David Sagan and Peter Tenenbaum
Revision: 0.76
February 14, 2013

AML: Accelerator Markup Language

Draft

Introduction

New particle accelerator facilities are increasing in scale and complexity. This increase brings an ever greater need for global collaboration, and for accurate modeling of proposed designs to ensure technical performance and to manage costs. Various labs around the world have developed excellent accelerator modeling codes that could be used for such analysis. Using a variety of codes to analyze a single machine brings benefits because different codes tend to be optimized for different purposes. Additionally, cross-checking results between different codes is often essential for validating the results. However, different codes generally require input files in different formats, and this is a significant obstacle to collaboration. Furthermore, the *de facto* standard for accelerator descriptions, the Standard Input Format (SIF), is quite outdated, lacking in many features which are essential for describing a modern, highly complex accelerator facility.

To address this problem, a lattice description format called Accelerator Markup Language (*AML*) has been created. *AML* is based upon the standard eXtensible Markup Language (XML) which provides the necessary flexibility for *AML* to be easily extended to changing requirements, and integrates the language for accelerator description into the broader language and parsing community. The design of *AML* has incorporated the lessons learned and requirements determined by a quarter century of experience with complex accelerator facilities since the development of the Standard Input Format (SIF) which is used by numerous programs including the MAD program. Moreover, the extensibility of XML enables *AML* files to be used beyond lattice descriptions to include such information as the control system configuration, blueprint and other documentation, magnet history information, etc. In short, *AML* could be used as the basis for a complete database of an accelerator laboratory complex.

In conjunction with *AML*, a set of routines is being developed to simplify and speed the integration of *AML* into any program. These routines, collectively known as the Universal Accelerator Parser (UAP), will also provide the means to specify additional lattice file formats. This will allow programs that use the UAP code to read a variety of different file formats. Currently the UAP supports, besides *AML*, the MAD lattice format. The UAP will additionally be able to convert lattice files between any formats for which appropriate specifications are given.

The UAP code is being released under the open source GNU Lesser General Public License[1]. The *AML*/UAP project is a joint effort among various laboratories. The UAP source code along with the LaTeX input files for generating this manual may be obtained from SourceForge.com:

`http://sourceforge.net/projects/accelerator-ml`

The project web page is at:

`http://www.lepp.cornell.edu/~dcs/aml`

Contents

List of Figures

List of Tables

Part I

AML Syntax

Chapter 1

AML Syntax

This manual documents the `Accelerator Markup Language` (*AML*) lattice description format. *AML* is based upon the eXtensible Markup Language (XML). XML is a widely used, general-purpose text format standard developed by the World Wide Web Consortium (W3C) that has been used to support a wide variety of applications including describing web pages (XHTML) mathematical expressions (MathML), music (MML), etc. That is, XML defines, in a general way, the format for representing data in a file or within a program, and *AML* defines exactly what kind of data is actually present.

There are several reasons why XML was chosen as a basis for *AML*. For one, since XML is a widely accepted standard, it is well documented and there exists widely available XML parsers and text processors. Another good reason for choosing XML is that is allows AML lattice files to be extended, in a fairly transparent manner, to include custom information such as blueprint and other documentation, magnet history information, etc. In other words, AML can be used to incorporate lattice information in a general laboratory wide database. This is made all the more seamless by the fact that XML is widely used as a database format.

This chapter gives an overview of XML. Since there are many good books on XML, not to mention the resources on the web, this chapter is only ment as a summery and the reader is encourged to look elsewhere for more detailed information.

1.1 Nodes, Attributes, and Children

XML represents data in a hierarchical "tree" structure. An example will make this clear:

```
<laboratory>
  <doc> Hello World! </doc>
  <machine name = "LINAC">
    <element ref = "q01">
    </element>
  </machine>
</laboratory>
```

An XML `element` The text starting with a ‘‘<’’ character and ending with a ‘‘>’’ character is called an XML `tag`. Tags have `names` and tags are paired so that every opening start-tag of a given name, for example `<laboratory>`, must have a closing end-tag of the same name. End-tags are distinguished by having a “/” character right after the opening “<” character. In a document, The “top-level” node

at the base of the tree is called the `root` node. If this example represented an entire document, the `<laboratory>` node would be the root.

Element names must obey XML naming conventions:

- Names must start with either a letter or the "-" character.
- After the first character, numbers, hypens, and periods are allowed.
- Names cannot contain spaces.
- Names cannot start with the letters `xml` in uppercase, lowercase, or mixed case.
- There cannot be any white space between the opening `<` character and the name.

All the information from the start of a start-tag to the end of an end-tag, including everything in between, is called an `element`. Thus, in the above example, the `<laboratory>` element the five lines in between the start `<laboratory>` and end `</laboratory>` tags. Since accelerator builders typically use the word `element` to refer to an actual physical element of the beamline, in this document, the XML elements will be referred to as *nodes*.

The name of a node may contain a `prefix` that referrs to a `namespace` (§1.3). The `prefix` is separated from the rest of the name by a colon `":"`. For example, `md:mind` has the prefix `md` and the rest of the name (called the `local part`) is the string `mind`.

The data between a node's opening and closing tags (excluding the tags them selves) is called the node's `content`. The content of a node will consist of a number of `children` which may be either other nodes, text, or a combination of text and nodes. Thus, in the above example, the `<laboratory>` node has two children, a `doc` node and a `machine` node, and the `<doc>` node has the character text child `Hello World!`.

If a node has no content, then for abrevity, the end-tag may be combined with the start-tag into one `''self-closing''` tag with the characters "/>" at the end of the tag. Thus, in the above example, the `<element>` node may be written

```
<element ref = "q01" />
```

Besides a node's content, a node may also include `attributes` which are simple name/value pairs that appear after the node's name within the start-tag. In the above example, the `<element>` node has a single attribute named `ref` which has a value `"q01"`. Attributes must always have values even if that value is just an empty string (`""`) and attribute values must always be in quotes. Either single (') or double (") quotes are acceptable. Attribute names must be unique so the following is not permitted

```
<element ref = "q01" ref = "q02" />
```

Note that the word `attribute` is used in XML-speak as shown above, and in accelerator-speak as an attribute of a machine element such as the element's length or magnetic field strength. Which is which is generally obvious from context. However, to reduce confusion, the terms `XML attribute` and `machine element attribute` are sometimes used.

1.2 XML Syntax Rules

A `well formed` XML document is a document that conforms to the XML syntax rules. These are

- Every start-tag must have a matching end-tag or be a slef-closing tag.

- Tags cannot overlap and nodes must be properly nested. Thus the following is not allowed:
  `<a <b> >`
  or
  `<a> <b> </a> </b>`
- XML documents must have one and only one root node.
- Node names must obey XML naming conventions:
- XML is case sensitive. Thus `<node>` and `<Node>` are different names.

1.3 Namespaces and Adding Custom Information to AML Files

AML defines a set of node names like `<laboratory>`, `element`, and `quadrupole` that can be used to describe a lattice. To combine this in a single file with information that is *not* defined by *AML*, and to simultaneously keep things logically separated, namespaces need to be used.

An example will be used to explain the concept of namespaces:

```
<root xmlns:md  = "medical_info"
      xmlns:bio = "biographic_info"
      xmlns     = "my_info">
  <bio:person name = "Einstein">
    <md:brain size = "average" />
    <info tag = "Added 2008/10/11" />
  </bio:person>
</root>
```

The `xmlns` prefix is defined in XML to be used to declare namespace bindings. In this example, three namespaces, called `medical_info`, `biographic_info`, `my_info`, are declared. The first two namespaces are associated with a prefix: The `medical_info` namespace is assocaited with the `md` prefix and the `biographic_info` namespace is associated with the `bio` prefix. The third namespace is called the `default` namespace since it does not have a prefix associated with it.

Using the prefixes, nodes in the XML tree can be associated with the appropiate namespace. In the above example, the `<bio:person>` node is associated with the `biographi_info` namespace and the `<info>` node, since it does not have a prefix, is associated with the `my_info` namespace.

All prefixes must have an associated name space. For example, the use of `<bio:person>` is only allowed if the `bio` prefix has been associated with a namespace via an `xmlns` construct. However, a default namespace does not have to be defined.

Namespaces allow the integration of different applications. For example, Including MathML constructs within an HTMLX file. To keep things separate, the name of a namespace should be some unique string. Conventionally, this unique string looks like a web address that is associated with the people or organization creating the application. For example, MathML's namelist name is

`http://www.w3.org/1998/Math/MathML`

This fairly well ensures that no other application developed anywhere in the world will use the same namespace name. However, using a web address as the namelist name is not mandated by XML and even if the namelist name looks like a web address, there is no mandate that this address be connected to an actual web page.

The `scope` of a namespace declaration (using the `xmlns` construct) extends from the beginning of the start-tag in which it appears to the end of the corresponding end-tag. A namespace declaration overrides any encompassing declaration. For example:

```
<laboratory xmlns = "http://lepp.cornell.edu/aml">
  <controller name = "PS236" variation = "ABSOLUTE" default_attribute = "bend:g">
    <slave target = "BH1R10A" expression = "0.174533 * PS236[@actual]" />
    <control_sys xmlns = "http://atf.kek.jp/atf">
      <pvname>
        <read value = "SBEN:BH1R:IACT.VAL" />
        <write value = "SBEN:BH1R:IDES.VAL" />
      </pvname>
      <protocol value = "EPICS" />
    </control_sys>
  </controller>
</laboratory>
```

In this example, the `<laboratory>`, `<controller>`, and `<slave>` nodes are associated with the namelist named `"http://lepp.cornell.edu/aml"` which is definded in the `<laboratory>` node. The `<control_sys>` node defines a new default namelist called `"http://atf.kek.jp/atf"` so the `<control_sys>` node and all of its sub-nodes are associated with the vn"http://atf.kek.jp/atf" namelist.

XML allows attributes to have prefixes so that a single node can contain attributes from different namespaces. None the less, no attributes from a different namespace are allowed in *AML* namespace nodes. However, this restriction does not apply to non-*AML* namespace nodes.

1.4 Readability

There is no limit to the line length permitted by XML and *AML*, though common sense and readability constraints suggest that a line which is not much longer than the canonical 80 characters would be best. No continuation characters are required, since XML requires an explicit end tag for any node.

XML does not mandate how tags should be layed out. However for readablility, end-tags should either be on the same line as it's corresponding start-tag or, if on different lines, have the opening `<` character be aligned vertically. It is suggested that children start and end tags be indented two spaces to the right of their parent node's tags.

1.4.1 AML Names

AML, following XML, is case-sensitive. This includes all node names, attribute key words, etc. Thus machine element names which differ only by their case (such as `Q1` and `q1`) are considered different names by AML.

Many nodes in *AML* can take `name` as an attribute. All of the names in an *AML* lattice must be unique. As mentioned above, names which vary only in their use of upper and lower case constitute different names. Names may not include any whitespace.

Chapter 2

AML Top Level and Key Concepts

2.1 <laboratory> — The Root AML Node

Following XML, *AML* data is organized and represented in a hierarchical fashion. With *AML*, the root (top level) node in the hierarchy is the `<laboratory>` node. The tree from this root can describe not only the actual accelerator machine or machines but the entire laboratory complex. Example:

```
<laboratory name = "Wilson Lab">
  . . .
</laboratory>
```

The attributes of `<laboratory>` can be:

```
name
version
```

The `version` attribute allows an *AML* parser the ability to detect a problem if the *AML* syntax is changed. The current version is `"1"`.

The children of `<laboratory>` can be:

`<beam>`	§4.1	Initial beam parameters.
`<comment>`	§2.5	Simple documentation.
`<constant>`	§2.6.3	Numerical parameters.
`<controller>`	§6.1	Controllers.
`<doc>`	§2.5	Documentation.
`<element>`	§3	Machine components
`<lattice>`	§5.2.2	Lattice parameters.
`<machine>`	§5.2	Machine used for calculations.
`<global>`	§2.3	General global parameters.
`<set>`	§2.6.2	Pre-expansion attribute set.
`<post_set>`	§2.6.2	Post-expansion attribute set.
`<sector>`	§5.1	List of Machine components.
`<xi:include>`	§2.2	File inclusion.

2.2 Use of Multiple Files

The information which define an accelerator or accelerator complex can be spread out over multiple files. This is accomplished with the `<include>` node:

```
<laboratory xmlns:xi="http://www.w3.org/2001/XInclude">
  ...
  <xi:include href = "commondefs.aml" />
  ...
</laboratory>
```

the `xmlns` prefix (§1.3) in the example above associates the prefix `xi` with the namespace

```
http://www.w3.org/2001/XInclude
```

This namespace is defined by the World Wide Web Consortium for use in including XML files within other XML files. The only attribute of the `<include>` node is `href`, which is the file name to be included.

File paths in the `<include>` node may be either relative or absolute. A file specification which begins with a period (".") or an alphanumeric character is taken to be a relative path; any other first character (including forward or backslash) is taken to be an absolute path. When an `<include>` node is executed, the default path is updated to be the path used in the `<include>` node. For example:

```
<xi:include href = "/master/commondefs.aml" />
<xi:include href = "slave/specdefs.aml" />
```

When *AML* goes to load the second file, it will look for the file `/master/slave/specdefs.aml`.

2.2.1 Nesting of <laboratory> Nodes

In addition to the child nodes listed above, a `<laboratory>` node may take as a child node another `<laboratory>` node. This nesting permits use of an existing lattice as a subset of a greater whole without modifying the subsidiary lattice file itself. For example, if the file `<this_lattice.aml>` contains a `<laboratory>` node, then a wrapper file may be created which could look like:

```
<laboratory xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href = "this_lattice.aml" />
  <set attribute = "m[multipole:k(n=1)]" value = "2.7" />
  ... etc ...
</laboratory>
```

In other words, *AML* will not choke when it sees a `<laboratory>` which is a child of another `<laboratory>`.

