\frac{\partial B_x}{\partial x}$. The default is $0\mathrm{m}^{-2}$. The component is negative, if B_x is positive on the positive x-axis.

Example:

```
QF:QUADRUPOLE,L=1.5,K1=0.001;
```

The reference system for a quadrupole is a Cartesian coordinate system (see Figure 1.3).

4.9 Sextupoles

```
label: SEXTUPOLE,TYPE=string,APERTURE=real-vector,L=real,K2=real,K2S=real;
```

A `SEXTUPOLE` has three real attributes:

L The sextupole length (default: 0 m). A thin sextupole (see Section 4.4) is defined by setting the length to zero.

K2 The normal sextupole component $K_2 = \frac{1}{B\rho}\frac{\partial^2 B_y}{\partial x^2}$. The default is $0\mathrm{m}^{-3}$. The component is positive, if B_y is positive on the positive x-axis.

K2S The skew sextupole component $K_{2s} = \frac{1}{B\rho}\frac{\partial^2 B_x}{\partial x^2}$. The default is $0\mathrm{m}^{-3}$. The component is negative, if B_x is positive on the positive x-axis.

Example:

```
S:SEXTUPOLE,L=0.4,K2=0.00134;
```

The reference system for a sextupole is a Cartesian coordinate system (see Figure 1.3).

4.10 Octupoles

```
label:OCTUPOLE,TYPE=string,APERTURE=real-vector,L=real,K3=real,K3S=real;
```

An `OCTUPOLE` has three real attributes:

L The octupole length (default: 0 m). A thin octupole (see Section 4.4) is defined by setting the length to zero.

K3 The normal sextupole component $K_3 = \frac{1}{B\rho}\frac{\partial^3 B_y}{\partial x^3}$. The default is $0\mathrm{m}^{-4}$. The component is positive, if B_y is positive on the positive x-axis.

K3S The skew sextupole component $K_{3s} = \frac{1}{B\rho}\frac{\partial^3 B_x}{\partial x^3}$. The default is $0\mathrm{m}^{-4}$. The component is negative, if B_x is positive on the positive x-axis.

Example:

```
O3:OCTUPOLE,L=0.3,K3=0.543;
```

The reference system for an octupole is a Cartesian coordinate system (see Figure 1.3).

4.11 General Multipoles

A `MULTIPOLE` is thin lens of arbitrary order, including a dipole:

```
label:MULTIPOLE,TYPE=string,APERTURE=real-vector,L=real,
      KNORMAL=real-vector,KSKEW=real-vector;
```

L The multipole length (default: 0 m). A thin multipole (see Section 4.4) is defined by setting the length to zero.

KN A real vector (see Section 2.14), containing the normal multipole coefficients. A component is positive, if B_y is positive on the positive x-axis.

KS A real vector (see Section 2.14), containing the skew multipole coefficients. A component is negative, if B_x is positive on the positive x-axis.

The multipole coefficients are defined as $K_n = \frac{1}{B\rho}\frac{\partial^n B_y}{\partial x^n}$. (default: 0m^{-n}). The order n is unlimited, but all components up to the maximum must be given, even if zero. The number of poles of each component is $(2n+2)$. The most important error components of fully symmetric quadrupoles are: `KNORMAL[5]`, the 12-pole, and `KNORMAL[9]`, the twenty-pole. Superposition of many multipole components is permitted. The reference system for a multipole is a Cartesian coordinate system (see Figure 1.3).
Example:

```
M27:MULTIPOLE,L=1,KNORMAL[3]=0.0001,KSKEW[2]=0.0001;
```

A multipole with no dipole component has no effect on the reference orbit, i.e. the reference system at its exit is the same as at its entrance. If it includes a dipole component, it has the same effect on the reference orbit as a `SBEND` with the same length and deflection angle `KNORMAL[0]*L`.

4.12 Solenoids

```
label:SOLENOID,TYPE=string,APERTURE=real-vector,L=real,KS=real;
```

A `SOLENOID` has two real attributes:

L The length of the solenoid (default: 0 m)

KS The solenoid strength K_s (default: 0 rad/m). For positive `KS` and positive particle charge, the solenoid field points in the direction of increasing s.

Example:

```
SOLO:SOLENOID,L=2.,K=0.001;
```

The reference system for a solenoid is a Cartesian coordinate system (see Figure 1.3).

4.13 Orbit Correctors

Three types of closed orbit correctors are available:

HKICKER A corrector for the horizontal plane,

VKICKER A corrector for the vertical plane,

KICKER A corrector for both planes.

```
label:HKICKER,TYPE=string,APERTURE=real-vector,L=real,KICK=real;
label:VKICKER,TYPE=string,APERTURE=real-vector,L=real,KICK=real;
label:KICKER, TYPE=string,APERTURE=real-vector,L=real,HKICK=real,VKICK=real;
```

They have the following attributes:

L The length of the closed orbit corrector (default: 0 m).

KICK The kick angle for either horizontal or vertical correctors. (default: 0 rad).

HKICK The horizontal kick angle for a corrector in both planes (default: 0 rad).

VKICK The vertical kick angle for a corrector in both planes (default: 0 rad).

A positive kick increases p_x or p_y respectively.
Examples:

```
HK1:HKICKER,KICK=0.001;
VK3:VKICKER,KICK=0.0005;
KHV:KICKER, HKICK=0.001,VKICK=0.0005;
```

The first kicker is rotated about the longitudinal axis by 10 degrees. The reference system for an orbit corrector is a Cartesian coordinate system (see Figure 1.3).

4.14 RF Cavities

An `RFCAVITY` has seven real attributes and one integer attribute:

```
label:RFCAVITY,TYPE=string,APERTURE=real-vector,L=real,VOLT=real,LAG=real,
      HARMON=integer,BETRF=real,PG=real,SHUNT=real,TFILL=real;
```

L The length of the cavity (default: 0 m)

VOLT The peak RF voltage (default: 0 MV). The effect of the cavity is $\delta E = \texttt{VOLT} \sin(2\pi(\texttt{LAG} - \texttt{HARMON} \cdot f_0 t))$.

LAG The phase lag [2π] (default: 0).

HARMON The harmonic number h (no default). Note that the RF frequency is computed from the harmonic number and the revolution frequency f_0. **The frequency attribute `FREQ` must no longer be used.**

BETRF RF coupling factor (default: 0).

PG The RF power per cavity (default: 0 mW).

SHUNT The relative shunt impedance (default: $0 M\Omega/\mathrm{m}$).

TFILL The filling time of the cavity T_{fill} (default: $0 \mu\mathrm{s}$).

Use of a cavity requires the particle momentum `P` and the particle charge `CHARGE` to be set on the relevant optics command before any calculations is performed.
Example:

```
CAVITY:RFCAVITY,L=10.0,VOLT=150.0,LAG=0.0,HARMON=31320;
STATIC,P=50.0,PARTICLE=ELECTRON;
```

The reference system for a cavity is a Cartesian coordinate system (see Figure 1.3).

4.15 Electrostatic Separators

An `ELSEPARATOR` (electrostatic separator) has three real attributes:

```
label:ELSEPARATOR,TYPE=string,APERTURE=real-vector,L=real,EX=real,EY=real;
```

L The length of the separator (default: 0 m).

EX The horizontal electric field strength (default: 0 MV/m). A positive field increases p_x for positive particles.

EY The vertical electric field strength (default: 0 MV/m). A positive field increases p_y for positive particles.

A `ELSEPARATOR` requires the particle momentum `P` and its charge `CHARGE` to be set in the relevant `BEAM` command (see Section 6.2) before any calculations are performed.
Example:

```
BEAM,PARTICLE=POSITRON,PC=46.0;
SEP:ELSEPARATOR,L=5.0,E=0.5;
```

The reference system for a separator is a Cartesian coordinate system (see Figure 1.3).

4.16 Beam Position Monitors

A beam monitor acts on the beam like a drift space. In addition it serve records the beam position for closed orbit corrections. Four different types of beam position monitors are recognised:

HMONITOR Monitor for the horizontal beam position,

VMONITOR Monitor for the vertical beam position,

MONITOR Monitor for both horizontal and vertical beam position.

INSTRUMENT A place holder for any other type of beam instrumentation. Optically it behaves like a drift space; it returns **no beam observation**. It represents a class of elements which is completely independent from drifts and monitors.

```
label:HMONITOR,  TYPE=string,APERTURE=real-vecto,L=real;
label:VMONITOR,  TYPE=string,APERTURE=real-vecto,L=real;
label:MONITOR,   TYPE=string,APERTURE=real-vecto,L=real;
label:INSTRUMENT,TYPE=string,APERTURE=real-vector,L=real;
```

A beam position monitor has one real attribute:

L The length of the monitor (default: 0 m). If the length is different from zero, the beam position is recorded in the centre of the monitor (except for the `INSTRUMENT` element).

Examples:

```
MH:HMONITOR,L=1;
MV:VMONITOR;
```

The reference system for a monitor is a Cartesian coordinate system (see Figure 1.3).

4.17 Collimators

Two types of collimators are defined:

ECOLLIMATOR Elliptic aperture,

RCOLLIMATOR Rectangular aperture.

```
label:ECOLLIMATOR,TYPE=string,APERTURE=real-vector,L=real,
      XSIZE=real,YSIZE=real;
label:RCOLLIMATOR,TYPE=string,APERTURE=real-vector,L=real,
      XSIZE=real,YSIZE=real;
```

Either type has three real attributes:

L The collimator length (default: 0 m).

XSIZE The horizontal half-aperture (default: unlimited).

YSIZE The vertical half-aperture (default: unlimited).

For elliptic apertures, `XSIZE` and `YSIZE` denote the half-axes respectively, for rectangular apertures they denote the half-width of the rectangle. Optically a collimator behaves like a drift space, but during tracking, it also introduces an aperture limit. The aperture is checked at the entrance. If the length is not zero, the aperture is also checked at the exit.
Example:

```
COLLIM:ECOLLIMATOR,L=0.5,XSIZE=0.01,YSIZE=0.005;
```

The reference system for a collimator is a Cartesian coordinate system (see Figure 1.3).

4.18 Coordinate Transformations

4.18.1 Rotations About the Vertical Axis

```
label:YROT,TYPE=string,APERTURE=real-vector,ANGLE=real;
```

The element `YROT` rotates reference system about the vertical (y) axis (see Figure 1.7). `YROT` has no effect on the beam, but it causes the beam to be referred to the new coordinate system

$$\begin{aligned} x_2 &= x_1 \cos\theta - s_1 \sin\theta, \\ y_2 &= x_1 \sin\theta + s_1 \cos\theta, \end{aligned}$$

It has one real attribute:

ANGLE The rotation angle theta (default: 0 rad). It must be a **small** angle, i.e. an angle comparable to the transverse angles of the orbit.

A positive angle means that the new reference system is rotated clockwise about the local y-axis with respect to the old system.
Example:

```
KINK:YROT,ANGLE=0.0001;
```

4.18.2 Rotations Around the Longitudinal Axis

```
label:SROT,TYPE=string,APERTURE=real-vector,ANGLE=real;
```

The element `SROT` rotates the reference system about the longitudinal (s) axis (see Figure 1.6). `SROT` has no effect on the beam, but it causes the beam to be referred to the new coordinate system

$$\begin{aligned} x_2 &= x_1 \cos\psi - y_1 \sin\psi, \\ y_2 &= x_1 \sin\psi + y_1 \cos\psi, \end{aligned}$$

It has one real attribute:

ANGLE The rotation angle psi (default: 0 rad)

A positive angle means that the new reference system is rotated clockwise about the s-axis with respect to the old system.
Example:

```
ROLL1:SROT,ANGLE=PI/2.;
ROLL2:SROT,ANGLE=-PI/2.;
HBEND:SBEND,L=6.0,ANGLE=0.01;
VBEND:LINE=(ROLL1,HBEND,ROLL2);
```

The above is a way to represent a bend down in the vertical plane, it could be defined more simply by

```
VBEND:SBEND,L=6.0,K0S=-0.01/6.0;
```

4.18.3 General Change of Reference

```
label:PATCH,TYPE=string,APERTURE=real-vector,DX=real,DY=real,DZ=real,
            DTHETA=real,DPHI=real,DPSI=real;
```

The element `PATCH` applies a general change to the reference system. `PATCH` has no effect on the beam, but it causes the beam to be referred to the new coordinate system. It has six real attributes:

DX The displacement in x-direction of the new system with respect to the old one.

DY The displacement in y-direction of the new system with respect to the old one.

DS The displacement in s-direction of the new system with respect to the old one.

VX The rotation around the x-axis of the new system with respect to the old one.

VY The rotation around the y-axis of the new system with respect to the old one.

VZ The rotation around the s-axis of the new system with respect to the old one.

As an example consider a simplified model of the separation of the two beams (see Figure 4.1) near the interaction region of LHC:

```
ALPHA=...;    // The bend angle in the separator magnets.
DISTANCE=...; // The longitudinal distance between the separator magnets.
D1:HKICKER,L=0,KICK=ALPHA;
D2:HKICKER,L=0,KICK=-ALPHA;
DIST:DRIFT,L=DISTANCE;
PATCH1:PATCH,DX=DISTANCE*TAN(ALPHA);
```

```
PATCH2:PATCH,DX=-DISTANCE*TAN(ALPHA);
SHARED SEPARATION:LINE=(D1,DIST,D2);
RING1:SEQUENCE,...; // beam goes clockwise.
...
SEPARATION;
PATCH1;             // change reference to
...
ENDSEQUENCE;
RING2:SEQUENCE,...; // beam goes anticlockwise.
...
SEPARATION;
PATCH2;
...
ENDSEQUENCE;
```

The direction of travel of each beam determines the signs of the deflections, the patches change the reference. Note that the common reference between the two separator magnets allows a correct handling of long-distance beam-beam interactions in that area.