2.3 <global>

`<global>` defines some general parameters.

Possible attributes of `<global>` are

```
name = "Name"
```

The `name` attribute allows reference to `<global>` parameters in arithmetic expressions (§2.6.1). There can be at most one `<global>` node.

Child nodes of `<global>` are:

`<ran_seed value = "expr">`	Random number seed.
`<taylor_order value = "expr">`	Taylor order for Taylor maps.

2.4 Units and Dimensions

AML uses primarily SI units for quantities, with the exception of particle energy, which is entered in eV (rather than joules). The units used by AML are shown in Table 2.1. For more information on the

Quantity	*Units*
Angles	radians
Charge	Coulombs
Current	Amps
Frequency	Hz
Kick	radians
Length	meters
Magnetic Field	Tesla
Particle Energy	eV
Phase Angles (RF)	radians
Voltage	Volts

Table 2.1: Physical units.

magnetic field and the field expansion conventions used in AML, see §8.1.

2.5 Comments and Documentation

There are 3 syntaxes for entering comments of various kinds into an *AML* file. The intrinsic XML comment can be used, which indicates the beginning of a comment with `<!--` and the end of the comment with `-->`. This comment can span multiple lines:

```
<!--  It is legal to have a comment
    fall across several lines.
-->
```

The second method is to use an *AML* `<comment>` node. A `<comment>` node takes as attributes `type` and `text`, both of which take strings as their arguments. A `comment` can be used as the child of any other node:

```
<element name = "Q1" >
  <comment type = "Update",
      text = "Measured field values per RHI 01-jan-1992" />
</element>
```

More complicated documentary information can be entered using an *AML* `<doc>` node. The `<doc>` node can take as its attributes `author`, `date`, and `href`, all of which take string arguments. The `<doc>` node also allows the user to enter arbitrary text strings:

```
<element name = "Q1" >
  <doc date = "01-jan-1992" href = "http://www.slac.stanford.edu/~rhi">
     Field values updated to measured.  See the magnet strength
     database at the URL above for more information.
  </doc>
</element>
```

2.6 Parameters, Arithmetic Expressions, and Constants

Many of the attributes in *AML* are either numeric values or else expressions which can be evaluated to produce numeric values. Typically, the values are indicated with a `design` attribute, which, as the name implies, is the design value of the parameter in question. Any node which can take a `design` attribute

can also take an `err` attribute, which specifies an error value. In this manual, this will be indicated by a shorthand, `design|err`. In cases where an error value would never be appropriate, the attribute `value` is typically used instead of `design`.

A numeric parameter of a node can be referred to in a later parameter definition. The general syntax is:

```
name[child(att1=attval1,att2=attval2):grandchild...@value_attribute].
```

Here `name` is the name of the node of interest; `child` is the child node of interest within the node of interest, if any; `att1` and `att2` are any attributes of the child node which are needed for identification; `grandchild` is a child node of the child node; and `value_attribute` selects the numeric attribute of interest in that child, which is one of:

```
design,
value,
err,
actual.
```

For nodes which have a `design` attribute, the default `value_attribute` is `design`, while for nodes which have a `value` attribute, the default `value_attribute` is `value`. The `value_attribute actual` is equal to `design+err`, and gives the actual value of the parameter of interest, as the name implies.

For example, in the case of an `element` named QMA2 with a `quadrupole` child, the strength of the `quadrupole` is given with the following syntax (see §3.3.14 for more details):

```
<element name = "QMA2" >
  <quadrupole>
    <k design ="0.55" />
  </quadrupole>
</element>
```

The quad strength parameter of QMA2 can be referred to as `QMA2[quadrupole:k@design]`. Since the default attribute is `design`, in this example `QMA2[quadrupole:k]` would also be an acceptable syntax.

2.6.1 Arithmetic Expressions

AML allows the use of arithmetic expressions in parameter definitions. The standard operators are defined:

$a + b$	Addition
$a - b$	Subtraction
$a * b$	Multiplication
$a \,/\, b$	Division
$a \wedge b$	Exponentiation

Additionally, *AML* has the following intrinsic functions:

```
sqrt(x)       Square Root
log(x)        Logarithm
exp(x)        Exponential
sin(x)        Sine
cos(x)        Cosine
tan(x)        Tangent
asin(x)       Arc sine
acos(x)       Arc cosine
atan(x)       Arc Tangent
atan2(y, x)   Arc Tangent (= atan(y / x))
```

```
abs(x)        Absolute Value
ran()         Random number between 0 and 1
ran_gauss()   Gaussian distributed random number
```

2.6.2 Setting Parameter Values

The parameters of a node can be changed after definition. For numeric parametes, this is accomplished with the <set> node. The <set> node uses the parameter reference syntax described above to identify the parameter of interest, and uses the value attribute to set it:

```
<element name = "QMA1" >
  <length design = "0.46092" />
  <taylor_map>
    <term i_out = "1" coef = "3.7" exp = "1 0 1 0 0 0" />
  </taylor_map>
</element>
<set attribute = "QMA1[length@err]" value = "0.003" />
<set attribute = "QMA1[taylor_map:term(i_out=1,exp='1 0 1 0 0 0')]" value = "3.9" />
```

As described in §3.1.1, set cannot be used to change the parameters of elements which are used as templates for other elements via the inherit attribute.

The <set> node sets the parameter values prior to expansion of the lattice. After expansion of the lattice (see §5.3), the <set> command can no longer change the parameter values of the expanded lattice. At this time a <post_set> node is required. The syntax for <post_set> is identical to that of <set>, the only difference being the aforementioned one of acting on the expanded lattice. If there are multiple nodes with the same name in the expanded lattice, <post_set> will act on all of them.

As mentioned above, <set> and <post_set> operate on numeric parameters. For string parameters, there is a similar node, <string_set>. The syntax for <string_set> is:

```
<string_set attribute = "attribute-spec", string = "strval" />
```

where attribute_spec is the attribute to be set (using the name[child:attribute@value] syntax).

2.6.3 Named Constants

A free-standing constant can be defined by use of the <constant> node:

```
<constant name="Gradient" value = "31.5" /> <!-- MV/m -->
```

Constant definitions can use other constants or element numeric parameters, and may include arithmetic expressions. Constant definitions may refer to other constants which have not yet been defined:

```
<constant name = "c2" value = "c1*QMA1[length@design]" />
<constant name = "c1" value = "5" />
```

Once defined, a constant cannot be redefined, nor can its value be changed by the <set> command. However, a constant can be defined in terms of element parameters which can be changed. In this case, the value of the constant is updated. For example, after the sequence:

```
<set attribute = "Q01W[length]"  value = "1.0" />
<constant name = "z1" value = "2*Q01W[length]" />
<set attribute = "Q01W[length]" value = "10.0" />
```

the value of constant z1 is 20. However, the sequence:

```
<constant name = "z1" value = "10" />
<constant name = "z1" value = "20" />
```

is not permitted.

2.6.4 Predefined Constants

Commonly used physical and mathematical constants shown in Table 2.2 are defined. All values are defined in SI units.

Symbol	*Value*	*Units*	*Name*
pi	π		
twopi	$2\,\pi$		
fourpi	$4\,\pi$		
sqrt_2	$\sqrt{2}$		
degrees	$\pi/180$		
Hz	$2\,\pi$		
m_electron	$0.51099906 \cdot 10^{6}$	eV	Electron mass
m_proton	$0.938271998 \cdot 10^{9}$	eV	Proton mass
c_light	$2.99792458 \cdot 10^{8}$	m/s	Speed of light
r_e	$2.8179380 \cdot 10^{-15}$	m	Electron radius
r_p	$1.5346980 \cdot 10^{-18}$	m	Proton radius
e_charge	$1.6021892 \cdot 10^{-19}$	C	Electron charge
h_planck	$6.626196 \cdot 10^{-34}$	J/Hz	Planck's constant
h_bar_planck	$1.054591 \cdot 10^{-34}$	J s	Planck / 2π

Table 2.2: *AML* physical and mathematical constants.

Chapter 3

Machine Elements

All machine elements are defined via `<element>` nodes.

3.1 Attributes of <element> Nodes

The attributes of `<element>` are

```
name = "Name"
inherit = "Name"
ref  = "Name"
prefix = "String"
repeat = "number"
```

Either `name` or `ref` must appear but not both. `ref` is a reference to an element that is defined someplace else. If `ref` is used then there must be no child nodes; the one exception to this is that `<superimpose>` children are allowed for elements which are included by reference into a superposition sector (see §5.1.1). The `ref`, `prefix`, and `repeat` attributes can only be used in the definition of beamlines, see §5.1.

3.1.1 Duplicating Elements via the inherit Attribute

It is often useful to have multiple accelerator elements with different names which share some or all parameters (for example, all quadrupole magnets with a given design must have the same aperture and length). This can be accomplished with the `inherit` attribute, which copies the parameters of one element to another:

```
<element name = "QMA1" >
  <length design = "0.46092" />
  ...
</element>
< element name = "QMA2" inherit = "QMA1" />
```

In this example, a second element, `QMA2`, is created which is identical to `QMA1` in all but name. If desired, inherited parameters can be overwritten, leading to elements which are identical in some parameters and different in others:

```
<element name = "QMA1" >
```

```
    <length design = "0.46092" />
    ...
  </element>
  <element name = "QMA2" inherit = "QMA1" >
    <length err = "0.003" />
  </element>
```

In the example above, QMA2 is a duplicate of QMA1 except for the error on its length, which is independent. Note that once an element has been used as a template for another element, its parameters can no longer be changed. In these examples, attempting to later redefine the length or other parameters of QMA1 would result in an error.

3.2 The Children of <element>

The Children nodes of <element> are divided into two classes. The first class, listed in Table 3.1 defines the kind of element – quadrupole, bend, etc. These children are referred to as key children

<beambeam>	§3.3.1	<multipole>	§3.3.11
<bend>	§3.3.2	<octupole>	§3.3.12
<crab_cavity>	§3.3.4	<patch>	§3.3.13
<custom>	§3.3.5	<quadrupole>	§3.3.14
<electric_kicker>	§3.3.6	<rf_cavity>	§3.3.16
<field_table>	§3.4.3	<sextupole>	§3.3.17
<kicker>	§3.3.7	<solenoid>	§3.3.18
<linac_cavity>	§3.3.8	<taylor_map>	§3.3.19
<marker>	§3.3.9	<wiggler>	§3.3.20
<match>	§3.3.10		

Table 3.1: Key children of <element> that define the kind of element.

Multiple key children can be used to construct complex elements. For example, a combination quadrupole and solenoid would look like:

```
  <element name = "sq1">
    <quadrupole> ... </quadrupole>
    <solenoid> ... </solenoid>
    ...
  </element>
```

There is no explicit <drift> child. Rather, an element acts like a drift when it does not have any key children from Table 3.1.

The reference trajectory for any element that has a <bend> child is circular. Otherwise, except for an element with a <custom> child, the reference trajectory is straight. This is the essential difference between a bend and a kicker: The former has a curved reference trajectory and the latter has a straight one. If an element has a <custom> child the reference trajectory is undefined.

The second class of sub-nodes, listed in Table 3.2, define various general attributes like apertures, etc.

Most of these nodes will be children of <element> but there are a few exceptions. For one, <scaled_multipole> is never a child of element but can appear as a child in some key children. Similarly, <field_table> and <orientation> can appear as a child of <element> or may appear as a child of some key children. For <orientation> this is useful for a complex element where the orientation of the

<aperture>	§3.4.1	<methods>	§3.4.7
<comment>	§2.5	<orientation>	§3.4.8
<description>	§3.4.2	<reference>	§3.3.15
<doc>	§2.5	<scaled_multipole>	§3.4.9
<floor>	§3.4.4	<state>	§3.4.10
<instrument>	§3.4.5	<superimpose>	§3.4.11
<length>	§3.4.6	<wake>	§3.4.12

Table 3.2: Table of general property sub-nodes of <element>.

key children needs to be separately specified. The <orientation> child of <element> then represents the overall orientation.

3.3 Key Children of <element> – Definitions and Syntax

The key children of <element> define the space of possible element types which can be represented in *AML*. Their definitions and syntax are described below. Note that most of the key children can represent fields in a form which is normalized to the beam energy, which is analogous to the MAD k1, k2, etc, or in a form which is not normalized to the energy – essentially this is the normalized form multiplied by the beam energy or magnetic rigidity. The non-normalized forms are usually indicated by a ''_u'' (see Equations (8.2) and (8.3)).

3.3.1 <beambeam>

The children of <beambeam> are:

```
  <orientation>                    §3.4.8
  <sig_x design|err = "expr">      Horizontal size.
  <sig_y design|err = "expr">      Vertical size.
  <sig_z design|err = "expr">      Longitudinal size.
  <rel_charge design|err = "expr"> Strong beam charge. Default: design = -1
```

A <beambeam> element simulates an interaction with an opposing ("strong") beam traveling in the opposite direction.

The charge of the strong bunch is $r_c \times n_p$ where n_p is the number of particles in the strong bunch and is set by the <n_particles_bbi> child of the <beam> node. r_c is the relative charge per particle in the strong bunch and is set by the <rel_charge> child. (<rel_charge> can also be thought of as a way of effectively varying the number of particles in the strong bunch). If <rel_charge> is negative then the sign of the strong bunch's charge is the opposite of the sign of the beam in the beamline. The default design value for <rel_charge> is -1.

Children <sig_x>, <sig_y>, <sig_z> are the RMS dimensions of the strong bunch. Even if <sig_z> has a non-zero value, the beambeam element is always considered to have zero length in terms of the positioning of lattice elements after a <beambeam> element. That is, the <beambeam> element marks the point at the center of the "luminous region" where the opposing beams meet.

The <orientation> child gives the orientation of the strong bunch. Attributes orientation:x_offset and orientation:y_offset are used to offset the beambeam element. Attributes orientation:x_pitch and orientation:y_pitch give the beam-beam interaction a crossing angle. This is the full crossing angle, not the half-angle.

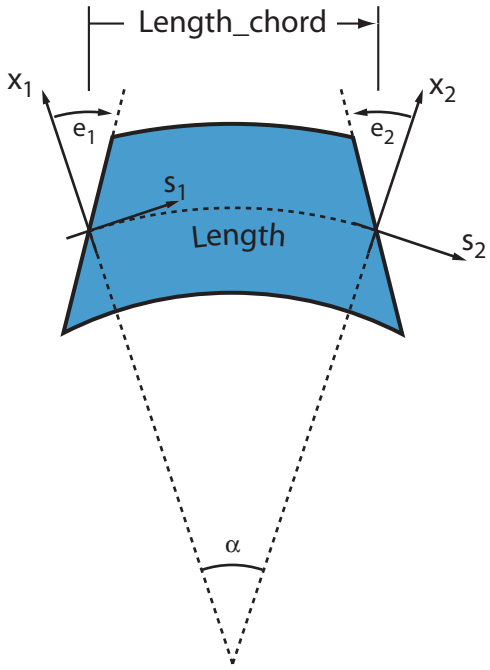

Figure 3.1: Coordinate systems for <bend> element.

The strong bunch is divided up into method:num_steps slices.

3.3.2 <bend>

<bend> is a dipole bend. The reference trajectory for any element that has a <bend> child is circular. An *AML* <bend> is essentially the same as a MAD sbend. The children of <bend> are:

<orientation>	§3.4.8
<scaled_multipole>	§3.4.9
<g design|err = "expr">	Normalized bending field = 1 / Bending_radius
<g_u design|err = "expr">	Unnormalized bending field
<e1 design|err = "expr">	Entrance end pole face rotation
<e2 design|err = "expr">	Exit end pole face rotation
<h1 design|err = "expr">	Entrance end pole face curvature
<h2 design|err = "expr">	Exit end pole face curvature
<f_int1 design|err = "expr">	Entrance end field integral
<f_int2 design|err = "expr">	Exit end field integral
<h_gap1 design|err = "expr">	Entrance end half height
<h_gap2 design|err = "expr">	Exit end half height

g, g_u
: g = 1/rho is the curvature function and is proportional to the design dipole magnetic field. The true field strength is given by design + err so changing err leaves the design orbit unchanged but does vary a particle's orbit. g_u is the unnormalized field strength (§8.1).

orientation:tilt
: The roll angle about the longitudinal axis at the entrance face of the bend is given by tilt.

`tilt` = 0 bends the reference trajectory in the $-x$ direction. If the `tilt` attribute is given without any value then the value $\pi/2$ will be used. This makes for a downward vertical ($-y$) bend.

<h1>, <h2>
`<h1>` and `<h2>` are the curvature of the entrance and exit pole faces respectively.

<e1>, <e2>
`<e1>` and `<e2>` are the rotation angles of the entrance pole and exit pole faces. A `bend` with `<e1>` = `<e2>` = `g` * `length` / 2 is equivalent to a rectangular shaped magnet (called an `rbend` in MAD)

<f_int1>, <f_int2>
The field integral for the entrance and exit pole faces

$$F_{int} = \int_{pole} ds \, \frac{B_y(s)(B_0 - B_y(s))}{2H_{gap}B_0^2} \tag{3.1}$$

If `<f_int1>` or `<f_int2>` is not present then the default value of 0 is used.

<h_gap1>, <h_gap2>
`<h_gap1>` is the half gap of the entrance pole face and `h_gap2` is the corresponding half gap for the exit end.

3.3.3 <cavity_design>

The `<cavity_design>` element can be a child of `<crab_cavity>` (§3.3.4), `<linac_cavity>` (§3.3.8), or a `<rf_cavity>` (§3.3.16). `<cavity_design>` is used to hold design parameters.

The children of `<cavity_design>` are

```
<geometric_beta design|err = "expr">  Relativistic beta of the design particles.
<n_cell value = "num">                Number of cells.
<transit_factor design|err = "expr">  Transit time factor of the cavity.
<transit_factor_derivative  design|err = "expr">
                                      First derivative of the transit time factor.
<mode value = "">                     Mode of operation. Values are "0" or "pi".
```

The `<geometric_beta>` gives the relativistic beta (v/c) of the particles that the cavity is designed to use. The `<transit_factor>` gives the transit time factor of the cavity for a particle with the design velocity. This is basically a factor that calculates the reduction in the acceleration due to the fact that the phase is still rotating during the finite time it takes for a particle to cross the gap. The first derivitive of this with respect to the cavity wavenumber, given by `<transit_factor_derivative>`, is necessary for calculating the phase slip due to velocity mismatches between the particle and the cavity.

3.3.4 <crab_cavity>

A `<crab_cavity>` is an RF cavity used to transversely deflect the beam. That is, the `<crab_cavity>` is excited in a dipole mode as opposed to `<rf_cavity>` and `linac_cavity` elements which are excited in a monipole mode.

The children of `<crab_cavity>` can be:

```
<cavity_design>                       §3.3.3
<orientation>                         §3.4.8
```

```
<field_table>                          §3.4.3
<rf_freq design|err = "expr">          Frequency [Hz].
<phase0 design|err = "expr">           Phase angle [radians].
<phase0_offset turn = "n" design|err = "expr">
                                       Phase offset on n'th pass
                                       through the cavity.
<gradient design|err = "expr">         Accelerating gradient [V/m].
```

In the ultra-relativistic limit, the transverse kick felt by a particle is

```
dPx = gradient * L * cos(phi - twopi * z * rf_freq / c_light) / Energy.
```

Here `phi` = `<phase0>` + `<phase0_offset>`. The `<phase0_offset>` node is used to shift the phase on the n^{th} pass of the beam through the element.

For `phi`=0, the cavity acts as an RF deflector, while for `phi`=$\pm\pi/2$, the cavity acts as a crab cavity. The `<crab_cavity>` always produces its deflection in the xz plane. A deflection in another plane can be produced via use of the `<orientation>` child.

A `<field_table>` child of `<crab_cavity>` can be used to specify cavity higher order modes. Alternatively, a `<field_table>` can be used to specify the cavity fundamental mode itself. In this case, no `<crag_cavity>` node is used, and the `<field_table>` is a child of `<element>`.

3.3.5 <custom>

A `<custom>` element is an element whose properties are defined outside of *AML*. That is, to use a custom element some programmer must write the appropriate custom routines which are then linked in with a program.

3.3.6 <electric_kicker>

An `<electric_kicker>` is a kicker that uses an electric field to produce the kick. The children of `<electric_kicker>` are:

```
<orientation>                          §3.4.8
<scaled_multipole>                     §3.4.9
<x_kick design|err = "expr" />
<y_kick design|err = "expr" />
<x_kick_u design|err = "expr" />
<y_kick_u design|err = "expr" />
<gap design|err = "expr" />
```

`<gap>` specifies the gap between any electrodes but does not affect the value of the kick. The strength of the kicker can be specified using one or the other of `<x_kick>` and `<y_kick>` or `<x_kick_u>` and `<y_kick_u>`. `<kick_x>` and `<kick_y>` gives the kick that a positive particle of unit charge and having the reference energy would receive.

3.3.7 <kicker>

A `<kicker>` gives the beam a kick. This is similar to a `<bend>` except the reference orbit through a `<kicker>` is always a straight line. In addition, a `<kicker>` can apply a displacement to a particle using the `<kicker:x_displacement>` and `<kicker:y_displacement>` attributes.

The children of `<kicker>` are:

```
<orientation>                          §3.4.8
<scaled_multipole>                     §3.4.9
<x_kick design|err = "expr">
<y_kick design|err = "expr">
<x_kick_u design|err = "expr">
<y_kick_u design|err = "expr">
<x_displacement design|err = "expr">
<y_displacement design|err = "expr">
```

<x_kick_u> and <y_kick_u> can be used instead of <x_kick> and y_kick to specify the unnormalized integrated field. A positive <x/y_kick_u>, like a positive <x/y_kick>, gives a kick in the positive direction.

3.3.8 <linac_cavity>

A <linac_cavity> is an RF cavity used for accelerating the beam.

The children of <linac_cavity> can be:

```
<orientation>                          §3.4.8
<field_table>                          §3.4.3
<cavity_design>                        §3.3.3
<rf_freq design|err = "expr">          Frequency [Hz].
<phase0 design|err = "expr">           Phase angle [radians].
<phase0_offset turn = "n" design|err = "expr">
                                       Phase offset on n'th pass
                                       through the cavity.
<gradient design|err = "expr">         Accelerating gradient [V/m].
```

The kick felt by a particle is

```
dE = gradient * L * cos(phi - twopi * z * rf_freq / c_light)
```

Here phi = <phase0> + <phase0_offset>. The <phase0_offset> node is used to shift the phase on the n^{th} pass of the beam through the element.

The convention for dE here is consistent with MAD and LIAR. Note: The MAD8 documentation for an lcavity has a wrong sign. Essentially the MAD8 documentation gives

```
dE = gradient * L * cos(phi + twopi * z * rf_freq / c_light) ! WRONG
```

This is incorrect.

The energy change of the reference particle is just the energy change for a particle with $z = 0$

```
dE(reference) = gradient * L * cos(phi)
```

This, aside from the difference in how the phi is defined, is the major difference between an linac_cavity and a rf_cavity: An rf_cavity does not effect the reference energy.

A <field_table> child of <linac_cavity> can be used to specify cavity higher order modes. Alternatively, a <field_table> can be used to specify the cavity fundamental mode itself. In this case, no <linac_cavity> node is used, and the <field_table> is a child of <element>.

3.3.9 <marker>

A <marker> is a zero length element meant to mark a position. A <marker> does not have any children

3.3.10 <match>

The children of match are:

```
<twiss>
```

See §4.2.

A <match> element is used to match the Twiss parameters between two points. There must be two <twiss> children of <match>: one for the entrance end and one for the exit end. Example:

```
<element name = "match_this">
  <match>
    <twiss at = "ENTRANCE">
      <beta_a design = "13.4" />
      ...
    </twiss>
    <twiss at = "EXIT">
      <beta_a design = "7.2" />
      ...
    </twiss>
  </match>
</element>
```

The transfer map for a match element is a linear matrix such that if the Twiss parameters at the exit end of the element preceding the match element are given by <twiss at = "ENTRANCE"> then the computed Twiss parameters at the exit end of the <match> element will be <twiss at = "EXIT">.

3.3.11 <multipole>

The <multipole> specifies a set of integrated magnetic multipole fields. See §8.1 for the multipole physics definition.

The attributes of <multipole> are:

```
at     = "Place"    "Place" may be: "ENTRANCE", "EXIT", "ENDS", or "CENTER"
                        Default is: "ENDS".
```

The at attribute indicates where to place the multipole longitudinally within the element. In the case of "ENDS", half the multipole is considered to be a the entrance end and half at the exit end.

The children of <multipole> are:

```
<orientation>                              §3.4.8
<kl  n = "order" design|err = "expr" />     Normalized non-skew components.
<ksl n = "order" design|err = "expr" />     Normalized skew components.
<kl_u  n = "order" design|err = "expr" />   Unnormalized non-skew components.
<ksl_u n = "order" design|err = "expr" />   Unnormalized skew components.
<tilt n = "order" design|err = "expr" />    Tilt of the n^th order components.
```

The kl and ksl components are normalized to the reference energy and the kl_u and ksl_u are not (§8.1).

Example:

```
<element>
  <multipole>
    <kl n = "3" design = "0.45" />
```

```
    <ksl n = "4" design = "1.74" />
  </multipole>
</element>
```

3.3.12 <octupole>

The children of <octupole> are

```
<orientation>                          §3.4.8
<scaled_multipole>                     §3.4.9
<k design|err = "expr">
<ks design|err = "expr">
<k_u design|err = "expr">
<ks_u design|err = "expr">
```

An <octupole> has a cubic field dependence. The field strength can either be specified directly using unnormalized normal (<k_u>) and skew (<ks_u>) components or the normalized normal (<k>) and skew (<ks>) focusing strength (§8.1).

3.3.13 <patch>

The children of <patch> are:

```
<orientation>      §3.4.8
```

A <patch> offsets the reference orbit. This is a generalization of *MAD*'s `yrot` and `srot` elements. For example, a positive `orientation:x_offset` offsets the reference orbit after the <patch> in the positive x–direction relative to the reference orbit before the <patch>. Hence, the `x` coordinate of a particle going through a patch with a positive `x_offset` will be decreased.

If the <element> containing the <patch> child also has a <length> child, then the length sets the longitudinal s distance between the previous and next elements. Otherwise, a length will have no effect on particle tracking or transfer map calculations.

3.3.14 <quadrupole>

The children of <quadrupole> are:

```
<orientation>              §3.4.8
<scaled_multipole>         §3.4.9
<k design|err = "expr">
<ks design|err = "expr">
<k_u design|err = "expr">
<ks_u design|err = "expr">
```

A <quadrupole> has a linear field dependence. The field strength can either be specified directly using unnormalized normal (<k_u>) and skew (<ks_u>) components or the normalized normal (<k>) and skew (<ks>) focusing strength (§8.1).

3.3.15 <reference>

<reference> holds information on the reference coordinates at the exit end of an element (§7.3).

The children of <reference> can be:

```
<total_energy value = "expr">   Reference energy
<pc value = "expr">             Reference momentum * c
<beta value = "expr">           Velocity / c
<gamma value = "expr">          Relativistic gamma factor
<s_position value = "expr">     Distance along the reference trajectory.
```

3.3.16 <rf_cavity>

An <rf_cavity> is a non-accelerating cavity that is generally used in a storage ring.