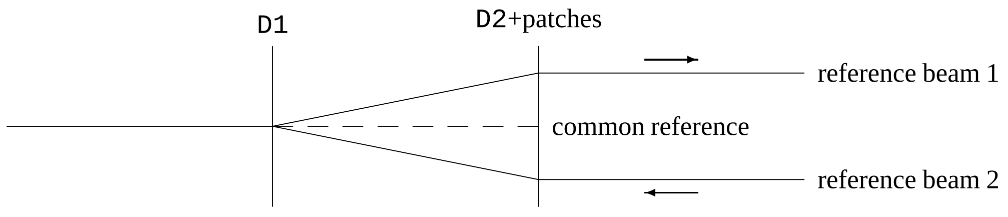

Figure 4.1: Separation of two beams

4.19 Beam-Beam Interactions

The command `BEAMBEAM` may be inserted in a beam line to simulate a beam-beam interaction point:

```
label:BEAMBEAM,TYPE=string,APERTURE=real-vector,
      SIGX=real,SIGY=real,XMA=real,YMA=real,CHARGE=real;
```

The code for this element has been contributed by J.M. Veuillen (1987), and has been adapted to C++ in 1997. It has six real attributes:

SIGX The horizontal extent (standard deviation) of the opposite beam (default: 0 m).

SIGY The vertical extent (standard deviation) of the opposite beam (default: 0 m).

XMA The horizontal displacement of the opposite beam with respect to the ideal orbit (default: 0 m).

YMA The vertical displacement of the opposite beam with respect to the ideal orbit (default: 0 m).

CHARGE The charge of particles in the opposite beam in proton charges (default: 0).

NPART The number of particles in the opposite beam. (default: 0).

A beam-beam element requires the particle momentum `P` and its charge `CHARGE` to be defined on the relevant optics command before any calculations are performed.
Example:

```
BEAM,MOMENTUM=46.0,MASS=PMASS,CHARGE=1.0;
BB:BEAMBEAM,SIGX=1.E-3,SIGY=5.E-4,CHARGE=1.0,NPART=1.0E12;
```

A three-dimensional representation of a beam-beam interaction is available with the command

```
label:BEAMINT,TYPE=string,APERTURE=real-vector,
      PHI=real,NSLI=integer,FAST=bool,XIYN=real,DX=real,DY=real,DZ=real,
      BETXS=real,BETYS=real,ALFXS=real,ALFYS=real,
      DXS=real,DPXS=real,DYS=real,DPYS=real,
      EXS=real,EYS=real,SIGTS=real,SIGES=real;
```

Its parameters are:

PHI Horizontal crossing angle.

NSLI Number of slices to be taken in strong beam.

FAST If true, use tables for computing the error function.

XIYN Beam-beam parameter.

DX Horizontal displacement of the interaction point in [m].

DY Vertical displacement of the interaction point in [m].

DZ Longitudinal displacement of the interaction point in [m].

The following parameters describe the strong beam:

BETXS Horizontal $\beta*$ in [m].

BETYS Vertical $\beta*$ in [m].

ALFXS Horizontal $\alpha*$* in [1].

ALFYS Vertical $\alpha*$ in [1].

DXS Horizontal dispersion in [m].

DPXS Horizontal dispersion slope in [m].

DYS Vertical dispersion in [m].

DPYS Vertical dispersion slope in [m].

EXS Horizontal emittance in [m rad].

EYS Vertical emittance in [m rad].

SIGTS Bunch length in [m].

SIGES Energy spread in [1].

4.20 Editing Element Definitions

An element definition can be changed in two ways:

Entering a new definition The element definition will be replaced. A beam element, `LINE` (see Section 5.1), or `SEQUENCE` (see Section 5.2) can be replaced by another beam element, beam line, or sequence; if the replaced item occurs in another beam line or sequence, all references in that line or sequence are replaced.

Entering the element name together with new attributes The element will be updated in place, and the new attribute values will replace the old ones. When the type of the element should not change, replacement of an attribute is the more efficient way.

Element definitions can be changed freely. The changes do not affect already defined objects which belong to its class (see Section 4.2). This example shows two ways to change the strength of a quadrupole:

```
QF:QUADRUPOLE,L=1,K1=0.01; // Original definition of QF
QF:QUADRUPOLE,L=1,K1=0.02; // Replace whole definition of QF
QF,K1=0.02;                // Replace value of K1
```

Chapter 5

Beam Lines, Sequences, and Ranges

5.1 Beam Lines

The accelerator to be studied is known to MAD as a sequence of physical elements called a **beam line**. A beam line is built from simpler beam lines whose definitions can be nested to any level. The syntax is powerful enough to repeat or to reflect pieces of beam lines. Formally a beam line is defined by a `LINE` command:

```
label:LINE=(member,...,member);
```

`label` (see Section 2.2) gives a name to the beam line for later reference.
Each `member` may be one of the following:

- An element label,
- A beam line label,
- A `SEQUENCE` (see Section 5.2) label,
- A sub-line, enclosed in parentheses,

Beam lines can be nested to any level.

5.1.1 Simple Beam Lines

The simplest beam line consists of single elements:

```
label:LINE=(member,...,member);
```

Example:

```
L:LINE=(A,B,C,D,A,D);
TWISS,LINE=L;
```

The `TWISS` command (see Section 7.7) tells MAD to perform lattice calculations on the sequence

```
A,B,C,D,A,D
```

5.1.2 Sub-lines

Instead of referring to an element, a beam line member can refer to another beam line defined in a separate command. This provides a shorthand notation for sub-lines which occur several times in a beam line. Lines and sub-lines can be entered in any order, but when a line is used, all its sub-lines must be known.
Example:

```
L:LINE=(A,B,S,B,A,S,A,B);
S:LINE=(C,D,E);
TWISS,LINE=printL;
```

This example produces the following expansion steps:

1. Replace sub-line `S`:

   ```
   (A,B,(C,D,E),B,A,(C,D,E),A,B)
   ```

2. Omit parentheses:

   ```
   A,B,C,D,E,B,A,C,D,E,A,B
   ```

5.1.3 Reflection and Repetition

An unsigned repetition count and an asterisk indicate repetition of a beam line member. An optional minus sign (`-`) prefix causes reflection, i.e. all elements in the subsequence are taken in reverse order. Sub-lines of reflected lines are also reflected, but on physical elements the reflection flag is ignored. The minus sign must precede any repetition count. Repetitions are expanded immediately when a line is read, so are reflections of anonymous beam lines. The result is a flat line referring to a sequence of named elements and/or beam lines. When the line is output, it has the form resulting from this expansion.

Example:

```
R:LINE=(G,H);
S:LINE=(C,R,D);
T:LINE=(2*S,2*(E,F),-S,-(A,B));
TWISS,LINE=T;
```

The three lines are stored as follows:

```
R:LINE=(G,H);
S:LINE=(C,R,D);
T:LINE=(S,S,E,F,E,F,-S,B,A);
```

When `T` is expanded, substitution is recursive:

1. Replace sub-line `S`:

   ```
   (C,R,D,C,R,D,E,F,E,F,D,-R,C,B,A)
   ```

2. Replace sub-line `R`:

   ```
   (C,G,H,D,C,G,H,D,E,F,E,F,D,H,G,C,B,A)
   ```

Note that the inner sub-line R is reflected together with the outer sub-line S.

5.2 Beam Line Sequences

A sequence of elements can easily be generated from a data base using a command looking like

```
label:SEQUENCE,REFER=keyword,L=expression,REFPOS=name;
   object-definition;
   ...;
   object-definition;
ENDSEQUENCE;
```

It reads a sequence of element definitions, compiles an object which resembles a beam line definition, and gives it the name ”label”. The resulting sequence can be used like a beam line. The attributes of the sequence are:

REFER The reference points for the elements are specified by the `REFER` attribute:

> **REFER=CENTRE** The reference points are at the element centres (default).
>
> **REFER=ENTRY** The reference points are at element entrances.

REFER=EXIT The reference points are at element exits.

L The length of the SEQUENCE can be given on the SEQUENCE command, or it may be entered with the ENDSEQUENCE command.

REFPOS Normally, the reference position for a nested sequence is defined by the REFER attribute of the enclosing sequence, but, if REFPOS is given, it specifies a **unique** element in the sequence whose AT attribute becomes the reference point for the sequence.

For each non-drift element in the sequence one element definition appears following the SEQUENCE command and preceding the ENDSEQUENCE command. These look similar to ordinary element definitions (see Section 4), but they may contain an optional specification to place the element:

```
class-name,AT=expression;
class-name,AT=expression,FROM=name;
class-name,DRIFT=expression;

object-name,class-name,AT=expression,attribute,...,attribute;
object-name,class-name,AT=expression,FROM=name,attribute,...,attribute;
object-name,class-name,DRIFT=expression,attribute,...,attribute;
```

The meaning of the AT specifications is:

AT=expression Place the element's entrance, centre, or exit at the specifiet position.

FROM=name Interpret the AT specification as relative to the **unique** element name.

FROM=#S Like omitting the FROM specification, AT is relative to the beginning of the sequence.

FROM=#E The AT specification is relative to the end of the sequence.

FROM=PREVIOUS The AT specification is relative to the previous element.

FROM=NEXT The AT specification is relative to the following element.

DRIFT=expression The element is preceded by a drift of the given length.

One should consider the following:

1. The name class-name must be an element class name (see Section 4.2), it may optionally be preceded by a minus sign (-). This inverts the order of the elements in the inserted object. This makes sense only for a beam line or sequence.
2. If the name object-name is not given, MAD inserts the element specified by class-name. In this case no further attributes are allowed.
3. If there is a non-blank object-name, this name should not be defined earlier in the data. MAD then first makes a copy of class-name and gives it the new name object-name. Any further attributes override the attributes inherited from class-name.b
4. The elements must be entered in order of increasing position, and they must not overlap. Their positions are evaluated while reading the SEQUENCE definition, and become **constant** values.
5. A LINE (see Section 5.1) or SEQUENCE (see Section 5.2) can be nested in another SEQUENCE or LINE.

Within the SEQUENCE, MAD generates the drift spaces for proper positioning.
Example:

```
// Define element classes for a simple cell:
B:  SBEND,L=35.09,ANGLE = 0.011306116;
QF: QUADRUPOLE,L=1.6,K1=-0.02268553;
QD: QUADRUPOLE,L=1.6,K1=0.022683642;
SF: SEXTUPOLE,L=0.4,K2=-0.13129;
SD: SEXTUPOLE,L=0.76,K2=0.26328;
```

```
// Define the cell as a sequence:
CELL:SEQUENCE,L=79.0;
   B1: B, AT=19.115;
   SF1:SF,AT=37.42;
   QF1:QF,AT=38.70;
   B2: B, AT=58.255,ANGLE=-B1->ANGLE;
   SD1:SD,AT=76.74;
   QD1:QD,AT=78.20;
ENDSEQUENCE;
```

In this example all members of the sequence have a new name, and MAD generates copies of the corresponding classes. The bending magnet `B1` is wrapped to change sign, but its definition is still equal to `B`. Thus `B2`, which has the negative field of `B1`, has the same effect as the wrapped element `B1`.

5.3 Lines and Sequences with arguments

A line or sequence definition can also have parameters like a `MACRO` (see Section 3.11). Such a line or sequence can be nested (and instantiated) in another line or sequence, but it **must have a unique name** when instantiated in a sequence.

Examples:

```
CELL(X,Y):SEQUENCE,L=79;
   QF&X: QF,AT=...;
   Y&X:  Y,L=1,AT=...;
   QD&X: QD,AT=...;
ENDSEQUENCE;
```

When used as

```
CELL12: CELL(12,SF);
```

this expands to

```
CELL12:SEQUENCE,L=79;
   QF12: QF,AT=...;
   SF12: SF,L=1,AT=...;
   QD12: QD,AT=...;
ENDSEQUENCE;
```

A second example:

```
CELL(X,Y):LINE=(D1,QF&X,D2,Y&X,D3,QD&X,D4);
```

If the proper drifts are used, this example is equivalent to the sequence example above.

5.4 Shared Lines

Normally, when a beam line or sequence is referred to in another line, each reference refers to a **distinct** copy of the line.

```
L:LINE=(A,B,C);
S:SEQUENCE,L=real;
   ...
ENDSEQUENCE;
```

Such a line or sequence is cloned when it is inserted in another line or sequence, thus permitting assignment of errors to its elements without affecting other occurrences of the same line.

A beam line or sequence can be shared by using the keyword `SHARED`, then the line or sequence is unique, and all references in other lines refer to the **same** instance. Example:

```
// Define the interaction region common to both rings:
SHARED IR2:SEQUENCE,L=...;
   ...
ENDSEQUENCE;
RING1:LINE=(...,IR2,...);
RING2:LINE=(...,IR2,...);
// Now assign imperfections to IR2 wich will be seen by both rings:
EALIGN,LINE=IR2,DX=...,DY=...;
```

Counterexample:

```
// Define one superperiod of the machine:
SUPER:SEQUENCE,L=...;
   ...
ENDSEQUENCE;
// Each occurrence of SUPER is distinct:
RING:LINE=(8*SUPER);
// Assign different imperfections to each SUPER:
EALIGN,LINE=RING,DX=...,DY=...;
```

5.5 Sequence Editor

During editing of a sequence, all element positions are evaluated immediately when defined and stored as **constant** values. To modify a sequence it must be selected for editing by the command

```
SEQEDIT,SEQUENCE=old-name;
```

MAD enters editing mode during which it only recognises the sequence editor commands (see Table 5.1), and makes the sequence `old-name` the current sequence being edited. Editing mode is switched off by the command

```
ENDEDIT,NAME=new-name;
```

If `new-name` is non-blank, it becomes the name of the edited sequence, otherwise the modified sequence is stored under its old name.