The children of <rf_cavity> can be:

```
<orientation>                          §3.4.8
<field_table>                          §3.4.3
<cavity_design>                        §3.3.3
<rf_freq design|err = "expr">          Frequency [Hz].
<harmonic design|err = "expr">         Harmonic number.
<phase0 design|err = "expr">           Phase angle [radians].
<phase0_offset turn = "n" design|err = "expr">
                                       Phase offset on n'th pass
                                       through the cavity.
<gradient design|err = "expr">         Accelerating gradient [V/m].
```

<phase0>/2π is identical to the lag attribute of *MAD*. The kick felt by a particle is

```
dE = gradient * L * sin(phi + twopi * z * rf_freq / c_light)
```

This is consistent with MAD and LIAR. Here phi = <phase0> + <phase0_offset>. The <phase0_offset> node is used to shift the phase on the n^{th} pass of the beam through the element.

Unlike a <linac_cavity>, the energy of the reference particle through an <rf_cavity> is inverient.

A <field_table> child of <rf_cavity> can be used to specify cavity higher order modes. Alternatively, a <field_table> can be used to specify the cavity fundamental mode itself. In this case, no <rf_cavity> node is used, and the <field_table> is a child of <element>.

3.3.17 <sextupole>

The children of <sextupole> are

```
<orientation>                  §3.4.8
<scaled_multipole>             §3.4.9
<k design|err = "expr">
<ks design|err = "expr">
<k_u design|err = "expr">
<ks_u design|err = "expr">
```

An <sextupole> has a quadratic field dependence. The field strength can either be specified directly using unnormalized normal (<k_u>) and skew (<ks_u>) components or the normalized normal (<k>) and skew (<ks>) focusing strength (§8.1).

3.3.18 <solenoid>

A <solenoid> is an element with a longitudinal magnetic field.

The children of <solenoid> are:

```
<orientation>                  §3.4.8
<scaled_multipole>             §3.4.9
<ksol design|err = "expr" />     Normalized Solenoid field
<dksol design|err = "expr" />    Normalized solenoid field derivative
<ksol_u design|err = "expr" />   Solenoid field
<dksol_u design|err = "expr" />  Solenoid field derivative
```

The field strength can either be specified directly using unnormalized `<ksol_u>` or the normalized focusing strength `<ksol>` (§8.1).

3.3.19 <taylor_map>

A `<taylor_map>` is a set of six Taylor series, one for each output coordinate. Each Taylor series has a number of terms. Each term is of the form

$$x_j(\text{out}) = C \cdot \Pi_{i=1}^{6} x_i^{e_i}(\text{in}) \tag{3.2}$$

where $\mathbf{x} = (x, p_x, y, p_y, z, p_z)$. The syntax for a `<taylor_map>` has the form

```
<taylor_map>
  <term i_out = "integer" coef = "real" exp = "e1 e2 e3 e4 e5 e6" />
  ...
</taylor_map>
```

where `coef` is a real number, `exp` is a string of 6 non-negative integers giving the exponents for the term, and `i_out` is an integer, between 1 and 6, which gives the index of the output coordinate of the term. For example:

```
<taylor_map>
  <term i_out = "1" coef = "0.92" exp = "1 0 1 0 0 0" />
  <term i_out = "4" coef = "2.73" exp = "0 0 2 0 0 1" />
  <term i_out = "6" coef = "13.75" exp = "2 0 1 0 0 0" />
  ...
</taylor_map>
```

The middle Taylor term in the above example, expressed as an equation, looks like:

$$p_y(\text{out}) = 2.73 \cdot y^2(\text{in})\, p_z(\text{in}) \tag{3.3}$$

or, putting it another way, the middle term in the example above represents the U_{4335} matrix

By default, a `<taylor_map>` element starts out as the unit map. That is, a `taylor_map` element starts with the following 6 terms

```
<taylor_map>
  <term i_out = "1" coef = "1.0" exp = "1 0 0 0 0 0" />
  <term i_out = "2" coef = "1.0" exp = "0 1 0 0 0 0" />
  <term i_out = "3" coef = "1.0" exp = "0 0 1 0 0 0" />
  <term i_out = "4" coef = "1.0" exp = "0 0 0 1 0 0" />
  <term i_out = "5" coef = "1.0" exp = "0 0 0 0 1 0" />
  <term i_out = "6" coef = "1.0" exp = "0 0 0 0 0 1" />
</taylor_map>
```

3.3.20 <wiggler>

A <wiggler> is a periodic array of alternating bends.

The children of <wiggler> are:

<wiggler_field> §3.3.20
<wiggler_params> §3.3.20

There are two types of wigglers. Those that that are described using a magnetic field map ("map type") and those that are described assuming a periodic field ("periodic type"). The periodic type is defined by specifying <wiggler_params>. For the "map" type wigglers the field is given by <wiggler_field>.

<wiggler_field>

```
<wiggler_field>
  <strength design = "expr" />
  <w_term coef = "Value" k_x = "Value" k_y = "Value"
                                  k_z = "Value" phi_z = "Value" />
</wiggler_field>
```

<wiggler_field> gives the wiggler field magnetic field as a sum of <w_term> terms as outlined in §8.3. <strength> is used to scale the magnetic field. The default value for <strength> is 1.0.

<wiggler_params>

The children of <wiggler_params> are:

```
<b_max design|err = "expr" />
<length_pole design|err = "expr" />
```

<b_max> is the maximum value of the magnetic field on–axis. <length_pole> is the pole length. The magnetic field in the wiggler is given by:

$$B_y(s) = B_{\max} \cos(ks), \tag{3.4}$$

where

$$k \equiv \frac{2\pi}{l_{\text{pole}}}. \tag{3.5}$$

3.4 Definition of General Element Properties

The general properties of an <element> are set via the child nodes listed in Table 3.2. These children are defined below, along with their attributes and children.

3.4.1 <aperture>

The <aperture> node is used to describe the aperture of an element.

The attributes of <aperture> are:

```
at     = "Place"    "Place" may be: "ENTRANCE", "EXIT", or "BOTH"
                        Default is: "EXIT".
```

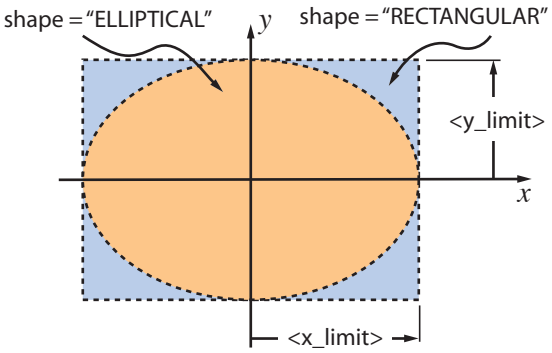

Figure 3.2: Apertures for elliptical and rectangular collimation

```
shape   = "Shape"     "Shape" may be: "ELLIPTICAL", "RECTANGULAR",
                          "CIRCLE", or "DIAMOND". Default is: "RECTANGULAR".
orientation_dependent = "T/F"
                      "T/F" may be: "true" or "false".
                          Default is "false".
```

The children of <aperture> are:

```
<orientation>                          §3.4.8
<x_limit design|err = "expr">
<y_limit design|err = "expr">
<xy_limit design|err = "expr">          Used with shape = "CIRCLE".
```

These children also have an optional attribute:

```
side    = "Side"      "Side" may be "+", "-", or "BOTH".
                          Default is "BOTH".
```

Example:

```
<aperture at = "BOTH">
  <x_limit design = "0.04" err = "0.0001" />
  <y_limit design = "0.005" side = "+" />
</aperture>
```

<x_limit>, <y_limit> specify the half–width of the aperture of an element as shown in figure 3.2. If the limit is only on one side then the side attribute of <x_limit> or <y_limit> can be used. With CIRCLE, xy_limit is used.

By default, an aperture is independent of the <orientation> children of its <element>. Conceptually, this is equivalent to having the beam pipe, which the aperture is part of, physically isolated from the element. However, if orientation_dependent is set to "true" then an element's orientation will move the aperture. Note that the <orientation> of any of the other children of the <element> will not affect the aperture; only the <orientation> child of the <element> will. Example:

```
<element>
  <quadrupole>
    <orientation>
      <tilt design = "pi/4" />
    </orientation>
  </quadrupole>
```

```
    <aperture orientation_dependent = "true">
      <x_limit design = "0.045" />
    </aperture>
    <orientation>
      <tilt err = "0.01" />
    </orientation>
  </element>
```

Here the <orientation> child of <quadrupole> will not affect the aperture but the <orientation> child of <element> will.

3.4.2 <description>

The <description> node is used for describing an element.

The children of <description> are:

```
<type string = "string">            Type of element.
<alias string = "string">           An alias name.
<ref string = "string">             Reference description.
<comment string = "string">         General comments §2.5.
<mad_element string = "string" />   If applicable: Original MAD element name.
```

Example:

```
<description>
  <type string = "half-shell" />
  <ref string = "http://www.anywhere.edu" />
</description>
```

3.4.3 <field_table>

A <field_table> gives the electric or magnetic field of an element in tabular form. A field table can be a child of <element> or can be a child of <rf_cavity>. When it is a child of <rf_cavity>, a <field_table> represents a cavity higher order mode.

The attributes of <field_table> are:

```
symmetry = "type"    May be "NONE" or "AXIAL" Default is "NONE"
x_symmetry = "type"  May be "NONE", MIRROR"", or "ANTI_MIRROR"
                            Default is "NONE"
y_symmetry = "type"  May be "NONE", MIRROR"", or "ANTI_MIRROR"
                            Default is "NONE"
reference_gradient = "Value"  Gradient reference.
```

reference_gradient is only used when <field_table> is a child of <rf_cavity>. In this case, all the fields are to be scalled by the value of gradient / reference_gradient.

The children of <field_table> are:

```
<frequency design|err = "expr" />
<phase design|err = "expr" />
<point x|y|s = "value" Bx|By|Bs = "value" Ex|Ey|Es = "value" phase = "value" />
```

If <field_table> is a child of <rf_cavity>, the <phase> will be relative to the overall cavity phase.

The three symmetry attributes can be used to reduce the number of points in the table. With `symmetry` set to `"AXIAL"` only the field in the (x, s) plane are needed. With `x_symmetry` set to `"MIRROR"`, the magnetic (or electric) fields have a mirror symmetry under the transformation $x \to -x$:

$$\begin{aligned} B_x(-x, y, s) &= -B_x(x, y, s) \\ B_y(-x, y, s) &= B_y(x, y, s) \\ B_s(-x, y, s) &= B_s(x, y, s) \end{aligned} \tag{3.6}$$

If set to `ANTI-MIRROR` the field behaves as:

$$\begin{aligned} B_x(-x, y, s) &= B_x(x, y, s) \\ B_y(-x, y, s) &= -B_y(x, y, s) \\ B_s(-x, y, s) &= -B_s(x, y, s) \end{aligned} \tag{3.7}$$

For `y_symmetry`, `"MIRROR"` symmetry transforms like

$$\begin{aligned} B_x(x, -y, s) &= B_x(x, y, s) \\ B_y(x, -y, s) &= -B_y(x, y, s) \\ B_s(x, -y, s) &= B_s(x, y, s) \end{aligned} \tag{3.8}$$

and `"ANTI_MIRROR` symmetry obeys

$$\begin{aligned} B_x(x, -y, s) &= -B_x(x, y, s) \\ B_y(x, -y, s) &= B_y(x, y, s) \\ B_s(x, -y, s) &= -B_s(x, y, s) \end{aligned} \tag{3.9}$$

For example

```
<field_table>
  <point x = "0" y = "0.02" s = "0" Ex = "-1.34" Ey = "0" Es = "0" />
</field_table>
```

3.4.4 <floor>

`<floor>` specifies the position of an element or a lattice in the global coordinate system. See Chapter 7 for the definition of the global coordinate system.

The attributes of `<floor>` are:

```
element = "Element-name"   Name of the lattice element for which the global
                             position is specified.
                             Used only when <floor> is a child of <lattice>.
                             Default is the Beginning "zeroth" element.
at      = "Place"          "Place" may be:  "ENTRANCE", "EXIT", or "MIDDLE."
                             Default is "EXIT".
```

children of `<floor>` are:

```
<x design|err = "expr" />
<y design|err = "expr" />
<z design|err = "expr" />
<phi design|err = "expr" />
<psi design|err = "expr" />
<theta design|err = "expr" />
<s design|err = "expr" />
```

3.4.5 <instrument>

An <instrument> child indicates that there is some kind of instrumentation attached to this element such as a beam position monitor. <instrument> has an attribute

```
type = "name"
```

name can be any string. MAD compatible names are

```
"hmonitor"
"vmonitor"
"monitor"
"instrument"
```

3.4.6 <length>

The Length node describes an elements length.

```
<length design|err = "expr" />                Length.
```

<length> is the path length of the reference trajectory. Note that for a wiggler the reference trajectory, which is straight, does not correspond to the path taken by any actual particle. Thus for a wiggler, the actual path length of a particle will never be the same as the wiggler's length.

3.4.7 <methods>

The <methods> node specifies information on how calculations are to be performed. This is necessarily program dependent.

The children of <methods> are:

```
<tracking value = "expr" />             Method used for tracking particles.
<transfer_map_calc value = "expr" />    Method used for calculating transfer matrices.
<n_steps value = "expr" />              Number of integration steps in a calculation.
<relative_tolerance value = "expr" />   Tolerance used in adaptive step calculations.
<absolute_tolerance value = "expr" />   Tolerance used in adaptive step calculations.
<integrator_order value = "expr" />     Order of symplectic integrator.
```

3.4.8 <orientation>

The <orientation> element defines the orientation of the physical element with respect to the local reference frame. The exception here is in a <patch> element the <orientation> attributes modify the reference orbit itself.

The attributes of <orientation> are:

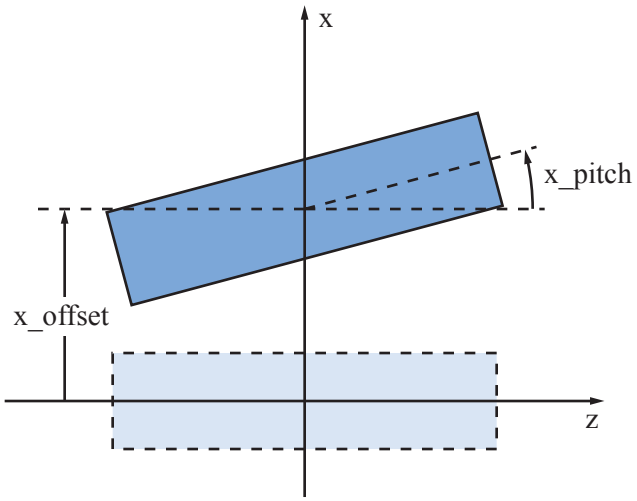

Figure 3.3: Geometry of Pitch and Offset attributes

```
origin = "Place"      "Place" may be: "ENTRANCE", "CENTER", or "EXIT"
                          Default is: "CENTER".
```

The children of <orientation> are:

```
<x_offset design|err = "expr" />
<y_offset design|err = "expr" />
<s_offset design|err = "expr" />
<x_pitch design|err = "expr" />
<y_pitch design|err = "expr" />
<tilt design|err = "expr" />
<roll design|err = "expr" />          Used with <bend> elements only.
<pz_offset design|err = "expr" />     Used with <patch> elements only.
```

Example:

```
<orientation origin = "CENTER">
  <x_offset design = "3.1 * x_nominal" />
  <y_pitch  design = "30 * degrees" />
  <tilt     design = "pi/4" />
</orientation>
```

<x_offset> translates an element in the local x–direction as shown in Figure 3.3. Similarly, <y_offset> and <s_offset> translate an element along the local y and s–directions respectively.

<origin> gives the origin of <x_pitch> and <y_pitch> rotations. The default <origin> is "CENTER". The <x_pitch> attribute rotates an element about the y–axis so that the exit face of the element is displaced in the $+x$–direction as shown in figure 3.3. Similarly the <y_pitch> attribute rotates an element about the x–axis so that the exit face of the element is displaced in the $+y$–direction. By default, the rotations are about the center of the element which is in contrast to the dtheta and dphi misalignments of *MAD* which rotate around the entrance point. In terms of rotation angle

```
x_pitch =  dtheta
y_pitch = -dphi
```

Like *MAD*, offsets are applied before pitches.

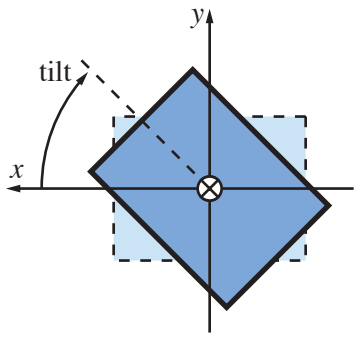

Figure 3.4: Geometry of a tilt. Positive z (the direction of travel of the beam) is into the page since the (x, y, z) coordinate system is a right handed one. Notice that with this view, positive tilts are clockwise.

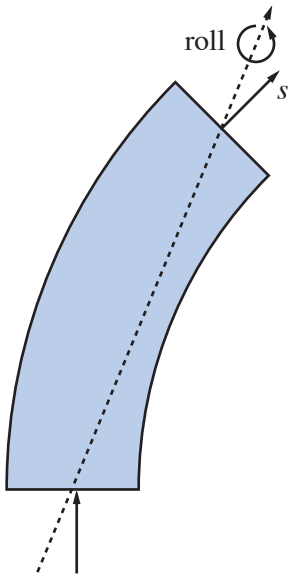

Figure 3.5: Geometry of a Roll

The tilt attribute rotates the element in the (x, y) plane as shown in figure 3.4. For a bend the rotation axis is the z-axis at the entrance face independent of any `<origin>` setting. The reference orbit is also rotated with the element. A bend with a tilt of $\pi/2$ will bend a beam upward vertically.

The `<roll>` attribute is only used for bends and rotates the bend, along an axis that runs through the entrance point and exit point as shown in figure 3.5. A `<roll>` does not affect the reference orbit. The major effect of a `<roll>` is to give a vertical kick to the beam.

3.4.9 <scaled_multipole>

A `<scaled_multipole>` is used to specify magnetic multipole fields and is meant to represent a "perturbation" of the main fields in an element.

The children of `<scaled_multipole>` are:

```
<orientation>                                  §3.4.8
<k_coef  n = "order" design|err = "expr" />    Normalized non-skew components.
<ks_coef n = "order" design|err = "expr" />    Normalized skew components.
<tilt n = "order" design|err = "expr" />       Tilt of the n^th order components.
<radius design|err = "expr" />                 Measurement radius. Default is 1.0.
```

See (§8.1.2). A <scaled_multipole> node can be a child of one of the magnetostatic children of <element>: <bend>, <kicker>, <octupole>, etc. The strength of the components of a <scaled_-multipole> scales with the strength of its parent (Section §8.1.2).

A <radius> child can be used to scale the multipole field values to the radius at which the multipole fields where measured. See §8.1.2 for more details. The following example shows a quadrupole with a sextupole perturbation:

```
<element name = "Quad_1">
  <quadrupole>
    <k design = "0.9843" />
    <scaled_multipole>
      <k_coef n = "3" design = "0.015" />
      <ks_coef n = "3" design = "0.072" />
    </scaled_multipole>
  </quadrupole>
</element>
```

3.4.10 <state>

The attributes of <state> are:

```
is_on  = "On-Off"          "On-Off" may be "true" or "false".
                           Default is "true".
```

There are no children of <state>. Example:

```
<state is_on = "false" />
```

The <is_on> attribute is used to turn an element off. Turning an element off essentially converts it into a drift. Note: <is_on> does not affect any apertures that are node.

3.4.11 <superimpose>

In an actual accelerator different machine elements can overlap spatially. For example, a quadrupole may be partially or totally inside a solenoid magnet. To model this, *AML* uses the concept of `superpositioning` of machine elements. A simple example shows how this works:

```
  <element name = "SOL" >
     ...
  </element>

<element name = "Q1">
  <quadrupole>
     ...
  </quadrupole>
  <superimpose offset = "0.8" ele_origin = "ENTRANCE"
               ref_element = "SOL" ref_origin = "CENTER" />
</element>
```

In this example the Q1 quadrupole element is superimposed over the reference element with the entrance end of Q1 0.8 meters from the middle of the reference element (SOL in this case). If the SOL solenoid is 4 meters in length, and Q1 is 1 meter in length, when the sector is expanded the expanded elements will be:

```
Element   Length
-------   ------
SOL|1     1.8
Q1|SOL    1.0
SOL|2     1.2
```

The SOL|1 and SOL|2 elements represent the space in the solenoid that is not overlapped by the quadrupole while the Q1|SOL element represents the space that where the solenoid and quadruple do overlap.

The convention *AML* uses to construct the names for these hybrid elements is to concatenate the individual names of the elements that make up the hybrid using the | character as a separator. For hybrid elements that are outside the region of the superposition (SOL|1 in the example) the string |n is appended to the element name where n is an integer starting with 1.

The list of hybrid elements constructed by *AML* during lattice expansion is what is wanted for calculations such as particle tracking. The standard *AML* parsing utility, UAP, also keeps a list of the original elements (in this case, SOL and Q1). Programs that read in *AML* lattice files using UAP can use this second list (called the master list) to do the appropriate bookkeeping should changes in machine attributes are desired.

There is no limit to the number of levels of superposition which are supported. In the above example, another element can be superposed on the Q1 element; this would potentially result in a combination of the new element, Q1, and the solenoid, depending on the lengths and positionings of the elements in the superposition. The resulting superposed element would be named newname|Q1|SOL.

If there are multiple instances of the reference element in a lattice then the superposition will be applied to each. If a superposed element extends past the end of its reference element, then the remainder of the superposed element will be superimposed over whatever elements follow the reference element in any and all lattices where the reference element appears. For a circular lattice, if a superimposed element extends beyond the ends of the lattice then the superimposed element is considered to "wrap" around the ends.

The attributes that control where a superposition is placed are:

```
offset      = "offset"             Offset in meters.
ref_element = "reference element"  Reference element name.
ele_origin  = "element origin"     May be "ENTRANCE", "CENTER", "EXIT",
                                    or "SECTOR_ORIGIN". Default is "CENTER".
ref_origin  = "reference origin"   May be "ENTRANCE", "CENTER", "EXIT".
                                    or "SECTOR_ORIGIN". Default is "CENTER".
```

The setting of SECTOR_ORIGIN for ele_origin can only be used when the <superposition> is a child of a <sector> (§5.1). The setting of SECTOR_ORIGIN for ref_origin can only be used when the reference element is a sector.

3.4.12 <wake>

A <wake> is a representation of a wakefield. There are no attributes to <wake>. The children of <wake> are:

```
<wake_modes>
<wake_table>
<e_loss design|err = "expr">
```

The parameter <e_loss> represents the mean energy loss to a bunch due to short-range wakefields. <e_loss> is used to calculate the energy loss of the reference particle via:

$$dE(wakes) = -e_loss * beam[n_particles] * e_charge. \tag{3.10}$$

The dynamics of the wakefield can be represented by either a table (<wake_table>) or a set of modes (<wake_modes>). Both <wake_table> and <wake_modes> can take as an attribute range, where the range can be SHORT-RANGE or LONG-RANGE. The physics of the wakefields is discussed in greater detail in §8.4.

A <wake_table> has one or more <point> children, which specify the longitudinal wake function (in V/C/m) and the transverse wake function (in V/C/m^2):

```
<wake_table range = "SHORT-RANGE">
  <!-- [m] [V/C/m]  [V/C/m^2] -->
  <point z = "0.0000E+00" Wz = "1.61125E+15" Wt = "0.00000E+00" />
  <point z = "1.0000E-05" Wz = "1.44516E+15" Wt = "1.30560E+15" />
  <point z = "2.0000E-05" Wz = "1.38148E+15" Wt = "2.50665E+15" />
</wake_table>
```

A <wake_modes> node has one or more <mode> children. Each <mode> has attributes freq, R_over_Q, Q, polarization_angle, and m. The <wake_modes> also has a freq_spread attribute; this attribute indicates the RMS fractional spread in actual modal frequencies when the elements with the <wake> are instantiated.

```
<wake_modes range = "LONG-RANGE" freq_spread = "0.003">
  <mode freq = "1.6506e9" R_over_Q = "0.76" Q = "7.0e4" m = "1" />
  <mode freq = "1.6506e9" R_over_Q = "0.76" Q = "7.0e4" m = "1"
                    polarization_angle = "90*degrees"           />
</wake_modes>
```

In the example above, there are two polarizations of a dipole mode at a frequency of 1.6506 GHz; one polarization is horizontal, the other is vertical.

Chapter 4

Specifying the Beam and Beam Parameters

AML provides several nodes for the purpose of defining the beams which are expected to propagate in a beamline. These tools are described below.

Note that *AML* uses a canonical coordinate system to define the positions and trajectories of the beam. The definition of the coordinate system can be seen in §7.3. Applications which use an alternate coordinate system (geometric coordinates, action-angle, etc.) can make appropriate conversions from the coordinates or beam matrices in the lattice file.

4.1 <beam>

The `<beam>` node defines parameters of the beam. A `<beam>` node can be the child of the following nodes:

- The `<laboratory>` node – a `<beam>` which is the child of `<laboratory>` is common to all machines and all paths. Only one global `<beam>` node is permitted.
- A `<machine>` node (§5.2). Only one `<beam>` child is permitted per `<machine>` node.
- A `<path>` node (§5.2.4). Only one `<beam>` child is permitted per `<path>` node.

Possible attributes of `<beam>` are

```
name = "String"
ref = "String"
inherit = "String"
```

The `name` attribute allows later reference to `beam` parameters (§2.6.1). The `ref` attribute allows a `beam` node to be a reference to a named `beam` node defined elsewhere, while the `inherit` attribute allows a `beam` node to duplicate some or all of the parameters of a `beam` node defined elsewhere.

The child nodes of <beam> are:

<position>	§4.1.1	Beam centroid position
<sigma_matrix>	§4.1.2	Beam sigma matrix
<n_particles design|err = "expr">		Number of particles in a bunch.
<n_bunches design|err = "expr">		Number of bunches in a beam.
<particle type = "type">		POSITRON, ELECTRON, etc.
<mass design|err = "expr">		Mass of particle.
<total_energy design|err = "expr">		Total particle energy
<pc design|err = "expr">		Particle momentum * c_light
<particle_charge design|err = "expr">		Particle charge.
<emittance_a design|err = "expr">		"Horizontal" emittance.
<emittance_b design|err = "expr">		"Vertical" emittance.
<emittance_z design|err = "expr">		"Longitudinal" emittance.
<norm_emittance_a design|err = "expr">		Emittance * gamma.
<norm_emittance_b design|err = "expr">		Emittance * gamma.
<sig_z design|err = "expr">		Longitudinal beam size.
<sig_e design|err = "expr">		Energy spread.
<sig_t design|err = "expr">		Time spread.
<num_bunches design|err = "expr">		Number of bunches.
<beam_current design|err = "expr">		Beam current.

Note that the emittances in the beam definition are explicitly normal mode and not projected. This is indicated by use of "a" and "b" to reference the emittances, rather than "x" and "y".

Possible type values for <particle> are:

POSITRON	default
ELECTRON	
PROTON	
ANTIPROTON	
MUON	
ANTIMUON	
NUCLEUS	

4.1.1 <position>

The <position> node specifies phase space coordinates; <position> is a child of <beam> only.

The children of <position> are

<x design|err = "expr">	Horizontal position.
<p_x design|err = "expr">	Horizontal momentum.
<y design|err = "expr">	Vertical position.
<p_y design|err = "expr">	Vertical momentum.
<z design|err = "expr">	Longitudinal position.
<deltap design|err = "expr">	Fractional momentum error.