Command	Purpose
`SELECT` (5.5.1)	Select elements to be affected
`CYCLE` (5.5.2)	Change starting point (cyclic interchange)
`FLATTEN` (5.5.3)	Flatten the sequence
`INSTALL` (5.5.4)	Install new elements
`MOVE` (5.5.5)	Move elements
`REFLECT` (5.5.6)	Reflect the sequence
`REMOVE` (5.5.7)	Remove elements
`REPLACE` (5.5.8)	Replace elements
`ENDEDIT` (5.5)	Leave sequence edit mode

Table 5.1: Commands accepted in editor mode

5.5.1 Selecting Element(s)

Elements are selected by a command

```
SELECT,RANGE=range,CLASS=name,PATTERN=regex,FULL=logical,CLEAR=logical;
```

When the clause `SELECTED` is used in an editor command, that command acts on all elements currently selected. A selection remains active until it is explicitly turned off by

```
SELECT,CLEAR=TRUE;
```

or by an `ENDEDIT` command

```
ENDEDIT;
```

See also details on element selection (see Section 7.2).

5.5.2 Change Start Point

The command

```
CYCLE,START=place;
```

makes a cyclic interchange of all elements in the current edit sequence so as to start at the specified place (see Section 2.9.1). This element should preferrably be a zero-length element like a `MARKER`. All further edit commands refer to the **new** positions.

5.5.3 Flatten the Sequence being Edited

The sequence loaded into the editor can be flattened by the command

```
FLATTEN;
```

This creates a copy of the sequence with all sub-lines and sub-sequences expanded, thus allowing changes also to elements within those nested parts.

5.5.4 Install an Element

New element(s) can be installed in the edited sequence by the commands

```
object-name:class-name,AT=at-expression,SELECTED,attributes;
object-name:class-name,AT=at-expression,FROM=place,attributes;
object-name:class-name,AT=at-expression,attributes;
class-name,AT=at-expression,SELECTED;
class-name,AT=at-expression,FROM=place;
class-name,AT=at-expression;
```

It has the following attributes:

object_name The name of the element to be installed. If the object name is given, attributes may also be specified to override the attributes of the class.

class_name The name of the class from which the element is to be defined.

AT The position where to install the element.

FROM Three cases are possible:

> **SELECTED** New elements are inserted at `AT=at-expression` from each currently selected element.
>
> **FROM=place** A new element is installed at position `AT=at-expression` relative to the (unique) element at `place` (see Section 2.9.1).
>
> **FROM omitted or blank** A new element is installed at the absolute position `AT=at-expression`.
>
> Relative positions may be negative.

The two names `object-name` and `class-name` interact in the same way as for a sequence definition (see Section 5.2). The reference points for elements is defined by the REFER attribute of the `SEQUENCE`.

5.5.5 Move an Element

The commands

```
MOVE,SELECTED,BY=by-expression;
MOVE,ELEMENT=place,BY=by-expression;
MOVE,ELEMENT=place,TO=to-expression;
MOVE,ELEMENT=place,TO=to-expression,FROM=from-name;
```

move element(s) to new location(s). Three cases are possible:

SELECTED,BY=by-expression All currently selected elements are moved by `by-expression`.

ELEMENT=place,BY=by-expression The (unique) element at `place` is moved by `by-expression`.

ELEMENT=place,TO=to-expression The (unique) element at `place` is moved to the absolute position `to-expression`.

ELEMENT=place,TO=to-expression,FROM=from-name The (unique) element at `place` (see Section 2.9.1) is moved to position `to-expression` relative to element `from-name`. A relative position may be negative.

Except in the first case, it is an error to move more than one element with one `MOVE` command. The `MOVE` command must not attempt to change the order of elements in the sequence; elements can not "hop" over each other.

5.5.6 Reflect a Sequence

The command

```
REFLECT;
```

reverses the order of all elements in the sequence currently being edited. All further editing commands must refer to the **new** positions. The sequence members are **not** reflected.

5.5.7 Remove an Element

One or more element(s) can be removed by the commands

```
REMOVE,SELECTED;
REMOVE,ELEMENT=place;
```

Two cases are possible:

SELECTED Removes all currently selected elements,

ELEMENT=place Removes the single element at `place` (see Section 2.9.1). It is an error of `name` occurs more than once in the sequence.

5.5.8 Replace an Element

The commands

```
REPLACE,SELECTED,BY=new-name;
REPLACE,ELEMENT=place,BY=new-name;
```

replace one or more element(s) in the sequence. Two cases are possible:

SELECTED All currently selected elements are replaced by an occurrence of the element `new-name`.

ELEMENT=old-name The (unique) element at `place` (see Section 2.9.1) is replaced by `new-name`. It is an error if `old-name` is not unique.

Example:

```
REPLACE,ELEMENT=QF15,BY=QF17; // replace one element
```

5.5.9 Example for the Sequence Editor

```
SEQ:SEQUENCE,L=79.00;
   B1:  B, AT=19.115;
   SF1: SF,AT=37.42;
   QF1: QF,AT=38.70;
   B2:  B, AT=58.255;
   SD1: SD,AT=76.74;
   QD1: QD,AT=78.20;
END;

B2W:B2,ANGLE=0.1*B2->ANGLE;

EDIT,SEQUENCE=SEQ;
   MOVE,ELEMENT=SF1,TO=-1.27,FROM=QF1;
   MOVE,ELEMENT=SD1,BY=0.01;
   REPLACE,ELEMENT=B2,BY=BS2;
END,NAME=NEWSEQ;
```

This example moves the two sextupoles and replaces the element `B2` by the element `B2W`. The effect of the above `RE-PLACE` command is equivalent to

```
INSTALL,ELEMENT=B2W,AT=0,FROM=B2;
REMOVE,CLASS=B2;
```

In this example a new element `B2W` is installed at the position of `B2` and the the latter is removed. This works, since all positions are evaluated to constants immediately.

5.6 Examples for Beam Lines

5.6.1 CERN SPS Lattice

The CERN SPS lattice may be represented using the following beam elements:

```
QF:QUADRUPOLE,...; // focusing quadrupole
QD:QUADRUPOLE,...; // defocusing quadrupole
B1:RBEND,...;      // bending magnet of type 1
B2:RBEND,...;      // bending magnet of type 2
DS:DRIFT,...;      // short drift space
DM:DRIFT,...;      // drift space replacing two bends
DL:DRIFT,...;      // long drift space
```

The SPS machine is represented by the lines

```
SPS:   LINE=(6*SUPER);
SUPER: LINE=(7*P44,INSERT,7*P44);
INSERT:LINE=(P24,2*P00,P42);
P00:   LINE=(QF,DL,QD,DL);
P24:   LINE=(QF,DM,2*B2,DS,PD);
P42:   LINE=(PF,QD,2*B2,DM,DS);
P44:   LINE=(PF,PD);
PD:    LINE=(QD,2*B2,2*B1,DS);
PF:    LINE=(QF,2*B1,2*B2,DS);
```

In order not to overload the example, small gaps between magnetic elements have been omitted.

5.6.2 LEP Lattice

A preliminary description of LEP has been given in the [7]. Translation of those element sequences to the MAD input format gives:

```
LEP: LINE=(4*SUP);
SUP: LINE=(OCT,-OCT);
OCT: LINE=(LOBS,RFS,DISS,ARC,DISL,RFL,LOBL);
LOBS:LINE=(L1,QS1,L2,QS2,L3,QS3,L4,QS4);
RFS: LINE=(L5,QS5,L5,QS6,L5,2*(QS7,L5,QS8,L5));
DISS:LINE=(QS11,L25,BW,L22,QS12,L25,B4,L22,QS13,L25,B4,
           L22,QS14,L25,B4,L31,QS15,L25,B4,L32,SF,L23,QS16);
ARC: LINE=(L21,B6,L22,SD,L23,QD,
           7*(CELL(SF1,SD1),CELL(SF,SD)),CELL(SF1,SD1),
           L24,B6,L41,QF,L21,B6,L22,SD4,L23,QD,
           7*(CELL(SF4,SD3),CELL(SF3,SD4)),CELL(SF4,SD3),
           L24,B6,L22,SF3,L23);
DISL:LINE=(QL16,L34,B4,L22,QL15,L33,B4,L22,QL14,L25,B4,
           L22,QL13,L25,B4,L22,QL12,L25,BW,L22,QL11);
RFL: LINE=(2*(L5,QL8,L5,QL7),L5,QL6,L5,QL5,L5);
LOBL:LINE=(QL4,L14,QL3,L13,QL2,L12,QL1,L11);
BW:  LINE=(W,L26,W);
B4:  LINE=(B,L26,B);
B6:  LINE=(B,L26,B,L26,B);
CELL(SF,SD):LINE=(L24,B6,L22,SF,L23,QF,L21,B6,L22,SD,L23,QD);
```

Here the element definitions have been left out to save space.

Chapter 6

Physics Commands

6.1 Global Reference Momentum

Before any physics computations are attempted the following command should be entered:

```
P0=real;
```

This command sets the global reference momentum in GeV/c, which is used to compute the magnetic fields from the normalised multipole coefficients. The `BEAM` command (see Section 6.2) then renormalises the multipole coefficients. This mechanism allows sending a beam with a momentum different from the design momentum through a beam line.

6.2 BEAM Command

All MAD commands working on a beam require the setting of various quantities related to this beam. These are entered by a `BEAM` command:

```
label:BEAM,PARTICLE=name,MASS=real,CHARGE=real,
      ENERGY=real,PC=real,GAMMA=real,BCURRENT=real,
      EX=real,EXN=real,EY=real,EYN=real,ET=real,
      KBUNCH=integer,NPART=real,BUNCHED=logical,
      RADIATE=logical,DAMP=logical,QUANTUM=logical;
```

The `label` is opional, it defaults to `UNNAMED_BEAM`. The particle mass and charge are defined by:

PARTICLE The name of particles in the machine. MAD knows the mass and the charge for the following particles:

- POSITRON The particles are positrons (`MASS`=m_e, `CHARGE`=1),
- ELECTRON The particles are electrons (`MASS`=m_e, `CHARGE`=-1),
- PROTON The particles are protons (default, `MASS`=m_p, `CHARGE`=1),
- ANTIPROTON The particles are anti-protons (`MASS`=m_p, `CHARGE`=-1).

For other particle names one may enter:

MASS The particle mass in GeV.

CHARGE The particle charge expressed in elementary charges.

By default the particle momentum is P0 (see Section 6.1). A different value can be defined by one of the following:

ENERGY The total energy per particle in GeV. If given, it must be greater then the particle mass.

PC The momentum per particle in GeV/c. If given, it must be greater than zero.

GAMMA The ratio between total energy and rest energy of the particles $\gamma = E/m_0$. If given, it must be greater than one. If the mass is changed a new value for the energy should be entered. Otherwise the energy remains unchanged, and the momentum and γ are recalculated.

The emittances are defined by:

- EX The horizontal emittance $E_x = \sigma_x^2/\beta_x$ (default: 1 m).
- EY The vertical emittance $E_y = \sigma_y^2/\beta_y$ (default: 1 m).
- ET The longitudinal emittance $E_t = \sigma_e/(p_0 c) \cdot c\sigma_t$ (default: 1 m). The emittances can be replaced by the normalised emittances and the energy spread:
- EXN The normalised horizontal emittance [m]: $E_{xn} = 4\beta\gamma E_x$ (ignored if E_x is given).
- EYN The normalised vertical emittance [m]: $E_{yn} = 4\beta\gamma E_y$ (ignored if E_y is given).
- SIGT The bunch length $c\sigma_t$ [m].
- SIGE The *relative* energy spread $\sigma_e/p_0 c$ [1].

For the time being, only the particle definition (`PARTICLE,MASS,CHARGE,PC,ENERGY,GAMMA`) is used. The other parameters will be implemented as needed when new commands become available.

Chapter 7

Tables

7.1 Introduction

A typical computation in MAD Version 9 follows these steps (see Figure 7.1):

1. Define a beam line using a `LINE` (see Section 5.1) or `SEQUENCE` (see Section 5.2) command.
2. Define a beam with using `BEAM` (see Section 6.2) command.
3. If desired, define imperfections on the defined beam line, using one of the error commands (see Section 10).
4. If desired, attach special integrators to selected elements of the defined beam line, using `SETINTEGRATOR` (see Section 7.3) commands.
5. If desired, perform an orbit correction on the defined beam line, using `THREADBPM` (see Section 7.5) or `MICADO` (see Section 7.6) command.
6. Define a "Table Object" using one of the table (see Section 7) commands. This selects a `RANGE` (see Section 2.9.2) of a beam line, and defines an algorithm to fill the table, e. g. with the lattice functions, the accumulated transfer map from the beginning of the range to the current position, or the `SURVEY` data (see Section 7.4). A table remains in memory and is recomputed as required. It is erased only if one of the contained elements or beam lines is changed.
7. If desired, apply the match module (see Section 9) to adjust parameters in the machine. The match module can work on any number of tables simultaneously.
8. Use a `LIST` (see Section 7.9) command to print the contents of any table.