4.1.2 <sigma_matrix>

<sigma_matrix> defines the beam shape in phase space using a 6x6 sigma matrix with three emittances. <sigma_matrix> is a child of <beam>.

The children of <sigma_matrix> are:

<ij design|err = "expr" />	Sigma_ij component.

`i` and `j` are the matrix induces and can be between `"1"` and `"6"`.

Example:

```
<sigma_matrix>
  <16 design = "3.7e-6" />
</sigma_matrix>
```

4.2 <twiss>

A `<twiss>` node specifies the Twiss and coupling parameters. As with the beam emittances defined in §4.1, the beta and alpha functions are normal-mode and thus specified with `a` and `b` rather than `x` and `y`. The coupling components are defined in §8.5.

A `<twiss>` node can be a child of a `<lattice>` node (§5.2.2) or of a `<match>` node (§3.3.10). Each `<lattice>` node is allowed one `<twiss>` child; each `<match>` node must have two `<twiss>` children. The attributes of `<twiss>` are:

```
at     = "Place"
```

The `at` attribute is used only in the case of a `<twiss>` child of a `<match>` node; its allowed values are `ENTRANCE` and `EXIT`.

The children of `<twiss>` are:

```
<twiss>
  <beta_a design|err = "expr">
  <beta_b design|err = "expr">
  <alpha_a design|err = "expr">
  <alpha_b design|err = "expr">
  <phase_a design|err = "expr">
  <phase_b design|err = "expr">
  <eta_x design|err = "expr">
  <eta_y design|err = "expr">
  <etap_x design|err = "expr">
  <etap_y design|err = "expr">
  <c11 design|err = "expr">      <!-- Initial coupling matrix numbers -->
  <c12 design|err = "expr">
  <c21 design|err = "expr">
  <c22 design|err = "expr">
</twiss>
```

Example:

```
<twiss>
   <beta_a design = "12.4" />
   <beta_b design = "3.45 err = 0.043" />
</twiss>
```

Chapter 5

Defining the Lattice

The Accelerator Markup Language uses two main constructs, the <sector> and the <machine>, to organize components, beam definitions, and other data elements into beam lines.

A <sector> is closely related to a MAD line definition: an ordered sequence of elements which can be used in turn to build yet more complex sequences of elements. A <sector> can be used to represent something as simple and generic as half a FODO cell, or something as complex and specific as an actual, existing accelerator.

A <machine>, by comparison, represents an actual, complete accelerator complex, accelerator, or section of accelerator which is intended to be instantiated by an application program. There is no particularly close analogy between <machine> and any MAD language concept, although there are similarities to the MAD use statement.

One important feature of a <machine> node is that, while a large <machine> can be built up out of smaller <machine> nodes, there is no implication that sequential <machine> children represent sequential sections of accelerator. In other words,

```
<machine name = "DRComplex">
  <machine ref = "NDR" />
  <machine ref = "SDR" />
</machine>
```

does not imply that the "SDR" machine sequentially follows the "NDR" machine. The relationships between the <machine> children of a <machine> node must be explicitly defined using the <branch> and <path> nodes as explained below.

5.1 <sector>

The accelerator lattice is defined by a sequence of physical elements. This sequence may be broken down into sub-sequences call <sector>s. A <sector> is a list of <elements>, other <sector>s, and <branch> nodes. For example:

```
<sector name = "this_sect">
  <element ref = "Q1" />
  <element ref = "Q2" />
  <sector ref = "sub_sect" repeat = "2" />
</sector>
```

```
<sector name = "sub_sect">
  <element ref = "Q3" />
  <element ref = "Q4" />
</sector>

<machine name = "CESR">
  <sector ref = "this_sect" />
</machine>
```

A <sector> with a name attribute gives the definition of a <sector>. In the above example the <sector> named this_sect is defined. This <sector> contains three element, the last of which is a reference to another sector called sub_sect. The definition of <sub_sect> needs to appear somewhere else. Sectors may only be defined once but they can be referred to in multiple places. When the CESR machine is expanded (§5.3) the list of elements will be

```
Q1, Q2, Q3, Q4, Q3, Q4
```

Using the ref attribute within a <element> means that the element is a copy of some element defined elsewhere.

A <sector> node may have a ref attribute or a name attribute but not both. An unnamed (not having a name or ref attribute) <sector> definition can appear within the definition of another <sector>. For example, this_sect could be rewritten:

```
<sector name = "this_sect">
  <element ref = "Q1" />
  <element ref = "Q2" />
  <sector repeat = "2">
    <element ref = "Q3" />
    <element ref = "Q4" />
  </sector>
</sector>
```

A <sector> definition can also be used in place of an <element> definition. In the above example, for instance, if the definition of element Q1 (which is not shown) is removed from the lattice file and replaced with a sector definition with name Q1, then this sector will be used wherever a reference to element Q1 appears. Thus *AML* has the rule that no <element> and <sector> can have the same name.

5.1.1 Superposition Sectors

In some circumstances it is not practical or optimal to specify a sector as a consecutive list of contiguous elements. For example, in an "as-built" beamline one often knows the positions of elements or the distances between them, and it is inconvenient in this case to generate drift elements of appropriate lengths between the non-drift elements. To simplify the process of defining such a beamline, a <sector> may be defined by specifying a length for it and by specifying a number of elements to be superimposed (§3.4.11) on top of it. For example:

```
<sector name = "super_sect" length = 19.1 sector_origin = "mark1">
  <element ref = "Q1">
    <superimpose offset = "3.5" />
  </element>
  <sector ref = "sub_sect">
    <superimpose offset = "14.7" ref_element = "Q1" />
  </sector>
```

```
  ...
</sector>
```

Such a sector will be termed a `superposition sector`. The rules for constructing a superposition sector are as follows:

- The optional `sector_origin` attribute selects a `<sector>` child element as the reference point. When this `<sector>` is superimposed as a child of another `sector`, the `ele_origin` setting of the superposition can then be set to `SECTOR_ORIGIN`.
- All the `<element>` or `<sector>` children within a superposition sector must have a `<superimpose>` child; note that in this case, an element or sector which uses a `ref` attribute can take a `<superimpose>` child, which is not permitted in the simpler case of elements which are superimposed on one another (§3.4.11).
- The default reference point for placing a `<sector>` child element is the beginning of the sector. An explicit reference element can only be another child element that appears prior in the list. Thus, in the above example, the `Q1` element cannot use the `sub_sect` sector as a reference element since it is a sector and it appears after the `Q1` element.
- All `<sector>` children within a superposition sector must themselves be superposition sectors; in the example above, it is assumed that the definition of `sub_sect` has a `length` attribute and generally follows the rules for construction of a superposition sector.
- No `<element>` or `<sector>` child can extend beyond the boundaries of a superposition sector.

5.1.2 Reflection and Repetition of Sectors

Sectors may be reflected and repeated using the attributes

```
reflection = "boolean"
repeat = "number"
```

For example:

```
<sector name = "a_sect">
  <sector reflection = "true" repeat = "3" />
    <element ref = "A" />
    <element ref = "B" />
   </sector>
 </sector>
```

The expansion of `a_sect` (§5.3) produces

```
B, A, B, A, B, A
```

Only reference and unnamed sectors may have `reflection` or `repeat` attributes.

The `<element>` children of `<sector>` can also take a `repeat` attribute, which works in the exact same way as the `repeat` attribute of a `<sector>`.

5.1.3 Sectors with Arguments

A `<sector>` node can take arguments. using the attribute

```
args = "comma separated list of arguments''
```

Example:

```
<sector name = "sect_with_args" args = "QF, QD">
  <element ref = "drift1" />
  <element ref = "QF" />
  <element ref = "drift2" />
  <element ref = "QD" />
</sector>
```

The dummy arguments `QF` and `QD` are replaced by the actual arguments when the sector is referred to. For example, upon expansion (§5.3), the following:

```
<sector name = "a_sector">
  <sector ref = "sector_with_args" args = "Q01, Q02" />
</sector>
```

would produce the sequence of elements:

```
drift1, Q01, drift2, Q02
```

5.1.4 <list>

A replacement `<list>` (equivalent to a MAD `list`) defines a name that is to be successively replaced, in sequence, by the elements in the list. Example:

```
<list name = "Z">
  <element ref = "A" />
  <element ref = "B" />
  <element ref = "C" />
</list>
```

If this `<list>` name is used in a `<sector>` then, during expansion, the elements from the `<list>` are successively used. For example:

```
<sector>
  <element ref = "Z" />
  <element ref = "D1" />
  <element ref = "Z" />
  <element ref = "D2" />
  <element ref = "Z" />
  <element ref = "D3" />
  <element ref = "Z" />
</sector>
```

would expand to:

```
A, D1, B, D2, C, D3, A
```

Notice that an `<element>` node is used to refer to the `<list>`. Like a `<sector>`, the definition of a `<list>` can contain `sector` subnodes. For example:

```
<list name = "Z">
  <element ref = "A" />
  <sector repeat = "2" />
    <element ref = "B" />
    <element ref = "C" />
  </sector>
</list>
```

In this case, the `<sector>` above would expand to:

```
A, D1, B, C, B, C, D2, A, D3, B, C, B, C
```

5.1.5 Prefixes: Differentiating Elements with the Same Name

Large machines typically have sets of elements with the same name. This makes things simpler in the initial design stage but at some point there needs to be a mechanism for differentiating different physical elements. For example, in simulations of optics errors, one needs to be able to specify individual elements. For such cases, *AML* has the `prefix` attribute. For example:

```
<element name = "Q1" />

<sector name = "sect1">
  <element ref = "Q1" prefix = "S1" />
  <element ref = "Q1" prefix = "S2" />
  ...
</sector>
```

When the sector `sect1` is expanded, the name of the first instance of `Q1` will be changed to `S1.Q1`, and the name of the second instance will be changed to `S2.Q1`.

Prefixes can be applied to entire sectors:

```
<sector name = "sect2">
  <sector ref = "sect1" prefix = "regionA" />
  <sector ref = "sect1" prefix = "regionB" />
  ...
</sector>
```

When the sector named `"sect2"` is expanded the element list will look like:

```
<element name = "Q1" prefix = "regionA.S1" />
<element name = "Q1" prefix = "regionA.S2" />
...
<element name = "Q1" prefix = "regionB.S1" />
<element name = "Q1" prefix = "regionB.S2" />
```

The general format is

```
prefix = "prf1.prf2. ... .prfN"
```

where `prf1`, `prf2`, etc. are individual prefixes used with `<sector>` nodes. The last prefix `prfN` may be a prefix used with either a `<sector>` or an `<element>` node. The individual prefixes cannot contain a dot (".") character.

An alternative way to give unique names to separate sectors that are repeated is to use the string `#COUNT` within the tag. For example, sector `sect1` in the above example could have been written:

```
<sector name = "sect1">
  <element ref = "Q1" repeat = "2" prefix = "S#COUNT" />
  ...
</sector>
```

To modify, for example, the first `"Q1"` element without affecting the others one can use a `<post_set>` command. For example:

```
<post_set attribute = "regionA.S1.Q1[orientation:x_offset]" value = "0.03" />
```

the wild cards `"*"` and `?` and `"..."` can be used. `*` matches a single prefix component, while the latter matches any number of prefixes: for example, `"*.Q1"` matches all elements with a single prefix followed by `"Q1"`, while `"...Q1"` matches all elements with any number of prefixes followed by `"Q1"`, while `"*.B...Q1"` looks for any initial prefix, a `"B"` as the second prefix, and then any number of additional prefixes followed by `"Q1"`.

5.2 <machine>

A <machine> node is used to define an actual accelerator region, accelerator, or accelerator complex which is ready to be expanded and instantiated. There may be multiple <machine> definitions in a lattice file but only the last <machine> in the file is used to define a lattice. In this regard, a <machine> node is similar to the MAD use statement.

The attributes of <machine> can be:

```
name = "name"
ref =  "name"
```

The name attribute allows a <machine> node to be named; the ref node allows a <machine> node to be referred to in another definition. These properties are mainly used in cases where a complex <machine> is constructed from other <machine> nodes, as described below.

A <machine> must have one or more <sector> children, or else one or more <machine> children. A <machine> cannot have both <sector> and <machine> children. For example:

```
<machine name = "NDR">
   <sector ref = "sect:NDR" />
   <sector ref = "sect:SDR" />
</machine>
<machine name = "DRComplex">
   <machine ref = "NDR" />
   <machine ref = "SDR" />
</machine>
```

Note that, unlike a <sector>, the consecutive <machine> children of a <machine> node are not considered to be sequential in the overall accelerator complex. They can be sequential, or connected in a more complex manner, or completely unconnected to one another. The connection of <machine> nodes is described in §5.2.3.

Besides <sector> or <machine>, a <machine> node can have the following children:

<beam>	§4.1	Initial beam parameters.
<comment>	§2.5	Simple documentation.
<doc>	§2.5	Documentation.
<include>	§2.2	File inclusion.
<lattice>	§5.2.2	Lattice parameters.
<path>	§5.2.4	Permissible Beam Pathway.
<magnetic_design_polarity>		Design charge polarity of the beamline.

The <magnetic_design_polarity> has one attribute:

```
sign = "polarity"     May be: "NONE" (default), "+", "-"
```

This is used to set the behavior of magnetostatic elements when beams of various polarities propagate through them in various directions (§5.2.5). When sign="+", a positively charged particle moving through the beamline in the forward direction sees the design polarity of the elements (horizontally focusing in quads with positive k, etc). When sign="-", it is negatively charged particles which see the design polarity. Finally, when sign="NONE", all beams passing through the beamline see the design polarity. sign="NONE" implies that beams going in opposite directions have opposite polarities.

5.2.1 Machines with Recirculation

A <machine> with <sector> children is similar to a <sector> with sector children with one important difference. Example:

```
<machine name = "ERL">
  <sector ref = "injector" />
  <sector ref = "linac" />
  <sector ref = "turn_around" />
  <sector ref = "linac" />
  <sector ref = "to_dump" />
</machine>
```

In this example, the two linac sector children each represent the same physical set of components. This is fundamentally different from a sector. For example, consider the sector defined by

```
<sector name = "this_sect">
  <element ref = "a_ele" />
  <element ref = "b_ele" />
  <element ref = "a_ele" />
</sector>
```

Here the two a_ele elements represent physically different elements. They just happen to have the same name.

Thus a <machine> with <sector> children can, for example, be used to describe an energy recovery Linac, where the beam goes through the some elements multiple times. This distinction is important when, for example, machine errors are to be simulated. In the above example for the "this_sect" sector, the a_ele elements of the sector can, in general, have different errors. However, the errors in the elements of the two linac machines in the "ERL" machine are, by definition, the same.

5.2.2 <lattice>

The <lattice> node defines various parameters of the lattice.

A global <lattice> node can be a child of <laboratory> in which case it is common to all machines (§5.2) or can be a child of a particular <machine> node. Only one global <lattice> is allowed and each <machine> is also allowed at most one <lattice> node.

Possible attributes of <lattice> are

```
name = "Name"
inherit = "Name".
```

The name attribute allows reference to lattice parameters (§2.6.1), while the inherit attribute allows a given <lattice> node to duplicate the properties of another named <lattice> node.

parameters set by a <lattice> node that is a child of a <machine> overrides any parameters of a global <lattice> node. For example

```
<lattice name = "global_lattice">
  <twiss>
    <beta_a design = "23.7" />
  </twiss>
</lattice>
<machine name = "linac">
```

```
    <lattice name = "linac_lattice">
      <twiss>
        <beta_a design = "34" />
      </twiss>
    </lattice
  </machine>
```

For the `linac` machine the `beta_a` twiss parameter will be 34.

The child nodes of `<lattice>` are:

`<label tag = "string">`		Label string
`<floor>`	§3.4.4	Global floor position.
`<geometry>`	§5.2.2	Linear or circular.
`<twiss>`	§4.2	Twiss parameters.
`<total_energy design|err = "expr">`		Beam total energy.
`<pc design|err = "expr">`		Particle momentum.
`<gamma design|err = "expr">`		Relativistic factor.

<geometry>

The `<geometry>` node is a child of `<lattice>` which defines whether the machine closes upon itself (`circular`) or does not (`linear`). The attributes of `<geometry>` are:

```
  <type = "Type" />     May be "LINEAR" or "CIRCULAR".
```

Example:

```
  <geometry type = "CIRCULAR" />
```

5.2.3 Branches

A `<branch>` node indicates a point at which a beamline in one `<machine>` connects to a beamline in another `<machine>`. The `<branch>` nodes must be included during the definition of `<sector>` nodes, but they come into use during instantiation (*i.e.*, in the `<machine>` nodes which are derived from the `<sector>` nodes).

The only attribute of a `<branch>` node is `name`. A branch which is a potential exit point for the beam will have a child node which indicates the destination for beams which exit, that node being `<to>`. The `<to>` node, in turn, can take the following attributes:

```
ref = "name" name of connecting point
reverse = "true|false" reverse tracking in destination beamline
(§5.2.5)
```

Additionally, the `<to>` node can take any number of `<set>` nodes as children. The `<set>` nodes indicate element parameters which are to be implemented when the `<branch>` in question is selected via use of a `<path>` node. Note that if a `<branch>` node marks a potential entry point for the beam but not a potential exit point, the `<branch>` will not have a `<to>` child.

For an example, consider a straight beamline which has a split between continued "straight-ahead" travel and an extraction line with pulsed kickers which send the beam into the extraction line. Such a system might look like this:

```
<machine name = "all">

   <machine name = "straight">
```

```
      <sector>
         <element name = "e1">
         ...
         <element name = "pkick">
            <kicker>
               <x_kick design = "0" />
            </kicker>
         </element>
         ...
         <branch name = "pulsedext">
            <to ref = "extline">
               <set "pkick[kicker:xkick@design] value = "0.09" />
            </to>
         </branch>
         <element name = "next_elem">
         ...
      </sector>
   </machine>

   <machine name = "ext">
      <sector>
         <branch name = "extline" />
         <element name = "firstextelem">
         ...
      </sector>
   </machine>

</machine>
```

In the example above, the <machine> straight is connected to the machine ext at the points defined by the two <branch> nodes. The <branch> in straight is an exit point, so it has a <to> child indicating the destination; the <branch> in ext is only an entry point, so it has no <to> child. The <set> node in the first <branch> node indicates that, when the beam is transferred from the straight machine to the ext machine, the design x kick strength of the pkick element is to be set to 0.09 radians, but that the kick is to be zero otherwise. Note that the <branch> node in straight is not the last element in straight; the <branch> nodes allow the connection between two <machine> nodes to occur at elements other than the first and last elements of those nodes.

Note that the <branch> names must be unique in the top-level <machine> node.

5.2.4 Paths

A <machine> node which represents a large and complex (*i.e.*, realistic) accelerator can have a very large number of interconnections represented by <branch> nodes. It is probable that some combination of possible branchings will represent illegitimate or impossible beam trajectories: in the example above, it will never be possible to go from the ext machine to the straight machine, even though there is a set of branches which connect them; similarly, even though the two storage rings of a B-factory are connected, it is never possible to send the beam from the low energy ring across the IR and into the high energy ring.

The combinations of branchings which are legitimate are determined by <path> nodes. A <machine>

may have as many <path> nodes as are required to define all the legitimate trajectories through the <machine>.

The only attribute of <path> is name. The possible children of <path> are:

```
<beam>
<branch_at>
<start>
```

The <beam> child (§4.1) defines the beam which is propagated down the path. The <start> child gives a reference to the element at which the path begins; its only attribute, reverse = "true|false", indicates whether the beam travels forwards or backwards starting from the referenced element (see §5.2.5 for more information). The <branch_at> nodes give reference to any <branch> nodes which are used in the path. The <branch_at> children must appear in the <path> node in the order in which they appear in the beam trajectory.

Continuing the example with the "straight" and "ext" beamlines, the appropriate <path> nodes would be the following:

```
<path name = "straight_ahead">
   <start ref = "e1" />
</path>
<path name = "to_dump">
   <start ref = "e1" />
   <branch_at ref = "pulsedext" />
</path>
```

The first path, named straight_ahead, starts at element e1 and continues straight to the last element in the machine straight, since there are no <branch_at> children in that path. The second path, to_dump, starts at the same element but, at the pulsedex branch point in the machine straight, jumps via that branch to the ext machine. Note that the machine may actually contain many other <branch> nodes, but only the <branch> nodes which are present as children of a <path> will be used when that <path> is active; all others that are encountered will be ignored.

5.2.5 Directionality and Polarity of Beamlines

Accelerator complexes often have beamlines which transport multiple beam species with different charges in different directions. The obvious example is an electron-positron collider, in which oppositely-charged particles counter-rotate in a common storage ring. More complicated examples include the Fermilab accelerator complex, in which even the injection lines for antiprotons are used as extraction lines for protons; and the SLAC linear accelerator, which transports electrons and positrons in the same direction.

The direction of travel in a machine is indicated by the reverse attribute of the <to> child of a branch, and also by the reverse attribute of the <start> child of a <path>. If reverse is set to true in a <to> node, it indicates that the beam is traveling through the destination beamline in the opposite direction from the order in which the elements were originally defined; the default value for reverse is false. Similarly, if reverse is true for a <start> node, then it indicates that tracking should begin at the element indicated by the start node but proceed backwards through the beamline rather than forwards. The default value of reverse is false for <start> nodes as it is for <to> nodes.

5.3 Lattice Expansion

Lattice expansion is the process by which the fully-instantiated, ordered list of elements which form the lattice is constructed. Ordinarily, the process of lattice expansion is the sort of technical detail which is

not important to a user who is constructing a lattice in *AML*. However, there are a few issues in this process which the user should be aware of.

Only one lattice is expanded, and it is the <machine> lattice which is defined last; expansion occurs after all parsing has been completed. This is different from typical accelerator parsing schemes, such as XSIF, which expand a lattice as soon as the expansion command (*i.e.*, `USE`) is encountered. If a <machine> is defined which has <machine> children, the lattice which is expanded is the top-level machine, including all of its machine children.

Because the lattice is not expanded until after all parsing is completed, `<set>` nodes are implemented before expansion occurs, even if the `<set>` node appears after the last `<machine>` node in the input files. In other words, the sequence

```
<set attribute = "QF[length@design]" value = "0.5" />
<machine name = "sample">
...
</machine>
<set attribute = "QF[length@design]" value = "1.0" />
```

will result in the `"sample"` machine being expanded with the length of the QF element set to 1.0 meters, even though the `<set>` node which establishes this value follows the `<machine>` definition in the input file.

During lattice expansion, the name prefixes (§5.1.5) are applied to the elements and sectors in the expanded lattice. Thus, an element name including prefixes cannot be referred to via the `<set>` node, and can only be referred to via the `<post_set>` node. The `<post_set>` nodes are applied after expansion, as implied by the name `"post_set"`.

Chapter 6

Controllers and Girders

6.1 <controller>

A <controller> is something that controls the attributes of machine elements. A <controller> may be used to simulate the effect of a klystron, power supply, control room knob, etc. In order to be useful with a particular program, that program has to programmed to recognize and use controllers.

The attributes of <controller> are:

```
name         = "Name"        "Name" is used to reference the controller.
variation    = "How"         "How" is either "ABSOLUTE" or "DELTA".
                                Default is "ABSOLUTE".
attribute    = "Attribute"   Default attribute of the slaves.
expression   = "Expression"  Default arithmetic expression to evaluate.
design       = "Value"       A design value.
err          = "Value"       An error value.
```

The children of <controller> are:

```
<slave>
<description>                §3.4.2
```

The attributes of a <slave> child are:

```
target = "Element[attribute]"   Element & attribute to be controlled.
                                  The default_attribute (see above) is used as
                                  the default if the attribute is not present.
expression = "Expression"       Arithmetic expression to evaluate.
```

target specifies both the element and the element's attribute that is being controlled. If only the element's name is specified, then the attribute attribute of the <controller> is used. For example:

```
<controller name = "ps1" variation = "ABSOLUTE"
              attribute = "quadrupole:k" design = "2" err = "0.1">
  <slave target = "q1" expression = "2.3 * sin(ps1[@actual])" />
  <slave target = "sol[solenoid:ks]" expression = "-5.7 * ps1[@actual]" />
</controller>
```

In this example, the controller controls two element attributes. The <variation> is set to "ABSOLUTE" which means that since the controller actual value is set to 2.1 (the sum of the design and err values), the <quadrupole:k> attribute of <q1> will be 1.9854 (= 2.3 * sin(2.1)) and the <solenoid:ks> attribute

of <sol> will be -11.97 (= 2.1 * -5.7). If multiple <controller>s control a particular attribute of a particular element, then the value of that attribute is obtained by summing the values over all the controllers. Notice that the actual component of <ps1> is used in the expressions of the slave elements.

If <variation> is set to "DELTA" then changes to the <controller> design or err attributes within a running program produce changes in the controlled attributes. Notice that it does not make sense for a particular attribute of a particular element to simultaneously be controlled by both a <controller> with a "DELTA" variation and another controller with a "ABSOLUTE" variation.

Controllers may control other controllers. For example, the ps1 controller above may be controlled via

```
<controller name = "ps1_control">
  <slave target = "ps1[]" />
</controller>
```

Notice that it is perfectly fine for a "DELTA" type <controller> to control a "ABSOLUTE" type <controller> or vice versa.

6.2 <girder>

A <girder> node defines a support structure that orients elements that are attached to it in space. If present, a <girder> node must be the first child of a <sector> node. The girder then supports all the elements contained within that sector.

When an <girder> supports an element, that elements orientation attributes (<x_offset>, <y_pitch>, <tilt>, etc.) give the orientation of the element with respect to the <girder>. An example will make this clear:

```
<element name = "q1">
  <length design = "10" />
  <quadrupole>
  </quadrupole>
</element>

<element name = "q2">
  <length design = "5" />
  <quadrupole>
  </quadrupole>
  <orientation>
    <x_pitch design = "0.01" />
    <x_offset design = "0.2" />
  </orientation>
</element>

<sector name = "supported1">
  <girder>
    <orientation>
      <x_pitch design = "0.1" />
      <x_offset design =  "0.3" />
    </orientation>
  </girder>
  <element ref = "q1" />
```

```
    <element ref = "q2" />
  </sector>
```

In this example `<girder>` element supports elements `q1` and `q2`. The girder starts at the beginning of `q1` which will be taken to be at $s = 0$ and ends at $s = 15$ which is the end of `q2`). Like other elements, `orientation:pitch` is calculated from the center of a `Girder` element (unless specified otherwise). The center of `q2` is at $s = 12.5$ so the distance between the center of `ib` and `q2` is $ds = 5$. The pitch of `ib` produces an offset at the center of `q2` of $0.5 = 0.1 * 5$. This, added to the offsets of `ib` and `q2`, give the total offset of `q2` to be $1.0 = 0.5 + 0.3 + 0.2$. The total `x_pitch` of `q2` is $0.11 = 0.1 + 0.01$. Note that for simplicity in this example, the small angle approximation has been used in the calculations.