7.2 Element Selection

Many MAD commands allow for the possibility to process a subset of the elements occurring in a beam line (see Section 5). For this purpose each beam line (see Section 5) and table (see Section 7) has a selection flag for each element. The `SELECT` command may be used to manipulate these flags. It affects the following commands:

- `EALIGN` (see Section 10.3),
- `EFIELD` (see Section 10.4),
- `EFCOM` (see Section 10.4),
- `EPRINT` (see Section 10.5),
- `ESAVE` (see Section 10.6),
- `SURVEY` (see Section 7.4),
- `TWISS` (see Section 7.7),

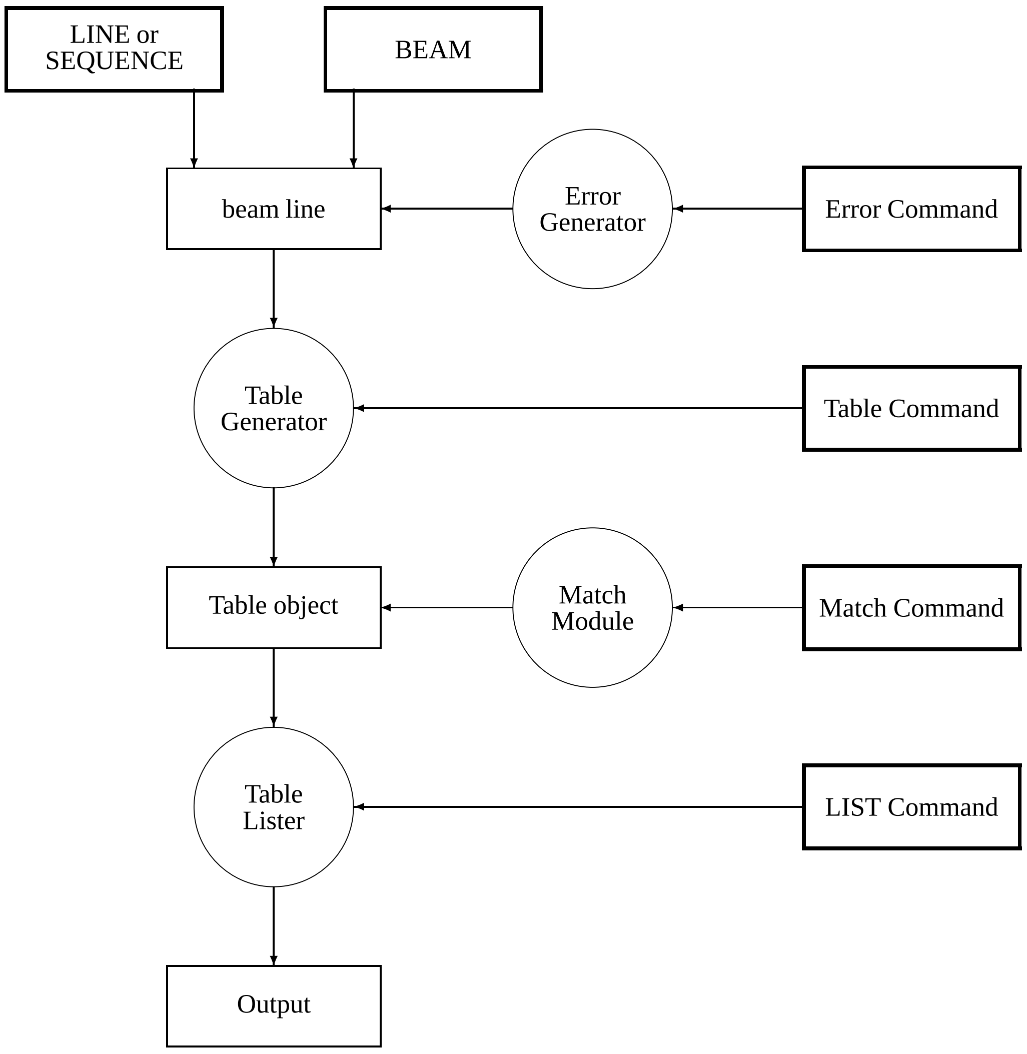

Figure 7.1: Schematic view of the Interaction between Objects in MAD

- `ATTLIST` (see Section 7.8),
- `LIST` (see Section 7.9),
- `EIGEN` (see Section 7.9.2),
- `ENVELOPE` (see Section 7.9.2),
- `MATRIX` (see Section 7.9.2),
- `TWISS3` (see Section 7.9.2).

Three formats are recognised for the command:

```
SELECT,LINE=name,FULL;
SELECT,LINE=name,CLEAR;
SELECT,LINE=name,RANGE=range,CLASS=name,TYPE=name,PATTERN=regular_expression;
```

All three forms require the parameter

LINE The label of a previously defined beam line (see Section 5) or table (see Section 7) on which the selection is to be done (no default).

The first format sets the selection flag for all elements in the beam line and the second format clears the selection flags for all elements.

The third format keeps all existing selections and additionally marks those elements as selected which belong to the intersection of the following four sets:

RANGE If RANGE is omitted, the first set contains all elements; if it is given, the first set is limited to the named range in the line. The default is equivalent to RANGE=#S/#E.

CLASS The label (see Section 2.2) of an element class (default: blank). If CLASS is omitted, the second set contains all elements; if it is given, the set contains only the elements derived directly or indirectly from the named class.

TYPE If TYPE is omitted, the third set contains all elements; if it is given, the set contains only the elements whose TYPE attribute is equal to the given name.

PATTERN If PATTERN is omitted, the fourth set contains all elements; if it is given, the regular expression (see Section 2.12) is applied to all element names, and the set contains only the matching elements.

The effect of subsequent SELECT commands produces the union of all selections. If a fresh selection is desired, precede the new SELECT command with the command

```
SELECT,LINE=name,CLEAR;
```

Example:

```
SELECT,LINE=X,RANGE=IP1/IP2;                             // (1)
SELECT,LINE=X,CLASS=BB;                                  // (2)
SELECT,LINE=X,PATTERN=".*\.L1";                          // (3)
SELECT,LINE=X,RANGE=IP1/IP2,CLASS=BB,PATTERN=".*\.L1";   // (4)
SELECT,LINE=LHC,CLASS=IP;                                // (5)
T:TWISS,LINE=LHC;                                        // (6)
SELECT,LINE=T,CLASS=QUADRUPOLE;                          // (7)
TWISS3,TABLE=T,FILE=name,TFS;                            // (8)
U:TWISS,LINE=LHC;                                        // (9)
```

Command (1) selects all elements from IP1 to IP2, both included. Command (2) selects all elements in the ring which belong to or are derived from class BB. Command (3) selects all elements in the ring whose names end .L1. Command (4) is the most restrictive, it selects all elements between IP1 and IP2 which are derived from class BB and whose names end in .L1.

Command (5) sets the selections flags on all interaction points in line LHC. The selection is transmitted to command (6), which generates the TWISS table T lists the default columns for the interaction points. Command (7) sets additional selection flags on the table T for all quadrupoles, and command (8) lists the Mais-Ripken functions for all interaction points and all quadrupoles. Command (9) gets the selection from line LHC, i. e. the selection concerns only the interaction points.

7.3 Attach Special Integrators

By default the algorithm applied to each element in the machine is defined by the table command. It may be overridden for selected elements by applying a SETINTEGRATOR command to attach a special integrator. It has the form

```
SETINTEGRATOR,LINE=name,TYPE=name,SLICES=integer;
```

with the meaning

LINE The name of the beam line to be affected.

TYPE The integrator type name. For the time being, only the name MPSPLITINTEGRATOR is known. This integrator type replaces any multipole-like element (dipole, quadrupole, sextupole, octupole, general multipole) by one or several thin lenses.

SLICES The number of thin lenses to be used. For one to four slices, the TEAPOT method of thin multipole slicing is used, for more slices, the thin lenses are placed in equidistant places.

The command runs through all elements in the beam line, and attaches an integrator as specified by TYPE to all selected elements.

Table 7.1: Global read-only values in a SURVEY table

Variable	Unit	Name
Total arc length	m	L
Final X coordinate	m	X
Final Y coordinate	m	Y
Final Z coordinate	m	Z
Final angle theta	rad	THETA
Final angle phi	rad	PHI
Final angle psi	rad	PSI

7.4 Compute Geometric Layout

The SURVEY command builds a table containing the geometric layout of the machine. The table can be referred to by its label for later manipulation. It has the following read/write attributes:

```
label:SURVEY,LINE=name,RANGE=range,REVERSE=logical,STATIC=logical,
      X0=real,Y0=real,Z0=real,THETA0=real,PHI0=real,PSI0=real;
label:SURVEY,LINE=name,RANGE=range,REVERSE=logical,STATIC=logical,
      INIT=table-row;
```

Its parameter list specifies the initial position and orientation of the reference orbit in the global coordinate system (see Figure 1.2) (X, Y, Z). Omitted attributes assume zero values. Valid attributes are:

LINE The label of a previously defined beam line (no default).

RANGE If this attribute is given, the table is restricted to the range given.

STATIC If this attribute is true, the table is filled at definition time, and it is then treated as frozen, even if parameters of the machine change.

INIT If this attribute is given, the initial position and direction is taken from the specified row of another SURVEY table. If this attribute is omitted, the initial position and direction is constructed from the attributes listed below.

X0 The initial X coordinate [m].

Y0 The initial Y coordinate [m].

Z0 The initial Z coordinate [m].

THETA0 The initial angle theta [rad].

PHI0 The initial angle phi [rad].

PSI0 The initial angle psi [rad].

Example:

```
GEOMETRY:SURVEY,LINE=RING;
```

This example computes the machine layout for the line RING with zero initial conditions and saves the result under the name GEOMETRY.

The SURVEY table has some read-only attributes (see Table 7.1):

In each position there are table values (see Table 7.2), each of which is accessible with the syntax (see Section 2.8.6)

```
table-name "@" place "->" column-name
```

The azimuth, elevation angle, and roll angle are defined as the components of a vector. This vector points in the direction of the rotation axis and has the rotation angle as its length. The direction cosines are the components of the unit vectors along the local coordinate axes, expressed in the global coordinate system.

The default column set for the command LIST,ALL (see Section 7.9) contains the element name, the arc length, the global positions and the three angles.

Table 7.2: Column values available in a `SURVEY` table

Variable	Unit	Name
Arc length from beginning	m	`S`
Global X coordinate	m	`X`
Global Y coordinate	m	`Y`
Global Z coordinate	m	`Z`
Azimuth	rad	`THETA`
Elevation angle	rad	`PHI`
Roll angle	rad	`PSI`

Direction cosines for local axes	Local axis		
Global direction	`x`	`y`	`s`
`X`	`W11`	`W12`	`W13`
`Y`	`W21`	`W22`	`W23`
`Z`	`W31`	`W32`	`W33`

7.5 Orbit Threader

When the closed orbit search in a `MICADO` (see Section 7.6) or `TWISS` (see Section 7.7) command fails due to machine imperfections, it may be possible to thread the orbit through the machine using the command

```
THREADBPM,LINE=name,BEAM=name,RANGE=range,TOL=real,LISTC=logical,LISTM=logical;
```

This command has the attributes:

LINE The label of a previously defined beam line (no default).

BEAM The label of a previously defined beam (default `UNNAMED_BEAM`). This is used to define the type and reference momentum for the particles to be sent through the line.

RANGE If this attribute is given, the table is restricted to the range given.

TOL The orbit is observed at all beam position monitors (BPM). When the orbit deviation exceeds `TOL`, a correction is attempted. MAD backtracks to find another monitor and two correctors and tunes the latter so as to adjust the readings of the two monitors to zero.

LISTC If true, list the corrector settings after correction.

LISTM If true, list the monitor readings after correction.

7.6 Closed Orbit Correction

Closed orbit correction with the MICADO algorithm is initiated by the command

```
MICADO,LINE=name,BEAM=name,RANGE=range,METHOD=name,TOL=real,
       ITERATIONS=integer,CORRECTORS=integer,PLANE=name,
       LISTC1=logical,LISTM1=logical,LISTC2=logical,LISTM2=logical;
```

This command has the attributes:

LINE The label of a previously defined beam line (no default).

BEAM The label of a previously defined beam (default `UNNAMED_BEAM`). This is used to define the type and reference momentum for the particles to be sent through the line.

RANGE If this attribute is given, the table is restricted to the range given.

METHOD This attribute specifies the method to be used for tracking the orbit. Known names are:

THIN Use thin lens approximations.

LINEAR Use linear map approximation (around the closed orbit). This is the default.

THICK Use finite-length lenses.

CORRECTORS The maximum number of correctors to be used in each iteration.

TOL The tolerance for the r.m.s. closed orbit.

ITERATIONS The number of iterations desired. When this number is zero, the orbit is only checked but not corrected. Each iteration is terminated when the imposed number of correctors has been used or when the tolerance is reached.

PLANE The plane to be corrected. Known names are:

X Correct the horizontal plane only.

Y Correct the vertical plane only.

BOTH Correct both planes (default).

LISTC1 If true, list the corrector settings before each iteration.

LISTM1 If true, list the monitor readings after each iteration.

LISTC2 If true, list the corrector settings after correction.

LISTM2 If true, list the monitor readings after correction.

7.7 Lattice Function Tables

Two commands are available to generate tables of lattice functions:

```
TWISS,LINE=name,BEAM=name,RANGE=range,STATIC=logical,METHOD= name,
      REVBEAM=logical,REVTRACK=logical;
TWISSTRACK,BEAM=name,RANGE=range,STATIC=logical,METHOD= name,
      INIT=table-row,REVBEAM=logical,REVTRACK=logical;
```

Either command builds a “twiss” table containing the lattice functions for a range of a previously defined beam line (see Section 5). The first command, `TWISS`, uses the periodic solution for that range, while the second command, `TWISSTRACK` a selected line of another twiss table. The table can be referred to by its label for later manipulation. Both commands have the following read/write attributes:

LINE The label of a previously defined beam line (no default).

BEAM The label of a previously defined beam (default `UNNAMED_BEAM`). This is used to define the type and reference momentum for the particles to be sent through the line.

RANGE If this attribute is given, the table is restricted to the range given.

STATIC If this attribute is true, the table is filled at definition time, and it is then treated as frozen, even if parameters of the machine change.

METHOD This attribute specifies the method to be used for filling the table. Known names are:

THIN Use thin lens approximations.

LINEAR Use linear map approximation (around the closed orbit). This is the default.

THICK Use finite-length lenses.

REVBEAM If true, the beam is assumed to run backwards through the beam line (from $s = C$ to $s = 0$).

REVTRACK If true, the calculation proceeds in the direction opposite to the beam direction. This attribute may be used in matching, when tracking lattice functions from a known position back to a position to be used to adjust an insertion.

The `TWISS` command has the following read-only attributes:

L The total arc length [m].

Table 7.3: Global read-only values in a `TWISS` table

Variable	Unit	Name
Revolution frequency	Hz	`FREQ`
Tune for mode 1	1	`QX`
Tune for mode 2	1	`QY`
Tune for mode 3	1	`QS`
Energy loss per turn	MeV	`U0`
Damping partition number for mode 1	1	`JX`
Damping partition number for mode 2	1	`JY`
Damping partition number for mode 3	1	`JE`

FREQ0 The computed revolution frequency in MHz.

QX The computed tune for mode 1.

QY The computed tune for mode 2.

QS The computed tune for mode 3.

U0 The computed energy loss per turn [MeV].

JX The computed damping partition number for mode 1.

JY The computed damping partition number for mode 2.

JE The computed damping partition number for mode 3.

The values related to damping exist only when the flag `DAMP` has been set.

The `TWISSTRACK` can be initialised by the following attribute:

INIT If this attribute is given, the initial position and direction is taken from the specified row of another `TWISS` table.

Example:

```
LATFUN:TWISS,LINE=CELL;
```

This example computes the lattice functions for `CELL` with zero initial conditions and saves the result under the name `LATFUN`.

A table generated by `TWISS` (not `TWISSTRACK`) also has some read-only attributes (see Table 7.3):

In each the `TWISS` table stores the closed orbit and the transfer map from the beginning to this position. Several column values (see Table 7.5), as well as the eigenvectors (see Table 7.6), second moments (see Table 7.7), and transfer matrices (see Table 7.8) can be derived from this information These quantities are accessible with the syntax (see Section 2.8.6)

```
table-name "@" place "->" column-name
```

The Courant-Snyder lattice functions are calculated in a way analogous to the lattice functions given by [6]. After partitioning the eigenvector matrix into two by two blocks it can be factored as

$$E = \begin{pmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{pmatrix} = RW = \begin{pmatrix} r_1 I & R_{12} & R_{13} \\ R_{21} & r_2 I & R_{23} \\ R_{31} & R_{32} & r_3 I \end{pmatrix} \begin{pmatrix} W_1 & 0 & 0 \\ 0 & W_2 & 0 \\ 0 & 0 & W_3 \end{pmatrix},$$

where the r_i are scalars and R and W are symplectic matrices. This implies

$$r_i = \sqrt{|R_{ii}|}, \qquad W_i = R_{ii}/r_i \rightarrow |W_i| = 1, \qquad R_{ik} = E_{ik} W_i^{-1}.$$

The lattice functions are calculated from the diagonal blocks W_i. Note that the matrix R which defines the coupling between planes is not available for output, and that the formalism breaks down when $r_i = 0$. The default column set

Table 7.4: Column values in a TWISS table

Variable	Unit	Name
Arc length from beginning	m	S
Horizontal position for closed orbit	m	XC
Horizontal momentum for closed orbit	1	PXC
Vertical position for closed orbit	m	YC
Vertical momentum for closed orbit	1	PYC
Longitudinal position for closed orbit	m	TC
Longitudinal momentum for closed orbit	1	PTC
Dispersion for horizontal position	m	DX
Dispersion for horizontal momentum	1	DPX
Dispersion for vertical position	m	DY
Dispersion for vertical momentum	1	DPY
Dispersion for longitudinal position	m	DT
Dispersion for longitudinal momentum	1	DPT

		Plane		
Courant-Snyder Functions	Unit	X	Y	T
Beta-function	m	BETX	BETY	BETT
Alpha-function	1	ALFX	ALFY	ALFT
Phase	1	MUX	MUY	MUT

Table 7.5: Some of the column values in a TWISS3 table

			Plane		
Mais-Ripken Functions	Unit	Mode	X	Y	T
Beta-function	m	1	BETX1	BETY1	BETT1
		2	BETX2	BETY2	BETT2
		3	BETX3	BETY3	BETT3
Alpha-function	1	1	ALFX1	ALFY1	ALFT1
		2	ALFX2	ALFY2	ALFT2
		3	ALFX3	ALFY3	ALFT3
Gamma-function	1/m	1	GAMX1	GAMY1	GAMT1
		2	GAMX2	GAMY2	GAMT2
		3	GAMX3	GAMY3	GAMT3

contains the element name, the arc length, the beta, alpha and mu functions for the transverse planes, the closed orbit position, and the horizontal dispersion.

The Mais-Ripken functions have been introduced by [10].

The second moments (beam envelope) are defined in the sense of [2]:

$$\Sigma_{ik} = E_x(E_{i1}E_{k1} + E_{i2}E_{k2}) + E_y(E_{i3}E_{k3} + E_{i4}E_{k4}) + E_t(E_{i5}E_{k5} + E_{i6}E_{k6}).$$

7.8 Listing Element Attributes

Most element attributes can be listed for a beam line or table. The elements for which the attributes should be listed are selected by a `SELECT` command (see Section 7.2). The attributes are specified by the command

```
ATTLIST,LINE=name,FILE=name,COLUMN={string,string,...,string};
```

Most element attributes can be listed as columns. At present, MAD-9 does not recognize `ATTRIBUTES` or attributes of a `BEAMBEAM` element. Example:

Table 7.6: Eigenvector components in a EIGEN table

Eigenvector		Mode 1		Mode 2		Mode 3	
Component	Unit	real	imag	real	imag	real	imag
X-component	m	E11	E12	E13	E14	E15	E16
PX-component	1	E21	E22	E23	E24	E25	E26
Y-component	m	E31	E32	E33	E34	E35	E36
PY-component	1	E41	E42	E43	E44	E45	E46
T-component	m	E51	E52	E53	E54	E55	E56
PT-component	1	E61	E62	E63	E64	E65	E66

Table 7.7: Second moments <X1,X2> in a ENVELOPE table

Second moment	Second variable X2					
First variable X1	X	PX	Y	PY	T	PT
X	S11	S12	S13	S14	S15	S16
PX	S21	S22	S23	S24	S25	S26
Y	S31	S32	S33	S34	S35	S36
PY	S41	S42	S43	S44	S45	S46
T	S51	S52	S53	S54	S55	S56
PT	S61	S62	S63	S64	S65	S66

```
 SELECT, LINE=LIN1, FULL;
 ATTLIST, LINE=LIN1, FILE=TERM,
   COLUMN={NAME,CLASS,KEYWORD,TYPE,LENGTH,K0L,UNKNOWN};
```

This may produce the following output:

```
@ TYPE     %s  ATTRIBUTE
@ LINE     %s  LIN1
@ DATE     %s  11/10/1999
@ TIME     %s  14.08.14
@ ORIGIN   %s  MAD_9.3
@ COMMENT  %s  ""
* NAME CLASS KEYWORD TYPE L K0L UNKNOWN
$ %s %s %s %s %le %le %le
  #S MARKER MARKER Null 0 0 0
  L4 DRIFT DRIFT Null 1.57 0 0
  B1 RBEND RBEND Null 35.09 0.0113061 0
  L2 DRIFT DRIFT Null 0.56 0 0
  SF SEXT SEXTUPOLE Null 0.4 0 0
  L3 DRIFT DRIFT Null 0.28 0 0
  QF QUAD QUADRUPOLE MQ 1.6 0 0
  L1 DRIFT DRIFT Null 1.21 0 7
  B1 RBEND RBEND Null 35.09 0.0113061 0
  L2 DRIFT DRIFT Null 0.56 0 0
  SD SEXT SEXTUPOLE Null 0.76 0 0
  L3 DRIFT DRIFT Null 0.28 0 0
  QD QUAD QUADRUPOLE MQ 1.6 0 0
  #E MARKER MARKER Null 0 0 0
```

As it can be seen, empty strings are printed as the word “Null”.

Table 7.8: Transfer matrix elements (X2,X1) in a MATRIX table

Matrix Element	Second variable X2					
First variable X1	X	PX	Y	PY	T	PT
X	R11	R12	R13	R14	R15	R16
PX	R21	R22	R23	R24	R25	R26
Y	R31	R32	R33	R34	R35	R36
PY	R41	R42	R43	R44	R45	R46
T	R51	R52	R53	R54	R55	R56
PT	R61	R62	R63	R64	R65	R66

7.9 Listing a Table

Five commands exist which permit listing of a previously generated table. All of these have the option of printing on a file or on the terminal, and using a human-readable or a TFS format.

7.9.1 LIST Command

The most general of these commans is

```
LIST,TABLE=name,FILE=name,ALL=logical,COLUMN={expression,...,expression};
```

The attributes of this command are:

TABLE The name of the table to be listed (no default). The rows to be listed can be selected by a SELECT (see Section 7.2) command.

FILE The name of the file to be written (default LIST). The table is written in the TFS format (see Section **??**) known from MAD Version 8. If the file name is TERM, output goes to the terminal.

TFS If true, the format is in TFS format, otherwise in a format meant for viewing with a fixed-width fount ("line-printer" style).

ALL If true, the column selection is by default, and the column width and precision is selected by MAD. In this case COLUMN is ignored.

COLUMN A list of column descriptors. Each column descriptor consists of a real expression (see Section 2.6), which may also contain one or more names of columns as operands. If a name refers both to a column and to a global parameter, the column takes precedence. The expression may optionally be followed by a format indication of the form:

```
:width
:width:precision
```

Both width and precision are unsigned integers. width denotes the column width in characters (default 12), and precision controls the number of digits in the column (default width-4).

Example:

```
list,table=su1,file=term,column={x,y,z,s*z:16:12};
```

7.9.2 "Canned" List Commands

Certain predefined column sets are obtainable with the commands

```
EIGEN,TABLE=name,FILE=name,TFS=logical;
ENVELOPE,TABLE=name,FILE=name,TFS=logical;
MATRIX,TABLE=name,FILE=name,TFS=logical;
TWISS3,TABLE=name,FILE=name,TFS=logical;
```

These commands provide the following output:

EIGEN The name of the element, the arc length, the closed orbit, and the eigenvectors (see Table 7.6). The choice is simalar to the MAD-8 command of the same name.

ENVELOPE The name of the element, the arc length, the closed orbit, and the beam envelope (see Table 7.7). The choice is simalar to the MAD-8 command of the same name.

MATRIX The name of the element, the arc length, the closed orbit, and the accumulated matrix (see Table 7.8). There is no MAD-8 equivalent.

TWISS3 The name of the element, the arc length, the closed orbit, and the Mais-Ripken functions (see Table 7.5). The choice is simalar to the MAD-8 command of the same name.

The attributes of these commands are:

TABLE The name of the table to be listed (no default). The rows to be listed can be selected by a `SELECT` (see Section 7.2) command.

FILE The name of the file to be written (default `LIST`). The table is written in the TFS format (see Section **??**) known from MAD Version 8. If the file name is `TERM`, output goes to the terminal.

TFS If true, the format is in TFS format, otherwise in human-readable format.

Chapter 8

Action Commands

8.1 Normal-Form Analysis

The two commands

```
DYNAMIC,LINE=name,BEAM=name,FILE=string,ORDER=integer;
STATIC, LINE=name,BEAM=name,FILE=string,ORDER=integer;
```

both evaluate the truncated Taylor series map for one turn up to order `ORDER` and perform normal-form analysis on the result. Both have the following attributes:

LINE The label of a **beam line or sequence** (see Section 5) defined previously (no default). Its transfer map will be evaluated and analysed to the desired order.

BEAM The label of a `BEAM` command (see Section 6.2) defined previously (default = `UNNAMED_BEAM`). It defines the charge, kinetic energy, rest mass, and reference velocity for the reference particle.

FILE The name of the file to be written (default = ”`dynamic`” or ”`static`” respectively).

ORDER The maximum order for the map.

8.1.1 Normal-Form Analysis for Dynamic Map

The command interprets the transfer map as dynamic, i.e. there is synchrotron motion with an average momentum p_s.

8.1.2 Normal-Form Analysis for Static Map

The command assumes that there are no cavities, and interprets the transfer map as static, i.e. the particles have constant momentum. The `STATIC` command also finds the fixed point of the map, that is the variation of the closed orbit with momentum.

Chapter 9

Matching Module

Command	Purpose
`MATCH` (9.1)	Enter matching mode
`name=expression` (3.4.1)	Parameter relation
`CONSTRAINT` (9.3)	Impose matching constraint
`VARY` (9.2)	Vary parameter
`OPTION` (9.4)	Set print level
`LMDIF` (9.5)	Minimisation by gradient method
`MIGRAD` (9.5)	Minimisation by gradient method
`SIMPLEX` (9.5)	Minimisation by simplex method
`ENDMATCH` (9.1)	Leave matching mode
`SURVEY` (7.4)	Define a survey table
`TWISS` (7.7)	Define a periodic lattice function table
`TWISSTRACK` (7.7)	Define a lattice function table

Table 9.1: Commands accepted in Matching Mode

9.1 Matching Mode

To initiate a matching operation, the command

```
MATCH;
```

is entered. MAD then enters "matching mode", in which it accepts only the matching commands (see Table 9.1). Most of the procedures have originated in the [8] program.

This command must be followed by commands to

- Define one or more tables (see Section 7) to be matched. The tables can also be defined before the `MATCH` command.
- Define matching constraints (see Section 9.3) on these tables.
- Define the variables (see Section 9.2) to be adjusted.
- Select the verbosity (see Section 9.4) of output.
- Initiate matching (see Section 9.5).

The `ENDMATCH` command terminates matching mode and deletes all tables related to a matching run:

```
ENDMATCH;
```

9.2 Variable Parameters

A parameter to be varied is specified by the command:

```
VARY,NAME=variable,STEP=real,LOWER=real,UPPER=real;
```

It has four attributes:

NAME: The name of the parameter (see Section 3.4.1) or attribute (see Section 2.6) to be varied.

STEP: The approximate initial step size for varying the parameter. If the step is not entered, MAD tries to find a reasonable step, but this may not always work.

LOWER: Lower limit for the parameter (optional),

UPPER: Upper limit for the parameter (optional).

Upper and/or lower limits can also be imposed via the `CONSTRAINT` command (see Section 9.3), which is usually faster than entering limits on the `VARY` command. Examples:

```
VARY,PAR1,STEP=1.0E-4       // vary global parameter PAR1
VARY,QL11->K1,STEP=1.0E-6   // vary attribute K1 of the QL11
VARY,Q15->K1,STEP=0.0001,LOWER=0.0,UPPER=0.08 // vary with limits
```

If the upper limit is smaller than the lower limit, the two limits are interchanged. If the current value is outside the range defined by the limits, it is brought back to range. After a matching operation all varied attributes retain their last value. They are never reset to an old value. If a matching variable depends on another variable, this dependency is broken by the `VARY` command. Example:

```
P1:=10.0-P2
```

Both `P1` and `P2` may be varied. If `P1` is varied however, its dependence on `P2` is broken.

9.3 Constraints

All constraints are imposed by the `CONSTRAINT` command:

```
CONSTRAINT,left relation right,WGT=weight;
```

In this command `relation` is one of the relational operator $==$, $<$, or $>$. All of `left`, `right`, and `weight` are vector expressions (see Section 3.4.3). All three must evaluate to the same number n of components. The command is interpreted as n matching constraints like `left[i] relation right[i]` with a weight of `weight[i]`, where i runs from one to n. A wide spectrum of vector expressions are possible.

Examples. Single constraint:

```
CONSTRAINT,T1@M[3]->BETX==120;WGT=1;
```

Two constraints in the same point:

```
CONSTRAINT,ROW(T1,M[3],{BETX,BETY})=={120,120},WGT=TABLE(2,1);
```

Maximum over a range:

```
CONSTRAINT,COLUMN(T1,BETX,#S/#E)<200,WGT=1;
```

Coupling between two points:

```
CONSTRAINT,ROW(T1,M1,{BETX,BETY})==ROW(T1,M2,{BETX,BETY}),WGT=TABLE(2,1);
```

Interchange of `BETX` and `BETY`:

```
CONSTRAINT,ROW(T1,M1,{BETX,BETY})==ROW(T1,M2,{BETY,BETX}),WGT=TABLE(2,1);
```

For complex matching conditions in Version 8 of MAD one had to introduce ancillary variables. Several quantities could then be tied to such a variable, resulting in coupling between these quantities. This is no longer needed in Version 9, as can be seen from these examples.

9.4 Matching Output Level

The `OPTION` command sets the output level for matching:

```
OPTION,LEVEL=integer;
```

Recognised level numbers are:

0 Minimum printout: Messages and final values only,

1 Normal printout: Messages, initial and final values,

2 Like `LEVEL=1`, plus every tenth new minimum found,

3 Like `LEVEL=2`, plus every new minimum found,

4 Like `LEVEL=3`, plus eigenvalues of covariance matrix (`MIGRAD` method only).

Example:

```
OPTION,LEVEL=2;
```

9.5 Matching Methods

9.5.1 LMDIF, Gradient Minimisation

The `LMDIF` command minimises the sum of squares of the constraint functions using their numerical derivatives:

```
LMDIF,CALLS=integer,TOLERANCE=real;
```

It is the fastest minimisation method available in MAD. The command has two attributes:

CALLS The maximum number of calls to the penalty function (default: 1000).

TOLERANCE The desired tolerance for the minimum (default: 10^{-6}).

Example:

```
LMDIF,CALLS=2000,TOLERANCE=1.0E-8;
```

9.5.2 MIGRAD, Gradient Minimisation

The `MIGRAD` command minimises the penalty function using its numerical derivatives of the sum of squares:

```
MIGRAD,CALLS=integer,TOLERANCE=real,STRATEGY=1;
```

The command has three attributes:

CALLS The maximum number of calls to the penalty function (default: 1000).

TOLERANCE The desired tolerance for the minimum (default: 10^{-6}).

STRATEGY A code for the strategy to be used (default: 1). The user is referred to the MINUIT manual for [8].

Example:

```
MIGRAD,CALLS=2000,TOLERANCE=1.0E-8;
```

9.5.3 SIMPLEX, Minimisation by Simplex Method

The `SIMPLEX` command minimises the penalty function by the simplex method:

```
SIMPLEX,CALLS=integer,TOLERANCE=real;
```

The user is referred to the MINUIT manual for [8]. The command has two attributes:

CALLS The maximum number of calls to the penalty function (default: 1000).

TOLERANCE The desired tolerance for the minimum (default: 10^{-6}).

Example:

```
SIMPLEX,CALLS=2000,TOLERANCE=1.0E-8;
```

9.6 Matching Examples

9.6.1 Simple Periodic Beam Line

Match a simple cell with given phase advances:

```
// Some definitions:
QF:     QUADRUPOLE,...;
QD:     QUADRUPOLE,...;
EM:     MARKER;
CELL1:  LINE=(...,QF,...,QD,...,EM);
P0 = ...;
BEAM1:  BEAM,PC=P0;
TW1:    TWISS,LINE=CELL1,BEAM=BEAM1,METHOD=LINEAR;

// Match CELL1:
MATCH;
  VARY,NAME=QD->K1,STEP=0.01;
  VARY,NAME=QF->K1,STEP=0.01;
  CONSTRAINT,ROW(TW1,EM,{MUX,MUY})=={0.25,1/6};
  LMDIF,CALLS=200;
ENDMATCH;
```

9.6.2 Insertion Matching

Match an insertion `INSERT` to go between two different cells `CELL1` and `CELL2`:

```
// Some definitions:
BI:     MARKER;
EI:     MARKER;
BM:     MARKER;
EM:     MARKER;
CELL1:  LINE=(...);
CELL2:  LINE=(...);
INSERT: LINE=(...);
MAIN:   LINE=(BM,CELL1,BI,INSERT,EI,CELL2,EM);
P0 = ...;
BEAM1:  BEAM,PC=P0;
C1:     TWISS,LINE=CELL1,BEAM=BEAM1,METHOD=LINEAR,RANGE=BM/BI,STATIC;
C1:     TWISS,LINE=CELL2,BEAM=BEAM1,METHOD=LINEAR,RANGE=EI/EM,STATIC;
MATCH;
  INS:  TWISS,LINE=INSERT,BEAM=BEAM1,METHOD=LINEAR,RANGE=BI/EI;
  VARY,...;
  CONSTRAINT,
    ROW(C1,BI,{BETX,ALFX,BETY,ALFY})==ROW(INS,BI,{BETX,ALFX,BETY,ALFY}),
    WGT={10,1,10,1};
  CONSTRAINT,
    ROW(C2,EI,{BETX,ALFX,BETY,ALFY})==ROW(INS,EI,{BETX,ALFX,BETY,ALFY}),
    WGT={10,1,10,1};
  CONSTRAINT,ROW(INS,EI,{MUX,MUX})={...,...},WGT={10,10};
  SIMPLEX;
  MIGRAD;
ENDMATCH;
```

This matches the optical functions `BETX,ALFX,BETY,ALFY` to fit together at the markers `BI,EI`, and the phase advances over the insertion.

9.6.3 A Matching Example for LHC

In this case we impose constraints on maximum values of BETX. This is done by means of the vector to scalar function VMAX (see Section 3.4.3) with the vector-valued function COLUMN (see Section 3.4.3).

```
MATCH;
  BIR2:TWISS,LINE=LHC,RANGE=CELL23;
  IR2:TWISSTRACK,LINE=LHC,RANGE=S.DS.L2/E.DS.R2,INIT=BIR2@S.DS.L2,
       METHOD=LINEAR;

  // Some constraints
  ...;

  // (Part of the ) beta limitation in the dispersion suppressor
  CONSTRAINT,VMAX(COLUMN(IR2,BETX,S.DS.L2/Q7.L2)) < BETMAX,WGT=10;
  CONSTRAINT,VMAX(COLUMN(IR2,BETX,Q7.L2/Q5A.L2))  < 350.0, WGT=10;
  CONSTRAINT,VMAX(COLUMN(IR2,BETX,Q5A.R2/Q7.R2))  < 350.0, WGT=10;
  CONSTRAINT,VMAX(COLUMN(IR2,BETX,Q7.R2/E.DS.R2)) < BETMAX,WGT=10;

  // Some strength limits
  CONSTRAINT,KQ4.L2 > -9.6E-3,WGT=1;
  CONSTRAINT,KQ5.L2 <  9.6E-3,WGT=1;
  CONSTRAINT,KQ6.L2 > -9.6E-3,WGT=1;
  CONSTRAINT,KQ7.L2 <  9.6E-3,WGT=1;

  // Vary commands
  ...;

  // Matching method
  OPTION,LEVEL=3;
  LMDIF,CALLS=200,TOLERANCE=1.E-16;
ENDMATCH;
```

Chapter 10

Error Definitions

10.1 Error Commands

Command	Purpose
`name=expression` (3.4.1)	Parameter relation
`ERROR` (10.2)	Start error mode
`SELECT` (7.2)	Select elements to be affected
`EALIGN` (10.3)	Define misalignment errors
`EFIELD` (10.4)	Define multipole field errors (by amplitude and rotation)
`EFCOMP` (10.4)	Define multipole field errors (by components)
`EPRINT` (10.5)	Print error definitions
`ESAVE` (10.6)	Save error definitions to file
`ENDERROR` (10.2)	End error mode

Table 10.1: Error Commands

10.2 Error Mode

For all error-defining commands (see Table 10.1), the beam line (see Section 5.1) or sequence (see Section 5.2) to be affected must be specified with the command

```
ERROR,LINE=name,SEED=real,ADD=logical,CLEAR=logical;
```

It has the following attributes:

LINE The line to be affected.

SEED Selects a particular sequence of random values. A SEED value is an integer in the range [0...999999999] (default: 123456789). SEED can be an expression. See also: random values (see Section 2.8.5).

ADD If this value is entered as true, the subsequent error definitions are added to any existing ones. Otherwise, the subsequent error definitions overwrite any existing ones.

CLEAR If this value is true, all errors are erased from the given line.

After this command MAD enters "error mode", in which it accepts only the error defining commands (see Table 10.1). Normal mode is resumed by entering the command

```
ENDERROR;
```

It is possible to assign alignment errors and field errors to single beam elements or to ranges of beam elements. Errors can be entered both with given or with random values.

10.3 Define Misalignments

Alignment errors are defined by the `EALIGN` command. The misalignments refer to a local reference system (see Figure 1.1) for a perfectly aligned machine, placed half-way through the element on the design orbit. Possible misalignments are displacements along the three coordinate axes, and rotations about the coordinate axes. Alignment errors can be assigned to all beam elements except drift spaces. The effect of misalignments is treated in a linear approximation.

Beam position monitors (see Section 4.16) can be given read errors in both horizontal and vertical planes. Monitor read errors are ignored for all other elements. Each new `EALIGN` statement replaces the misalignment errors for all elements in its range, unless `ERROR,ADD` has been entered.

The elements to be misaligned must first be selected by the `SELECT` command (see Section 7.2). Alignment errors can then be defined by the statement

```
EALIGN,DX=real,DY=real,DS=real,DPHI=real,DTHETA=real,DPSI=real,
       MREX=real,MREY=real;
```

Its attributes are:

DX The misalignment in the x-direction for the entry of the beam element (default: 0 m). A positive value displaces the element in the positive x-direction (see Figure 10.1).

DY The misalignment in the y-direction for the entry of the beam element (default: 0 m). A positive value displaces the element in the positive y-direction (see Figure 10.2).

DS The misalignment in the s-direction for the entry of the beam element (default: 0 m). A positive value displaces the element in the positive s-direction (see Figure 10.1).

DPHI The rotation around the x-axis, (default: 0 rad). A positive angle (see Figure 10.1) gives a greater x-coordinate for the exit than for the entry.

DTHETA The rotation around the y-axis (default: 0 rad) according to the right hand rule (see Figure 10.1).

DPSI The rotation around the s-axis (default: 0 rad) according to the right hand rule (see Figure 10.3).

MREX The horizontal read error for a beam position monitor (default: 0 m). This is ignored if the element is not a monitor For a positive value (see Figure 10.4) the reading for x is too high.

MREY The vertical read error for a beam position monitor (default: 0 m). This is ignored if the element is not a monitor. For a positive value (see Figure 10.4) the reading for y is too high.

Example:

```
EALIGN,QF[2],DX=0.002,DY=0.0004*RANF(),DPHI=0.0002*GAUSS();
```

See also:

- Deferred Expressions (see Section 2.8.5),
- Random Generators (see Section 2.8.5).

10.4 Define Field Errors

Field errors can be entered as relative or absolute errors. Different multipole components can be specified with different kinds of errors (relative or absolute). If an attempt is made to assign both a relative and an absolute error to the same multipole component, the absolute error is used and a warning is given. Relations between absolute and relative field errors are listed below.

All field errors are specified as the integrated value $\int K_k ds$ of the field components along the magnet axis in m^{-k}. At present field errors may only affect field components allowed as normal components in a magnet. This means for example that a dipole may have errors of the type dipole, quadrupole, sextupole, and octupole; but not of the type decapole. There is no provision to specify a global relative excitation error affecting all field components in a combined function magnet. Such an error may only be entered by defining the same relative error for all field components.

First the elements to be affected must be selected by the `SELECT` command (see Section 7.2). Field errors can then be specified for magnetic elements by one of the statements

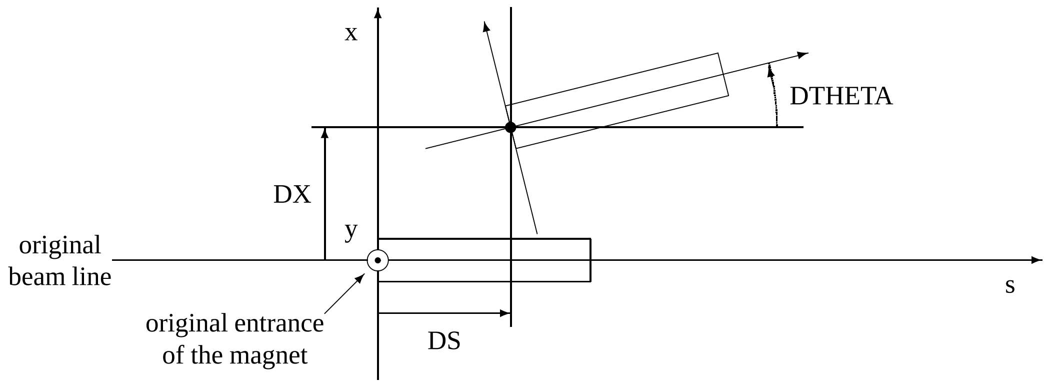

Figure 10.1: Example of Misplacement in the (x, s)-plane.

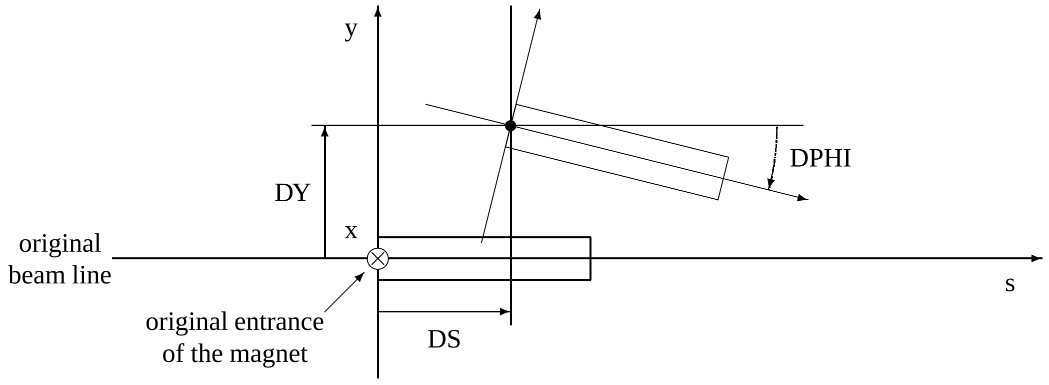

Figure 10.2: Example of Misplacement in the (y, s)-plane.

```
EFIELD,ORDER=integer,RADIUS=real,ROT=deferred-vector,
       DKR=deferred-vector,DK=deferred-vector;
EFCOMP,ORDER=integer,RADIUS=real,
       DKN=deferred-vector,DKS=deferred-vector,
       DKNR=deferred-vector,DKSR=deferred-vector;
```

Each new `EFIELD` or `EFCOMP` statement replaces the field errors for all selected elements. Any old field errors present in the range are discarded or incremented depending on the setting of `ERROR,ADD`. `EFIELD` defines the error in terms of relative or absolute amplitude and rotation; while `EFCOMP` defines them in terms of relative or absolute components.

Both commands have the attributes:

ORDER If relative errors are entered for multipoles, this defines the order of the base component to which the relative errors refer. The default is zero.

RADIUS Radius R were the relative components are specified (default 1 m). This attribute is required if any relative component is used.

The command `EFIELD` has the following additional attributes:

DK A `vector` (see Section 3.4.3) of deferred expressions (see Section 2.8.5). Its component k is the absolute error amplitude for the multipole strength with $(2k + 2)$-poles (default: $0\mathrm{m}^k$).

DKR A `vector` (see Section 3.4.3) of deferred expressions (see Section 2.8.5). Its component k is the relative error amplitude for the multipole strength with $(2k + 2)$-poles (default: $0\mathrm{m}^k$).

ROT A `vector` (see Section 3.4.3) of deferred expressions (see Section 2.8.5). Its component k is the rotation angle (see Figure 10.3) for the multipole strength with $(2k + 2)$-poles (default: 0 rad).

The command `EFCOMP` has the following additional attributes:

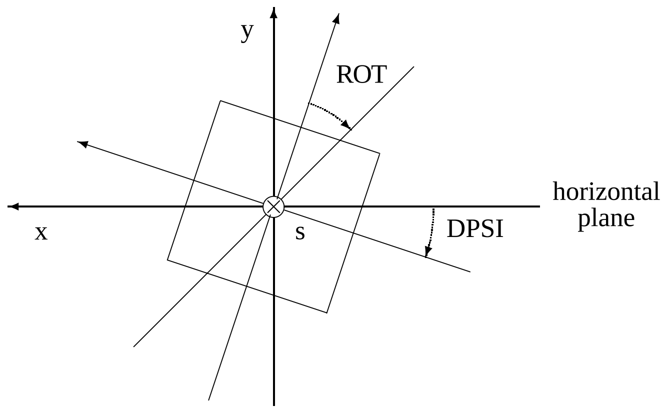

Figure 10.3: Example of Misplacement in the (x, y)-plane.

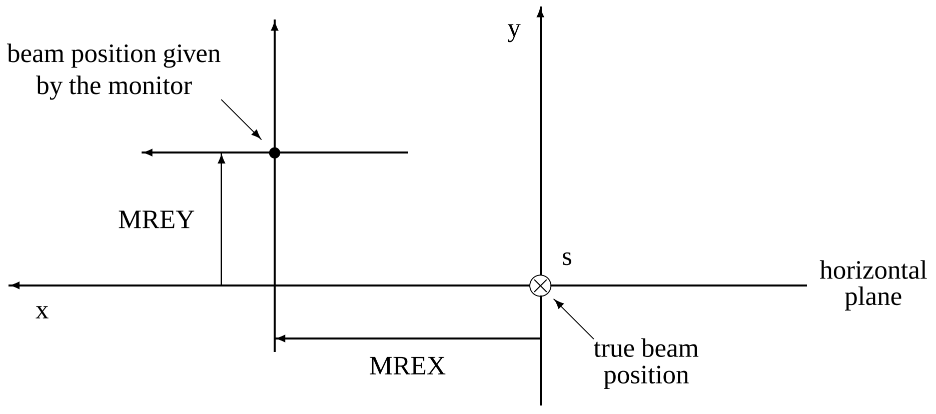

Figure 10.4: Example of Read Errors in a monitor.

DKN A `vector` (see Section 3.4.3) of deferred expressions (see Section 2.8.5). Its component k is the absolute error for the normal multipole strength with $(2k + 2)$-poles (default: $0\mathrm{m}^k$).

DKS A `vector` (see Section 3.4.3) of deferred expressions (see Section 2.8.5). Its component k is the absolute error for the skew multipole strength with $(2k + 2)$-poles (default: $0\mathrm{m}^k$).

DKNR A `vector` (see Section 3.4.3) of deferred expressions (see Section 2.8.5). Its component k is the relative error for the normal multipole strength with $(2k + 2)$-poles (default: $0\mathrm{m}^k$).

DKSR A `vector` (see Section 3.4.3) of deferred expressions (see Section 2.8.5). Its component k is the relative error for the skew multipole strength with $(2k + 2)$-poles (default: $0\mathrm{m}^k$).

All random vectors have an unlimited length.
Examples:

```
EFIELD,DK={0,0.0025*RANF(),0,0,0,DK(5)=0.0092*GAUSS()};
EFIELD,DKR={0,0,0,0.0025*RANF(),0,DKR(5)=0.0092*GAUSS()};
```

See also:

- Multipole Field Errors (see Section 10.4),
- Sign Conventions for magnetic fields (see Section 1.5),
- Attribute Vectors (see Section 3.4.3)
- Deferred Expressions (see Section 2.8.5),
- Random Generators (see Section 2.8.5).

10.4.1 Multipole Field Errors

Multipole field errors can be assigned to all multipole-like elements:

- `RBEND` (see Section 4.7),
- `SBEND` (see Section 4.7),
- `QUADRUPOLE` (see Section 4.8),
- `SEXTUPOLE` (see Section 4.9),
- `OCTUPOLE` (see Section 4.10),
- `MULTIPOLE` (see Section 4.11),

The main field and the error field are evaluated separately and then added. Error field components can be absolute or relative values. Relative errors are referred to the amplitude of the nominal component of order `ORDER` = n (default = 0), taken at the radius `RADIUS` = R (default = 1).

Given the factors

$$\psi_k = (k+1) * \texttt{ROT[k]}, \qquad c_k = \cos\psi_k, \qquad s_k = -\sin\psi_k, \qquad f_k = \texttt{RADIUS[k]}^(k-n) * n!/k!.$$

the field components δK_{kn} (normal) and δK_{ks} (skew) are computed as follows from the input data:

$$\begin{array}{ccccccccc} \delta K_{kn} & = & c_k\texttt{DK[k]} & = & f_k c_k \texttt{DKR[k]} & = & \texttt{DKN[k]} & = & f_k\texttt{DKNR[k]} \\ \delta K_{ks} & = & s_k\texttt{DK[k]} & = & f_k s_k \texttt{DKR[k]} & = & \texttt{DKS[k]} & = & f_k\texttt{DKSR[k]} \end{array}$$

The first line gives the normal components, the second line the skew components.

10.5 Print Machine Imperfections

Before imperfections can be listed, the elements to be printed must be selected by a `SELECT` command (see Section 7.2). Then a table of errors assigned to these elements can be printed by the command

```
EPRINT,FILE=string,ALIGN=logical,FIELD=logical,ORDER=integer,RADIUS=real;
```

The command has the following attributes:

FILE The file to receive the output.

ALIGN If true, the alignment errors are printed.

FIELD If true, the field errors are printed.

ORDER The order used for normalizing, like on `EFIELD` and `EFCOMP`.

RADIUS The radius used for normalizing, like on `EFIELD` and `EFCOMP`.

10.6 Save Machine Imperfections

Before imperfections can be saved, the elements to be saved must be selected by a `SELECT` command (see Section 7.2). The element imperfections can then be saved to a file with the command

```
ESAVE,FILE=string,ALIGN=logical,FIELD=logical,ORDER=integer,RADIUS=real;
```

This command saves a table of errors assigned to elements on a file, using a format which can be read again by MAD to obtain the same results. This allows dumping the errors and reloading them for the same line. Field errors are saved as relative errors, and the `RADIUS` and `ORDER` must be entered as used on the `EFIELD` (see Section 10.4) or `EFCOMP` (see Section 10.4) commands. If `OPTION,MAD8` (see Section 3.3) is active, the output is given in MAD-8 format. The command has the following attributes:

FILE The file to receive the output.

ALIGN If true, the alignment errors are saved.

FIELD If true, the field errors are saved.

ORDER The order used for normalizing, like on `EFIELD` and `EFCOMP`.

RADIUS The radius used for normalizing, like on `EFIELD` and `EFCOMP`.

Chapter 11

Tracking

Command	Purpose
`TRACK` (11.1)	Enter tracking mode
`name=expression` (3.4.1)	Parameter relation
`START` (11.2)	Define initial conditions
`RUN` (11.3)	Run particles for specified number of turns
`TSAVE` (11.4)	Save end conditions
`ENDTRACK` (11.1)	Leave tracking mode

Table 11.1: Commands accepted in Tracking Mode

11.1 Track Mode

Before starting to track, a working beam line (see Section 5.1) or sequence (see Section 5.2) and a beam (see Section 6.2) must be selected by the command

```
TRACK,LINE=name,BEAM=name;
```

This command causes MAD to enter "tracking mode", in which it accepts only the track commands (see Table 11.1). The attributes of the command are:

LINE The label of a preceding `LINE` (see Section 5.1) or `SEQUENCE` (see Section 5.2) (no default).

BEAM The named `BEAM` command defines the particle mass, charge and reference momentum (default: `UNNAMED_BEAM`).

Particles are tracked in parallel i.e. the coordinates of all particles are transformed at each beam element as it is reached.
MAD leaves **track mode** when it sees the command

```
ENDTRACK;
```

11.2 Define Initial Conditions

The `START` command defines the initial coordinates of the particles to be tracked. There may be many `START` statements, one for each particle. Particles are always started with coordinates relative to the computed closed orbit for the defined energy error. The command format is:

```
START,X=real,PX=real,Y=real,PY=real,T=real,DELTAP=real;
START,FX=real,PHIX=real,FY=real,PHIY=real,FT=real,PHIT=real;
```

`START` statements must be entered after the `TRACK` command (see Section 11.1), but before the `RUN` command (see Section 11.3).

11.2.1 Absolute Particle Positions

The first form of the `START` command (see Section 11.2) defines absolute particle positions (see Section 1.6):

X Horizontal position x, referred to the ideal orbit [m].

PX Horizontal canonical momentum, divided by the reference momentum [1].

Y Vertical position y, referred to the ideal orbit [m].

PY Vertical canonical momentum, divided by the reference momentum [1].

T The negative time difference, multiplied by the instantaneous velocity of the particle [m].

PT Momentum error, divided by the reference momentum [1].

11.2.2 Normalised Particle Positions

The second form of the `START` command defines normalised particle positions (see Section 1.6.2):

FX The normalised amplitude for mode 1 [1]:

PHIX The phase for mode 1 [1]: $\phi_x = -\arctan(p_{xn}/x_n)/2\pi$,

FY The normalised amplitude for mode 2 [1]:

PHIY The phase for mode 2 [1]: $\phi_y = -\arctan(p_{yn}/y_n)/2\pi$,

FT The normalised amplitude for mode 3 [1]:

PHIT The phase for mode 3 [1]: $\phi_t = +\arctan(p_{tn}/t_n)/2\pi$,

11.2.3 Initial Conditions

Mixing absolute and normalised positions is possible, in this case the results are added. Initial conditions Z in unnormalised phase space are related to the closed orbit and the absolute and normalised coordinates as follows:

$$\begin{array}{rcl} Z = Z_{co} & + & \sqrt{E_x}\mathtt{FX}(\Re V_k \cos\mathtt{PHIX} + \Im V_k \sin\mathtt{PHIX}) \\ & + & \sqrt{E_y}\mathtt{FY}(\Re V_k \cos\mathtt{PHIY} + \Im V_k \sin\mathtt{PHIY}) \\ & + & \sqrt{E_t}\mathtt{FT}(\Re V_k \cos\mathtt{PHIT} + \Im V_k \sin\mathtt{PHIT}) \end{array}$$

where Z_{co} is the closed orbit vector, and Z is the vector

$$Z = (\mathtt{X},\mathtt{PX},\mathtt{Y},\mathtt{PY},\mathtt{T},\mathtt{PT})^T,$$

and $\Re V_k$ and $\Im V_k$ are the real and imaginary parts of the k^{th} eigenvector, which are computed in the `TRACK` command (see Section 11.1).

11.3 Track Particles for a Specified Number of Turns

This command starts or continues the actual tracking:

```
RUN,METHOD=name,FILE=string,TURNS=integer;
```

The `RUN` command initialises tracking and uses the most recent particle bunch for initial conditions. The particle positions may be the result of previous tracking, or they may have been generated by `START` commands (see Section 11.2). The command writes a file of particle positions. Its attributes are:

METHOD The name of the tracking method to be used. For the time being only one method is known:

> **THIN** All elements are treated a s thin lenses. This is the fastest of the known method which do not lump elements.

FILE The name of the file to be writen (default=”`track`”).

TURNS The number of turns (integer) to be tracked (default: 1).

Example:

```
RUN,TURNS=1000,FILE="table";
```

This command tracks 1000 turns and writes the results on file `table`.

11.4 Save Particle Positions

The command

```
TSAVE,FILE=string;
```

saves the most recent particle bunch positions on the file named by `string`. These will normally be the positions of surviving particles after the most recent `RUN` command (see Section 11.3). If no `RUN` command has been seen yet, the positions are the result of any `START` commands (see Section 11.2) seen. The positions are written as `START` commands, and the file may be read by a subsequent tracking run.

11.5 Tracking Example

```
L: LINE=(OCT,-OCT);
// Misalignments and closed orbit correction may be done here.
P0=60.0;
B:BEAM,ENERGY=60.0;
TRACK,LINE=L,BEAM=B;
   START,X=0.001,Y=0.001,PT=0.001;
   RUN,TURNS=1024;
ENDTRACK;
```

Appendix A

MAD-9 Language Syntax

Words in *italic font* are syntactic entities, and characters in `monospaced font` must be entered as shown. Comments are given in **bold font**.

Statements:

comment	:	`//` *anything-except-newline*
	|	`/*` *anything-except-*`*/` `*/`
identifier	:	`[a-zA-Z][a-zA-Z0-9-]`
integer	:	`[0-9]+`
string	:	`'`*anything-except-single-quote*`'`
	|	`"`anything-except-double-quote`"`
command	:	*keyword attribute-list*
	|	*label* `:` *keyword attribute-list*
keyword	:	*identifier*
label	:	*identifier*
attribute-list	:	*empty*
	|	*attribute-list* `,` *attribute*
attribute	:	*attribute-name* // **only for logical attribute**
	|	*attribute-name* = *attribute-value* // **expression evaluated**
	|	*attribute-name* `:=` *attribute-value* // **expression retained**
attribute-name	:	*identifier*
attribute-value	:	*string-expression*
	|	*logical-expression*
	|	*real-expression*
	|	*array-expression*
	|	*constraint*
	|	*variable-reference*
	|	*place*
	|	*range*
	|	*token-list*
	|	*token-list-array*
	|	*regular-expression*

Real expressions:

```
real-expression      :  real-term
                     |  + real-term
                     |  - real-term
                     |  real-expression + real-term
                     |  real-expression - real-term
real-term            :  real-factor
                     |  real-term * real-factor
                     |  real-term / real-factor
real-factor          :  real-primary
                     |  real-factor ^ real-primary
real-primary         :  real-literal
                     |  symbolic-constant
                     |  #
                     |  real-name
                     |  array [ index ]
                     |  object-name -> real-attribute
                     |  object-name -> array-attribute [ index ]
                     |  table-reference
                     |  real-function ( )
                     |  real-function ( real-expression )
                     |  real-function ( real-expression , real-expression )
                     |  function-of-array ( array-expression )
                     |  ( real-expression )
real-function        :  RANF
                     |  GAUSS
                     |  USER0
                     |  SI
                     |  SC
                     |  SO
                     |  ABS
                     |  TRUNC
                     |  ROUND
                     |  FLOOR
                     |  CEIL
                     |  SIGN
                     |  SQRT
                     |  LOG
                     |  EXP
                     |  SIN
                     |  COS
                     |  ABS
                     |  TAN
                     |  ASIN
                     |  ACOS
                     |  ATAN
                     |  TGAUSS
                     |  USER1
                     |  ATAN2
                     |  MAX
                     |  MIN
                     |  MOD
                     |  POW
                     |  USER2
```

function-of-array	:	`VMIN`
	|	`VMAX`
	|	`VRMS`
	|	`VABSMAX`

Real variables and constants:

real-prefix	:	empty
	|	`REAL`
	|	`REAL CONST`
	|	`CONST`
real-definition	:	*real-prefix real-name* = *real-expression* // **expression evaluated**
	|	*real-prefix real-name* `:=` *real-expression* // **expression ratained**
symbolic-constant	:	`PI`
	|	`TWOPI`
	|	`DEGRAD`
	|	`RADDEG`
	|	`E`
	|	`EMASS`
	|	`PMASS`
	|	`CLIGHT`
	|	*real-name*
real-name	:	*identifier*
object-name	:	*identifier*
table-reference	:	*table-name* @ *place* `->` *column-name*
table-name	:	*identifier*
column-name	:	*identifier*

Logical expressions:

logical-expression	:	*and-expression*
	|	*logical-expression* `||` *and-expression*
and-expression	:	*relation*
	|	*and-expression* && *relation*
relation	:	*logical-name*
	|	`TRUE`
	|	`FALSE`
	|	*real-expression relation-operator real-expression*
logical-name	:	*identifier*
relation-operator	:	`==`
	|	`!=`
	|	`<`
	|	`>`
	|	`>=`
	|	`<=`

Logical variables:

logical-prefix	:	`BOOL`
	|	`BOOL CONST`
logical-definition	:	*logical-prefix logical-name* = *logical-expression* // **expression evaluated**
	|	*logical-prefix logical-name* `:=` *logical-expression* // **expression retained**

String expressions:

string-expression	:	*string*
	|	*identifier* // **taken as a string**
	|	*string-expression* & *string*

String constants:

string-prefix	:	`STRING`
string-definition	:	*string-prefix string-name* = *string-expression* // **expression evaluated**
	|	*string-prefix string-name* `:=` *string-expression* // **expression retained**

Real array expressions:

array-expression	:	*array-term*
	|	`+` *array-term*
	|	`-` *array-term*
	|	*array-expression* `+` *array-term*
	|	*array-expression* `-` *array-term*
array-term	:	*array-factor*
	|	*array-term* `*` *array-factor*
	|	*array-term* `/` *array-factor*
array-factor	:	*array-primary*
	|	*array-factor* ^ *array-primary*
array-primary	:	{ *array-literal* }
	|	*array-reference*
	|	*table-generator*
	|	*row-reference*
	|	*column-reference*
	|	*real-function* `(` *array-expression* `)`
	|	`(` *array-expression* `)`
table-generator	|	`TABLE` `(` *last* `,` *real-expression* `)`
	:	`TABLE` `(` *first* `:` *last* `,` *real-expression* `)`
	:	`TABLE` `(` *first* `:` *last* `:` *step* `,` *real-expression* `)`
first	:	*integer*
last	:	*integer*
step	:	*integer*
table-row	:	*table-name* `@` *place*
row-reference	:	`ROW` `(` *table-name* `,` *place* `)` |
	|	`ROW` `(` *table-name* `,` *place* `,` { *column-list* } `)`
column-reference	:	`COLUMN` `(` *table-name* `,` *column-name* `)`
	|	`COLUMN` `(` *table-name* `,` *column-name* `,` *range* `)`
column-list	:	*column-name*
	|	*column-list* `,` *column-name*
array-literal	:	*real-expression*
	|	*array-literal* `,` *real expression*
array-reference	:	*array-name*
	|	*object-name* `->` *array-attribute*
array-name	:	*identifier*

Real array definitions:

array-prefix	:	`REAL VECTOR`
array-definition	:	*array-prefix array-name* = *array-expression*
	|	*array-prefix array-name* `:=` *array-expression*

Constraints:

constraint	:	*array-expression constraint-operator array-expression*
constraint-operator	:	`==`
	|	`<`
	|	`>`

Variable references:

variable-reference	:	*real-name*
	|	*object-name* `->` *attribute-name*

Places:

place	:	*element-name*
	|	*element-name* `[` *integer* `]`
	|	`#S`
	|	`#E`
	|	#integer
	|	*line-name* `::` *place*

Ranges:

range	:	*place*
	|	*place* `/` *place*

Token lists:

token-list	:	*anything-except-comma*
token-list-array	:	*token-list*
	|	*token-list-array* `,` *token-list*

Regular expressions:

regular-expression	:	`"`*UNIX-regular-expression*`"`

Bibliography

[1] Karl L. Brown. *A First-and Second-Order Matrix Theory for the Design of Beam Transport Systems and Charged Particle Spectrometers.* SLAC 75, Revision 3, SLAC, 1972.

[2] Karl 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] E. D. Courant and H. S. Snyder. *Theory of the alternating gradient synchrotron.* Annals of Physics, 3:1–48, 1958.

[4] Ph. Defert, Ph. Hofmann, and R. Keyser. *The Table File System, the C Interfaces.* LAW Note 9, CERN, 1989.

[5] 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.

[6] D. A. Edwards and L. C. Teng. *Parametrisation of linear coupled motion in periodic systems.* IEEE Trans. on Nucl. Sc., 20:885, 1973.

[7] LEP Design Group. *Design Study of a 22 to 130 GeV e^+e^- Colliding Beam Machine (LEP).* CERN/ISR-LEP/79-33, CERN, 1979.

[8] F. James. *MINUIT, A package of programs to minimise a function of n variables, compute the covariance matrix, and find the true errors.* program library code D507, CERN, 1978.

[9] D. E. Knuth. *The Art of Computer Programming.* Volume 2, Addison-Wesley, second edition, 1981. Semi-numerical Algorithms.

[10] H. Mais and G. Ripken, *Theory of Coupled Synchro-Betatron Oscillations.* DESY internal Report, DESY M-82-05, 1982.

[11] Gerhard Ripken, *Untersuchungen zur Strahlführung und Stabilität der Teilchenbewegung in Beschleunigern und Storage-Ringen unter strenger Berücksichtigung einer Kopplung der Betatronschwingungen.* DESY internal Report R1-70/4, 1970.

[12] L. C. Teng. *Concerning n-Dimensional Coupled Motion.* FN 229, FNAL, 1971.

Index