Girders may also be nested:

```
  <sector name = "super_sect">
    <girder> ... </girder>
    <sector ref = "sub_sect1">
      <girder> ... </girder>
    </sector>
  </sector>
```

Part II

Physics

Chapter 7

Coordinates

7.1 Reference Orbit

The `reference orbit` is the curved path used to define a coordinate system for particles as shown in Figure 7.1. At a given time t, a particle's position can be described by a point on the reference orbit a distance s relative to the reference orbit's zero position plus a transverse offset. This point on the

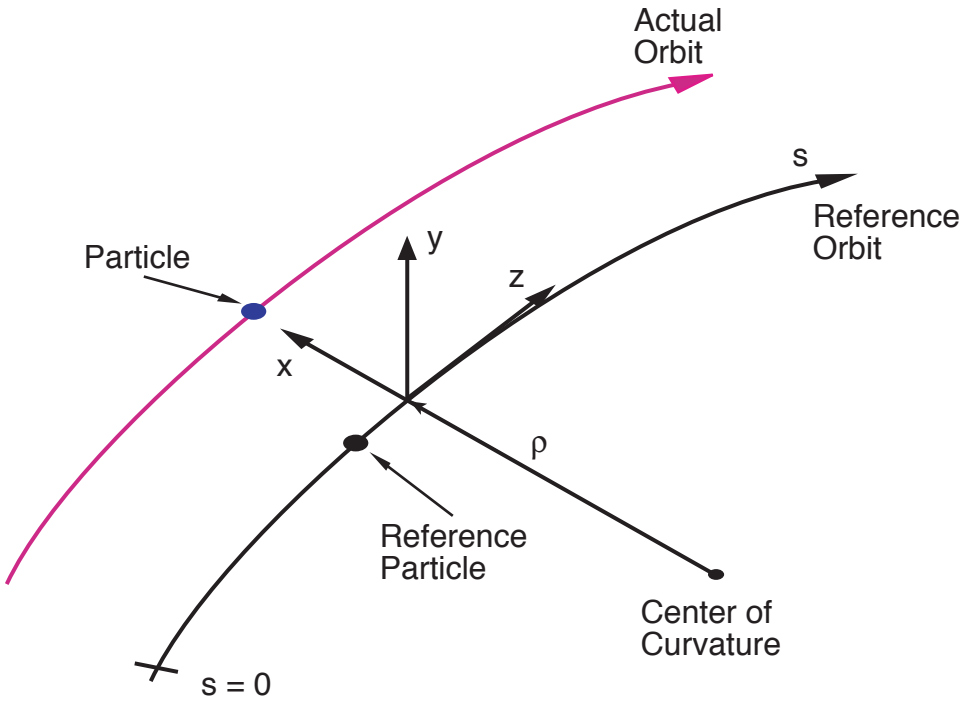

Figure 7.1: The Local Reference System. By construction, $z = 0$ for the particle coordinates in the local reference system.

reference orbit is used as the origin of the local (x, y, z) coordinate system with the z–axis tangent to the reference orbit and pointing in the direction of increasing s. The x and y–axes are perpendicular to the reference orbit. If the lattice has no vertical bends, the y–axis is in the vertical direction and the x–axis is in the horizontal plane. Notice that by construction, the particle is always at $z = 0$.

In *AML*, a lattice is comprised of a sequence of elements such as quadrupoles, bends, RF cavities, etc. Each element has an entrance point, an exit point, and a reference curve between them. For a bend, the reference curve is a segment of a circular arc. For all other elements, the reference curve is a straight line segment. The reference orbit itself is constructed by arranging the elements so that the exit point of one element coincides with the entrance point of the next with the reference curves forming an arc with no kinks. The reference orbit is then the sum of the reference curves. Exceptions to this construction method may be made by using `Patch` elements which can arbitrarily offset the entrance point of an element with respect to the exit point of the previous element. See §3.3.13. If not specified otherwise, the $s = 0$ position is the entrance point of the first element.

Notice that, in a `Wiggler`, the reference orbit, which is a straight line, does *not* correspond to the orbit that any actual particle could travel. Typically the physical entity of an element is centered about the reference curve. However, by specifying offsets and pitches, the physical element may be arbitrarily offset with respect to its reference curve. Shifting a physical magnet with respect to its reference curve generally means that the reference curve does *not* correspond to the orbit that any actual particle could travel.

7.2 Global Reference System

The global reference system describes the orientation of the reference orbit with respect to the laboratory coordinate system. *AML*, following the *MAD* convention, uses a Cartesian coordinate system (X, Y, Z) for the global reference system, along with three angles θ, ϕ, ψ used to define the reference orbit's orientation as shown in Figure 7.2. Conventionally, Y is the vertical coordinate and (X, Z) are the "floor" coordinates. The three angles are defined as follows:

θ **Azimuth angle:** Angle in the (X, Z) plane between the Z–axis and the projection of the z–axis onto the (X, Z) plane. A positive angle of $\theta = \pi/2$ corresponds to the projected z–axis pointing in the positive X direction.

ϕ **Pitch (elevation) angle:** Angle between the z–axis and the (X, Z) plane. A positive angle of $\phi = \pi/2$ corresponds to the z–axis pointing in the positive Y direction.

ψ **Roll angle:** Angle of the x–axis with respect to the line formed by the intersection of the (X, Z) plane with the (x, y) plane. A positive ψ forms a right–handed screw with the z–axis.

By default, at $s = 0$, the reference orbit's origin coincides with the (X, Y, Z) origin and the x, y, and z axes correspond to the X, Y, and Z axes respectively. θ decreases as one follows the reference orbit when going through a horizontal bend with a positive bending angle. This corresponds to x pointing radially outward. Without any vertical bends, the Y and y axes will coincide, and ϕ and ψ will both be zero. The `<initial>` statement (§3.4.4) in a lattice file can be use to override this default.

7.3 Beam Coordinate System

AML uses a canonical coordinate system to describe the position of beam particles with respect to the reference particle. In this coordinate system, the transverse coordinates x and y are defined in the

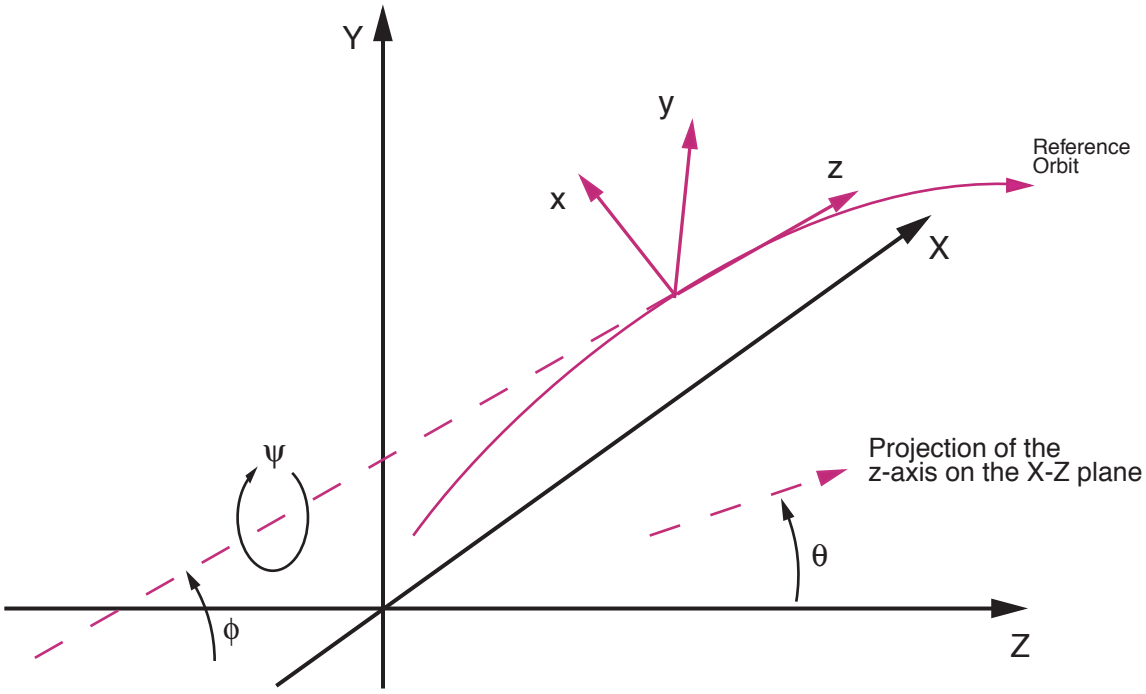

Figure 7.2: The Global Reference System

universally-accepted way. The transverse momentum / angle coordinates, p_x and p_y, are defined as follows:

$$p_{x,y} \equiv P_{x,y}/P_0, \tag{7.1}$$

where $P_{x,y}$ is the transverse component of a particle's momentum and P_0 is the reference value of the total momentum at the point in the accelerator where the particle is located.

The longitudinal coordinates are defined as follows:

$$\begin{aligned} z &\equiv -c\beta t \equiv -vt, \\ \delta p &\equiv \frac{\Delta P}{P_0}, \end{aligned} \tag{7.2}$$

where t is the arrival time of the particle with respect to the reference particle (where $t < 0$ indicates early arrival, therefore the head of the bunch), $\beta \equiv v/c$ is the rapidity of the particle, P_0 is the design momentum at the s location of the particle, and ΔP is the momentum error of the particle with respect to the reference momentum. Note that since $t < 0$ represents the head of the bunch, $z > 0$ also represents the head (*i.e.*, there is a sign change between the arrival time and the fifth coordinate used in *AML*).

There are subtle differences between the coordinate system described above and the coordinate systems used by some accelerator applications. In particular, the definition of the fifth and sixth coordinates is often somewhat different in the non-relativistic limit, and many codes normalize the transverse momentum to the total momentum of the particle or to the total longitudinal momentum rather than the total reference momentum. There are also differences with respect to the coordinate system used in MAD-8: that coordinate system is also canonical, but its fifth coordinate is $-ct$ and its sixth coordinate is $\Delta E/(P_0 c)$.

Chapter 8

Physics

Besides defining the syntax for describing an accelerator, *AML* must specify enough of the physics so that the meaning of various quantities are well defined. For example, *AML* must define what the `k` values of the various magnet families actually mean.

8.1 Magnetic Fields

8.1.1 Bends, Kickers, Quads, Sextupoles, and Octupoles

In a region where the local coordinate system is straight and not curved, and in which all of the magnets have midplane symmetry such that $B_x(y=0) \to 0$, the transverse kicks experienced by a particle passing through a magnetostatic field can be written as follows:

$$\begin{aligned} \Delta p_x &= L\left[-K_0 - K_1 x + \frac{1}{2}K_2(x^2-y^2) + \frac{1}{6}K_3(3xy^2-x^3) + ...\right], \\ \Delta p_y &= L\left[K_1 y + K_2 xy + \frac{1}{6}K_3(3x^2y-y^3) + ...\right]. \end{aligned} \tag{8.1}$$

Equation 8.1 defines the multipole expansion which is used by *AML* to define bends, kickers, quadrupoles, sextupoles, and multipoles. The bend child node `g` is equal to K_0, while the quadrupole, sextupole, and octupole `k` nodes are equal to K_1, K_2, and K_3, respectively. For a kicker, the `x_kick` value is equal to $-K_0 L$. Note that a positive `g` bends the beam in the $-x$ direction (ie, to the right), while a positive `x_kick` bends the beam in the $+x$ direction (ie, to the left).

For a particle with charge q and design momentum P_0, the relationship between the field coefficients K_n and the magnetic field B is:

$$K_n = \frac{q}{P_0}\frac{d^n B_y}{dx^n}. \tag{8.2}$$

AML also permits the use of magnetic expansion coefficients which are not normalized to the design momentum. These coefficients are indicated with a `_u` in the name: `g_u`, `k_u`, `x_kick_u`. The non-normalized components K_{nu} are related to the normalized components K_n by

$$K_{nu} = K_n \frac{P_0}{e} = N\frac{d^n B_y}{dx^n}, \tag{8.3}$$

where e is the fundamental charge in Coulombs and N is the number of fundamental charges per particle (1 for protons, muons, and positrons; -1 for antiprotons, antimuons, and electrons; larger values for ions).

Note that, as defined above, the K values in the multipole expansion always have the same effect on the beam dynamics regardless of the charge of the particle under consideration. The actual magnetic fields, however, will have a sign which depends on both the sign of the field coefficient K and the sign of the particle charge q. *AML* defines bends, kickers, quads, sextupoles, and octupoles in terms of either normalized or non-normalized K values, so the relevant strength parameters in *AML* are signed based only on their intended beam dynamics effects, not based on the design particle of the given beamline (*i.e.*, quads in an electron beamline use $K_1 > 0$ for horizontally focusing, as to quads in a positron beamline). The interesting case of beamlines which must transport particles with different charge signs is discussed in §5.2.5.

As noted, the expansion shown above assumes midplane symmetry, leading to magnets which are considered "normal" or "right" magnets. A "skew" magnet can be made from a normal magnet by rotating the normal magnet. The general form for the integrated kick is now written as

$$-\Delta p_x + i\Delta p_y = \frac{L}{n!}\left(K_n + i\,S_n\right)(x+iy)^n \quad . \tag{8.4}$$

This defines the skew component S_n. A pure skew magnet has $K_n = 0$. A magnet with positive S_n and zero K_n can be obtained by taking a normal magnet with positive k_n (and zero S_n) and rotating by an angle of $-\pi/[2(n+1)]$ (see Fig. (3.4)). Thus a skew quad can be obtained by rotating a normal quad by $-\pi/4$, etc. Alternatively, a rotation of a normal magnet with positive K_n by $\pi/[2(n+1)]$ will produce a pure skew magnet with negative S_n.

In *AML* a magnetic field component can generally be specified using the normal (K_n) and skew (S_n) components as well as specifying a tilt (ϕ_n). three parameters are not independent. If ϕ_n is taken to be zero, the relationship between the field components and the integrated kick is given in Eq. (8.4). If the skew component is taken to be zero. the integrated kick is given by

$$-\Delta p_x + i\Delta p_y = \frac{L}{n!}\,\widetilde{K}_n\, e^{-i(n+1)\phi_n}\,(x+iy)^n \quad . \tag{8.5}$$

The tilde is used here to differentiate K_n in this equation from the K_n in Eq. (8.4). The relationship between the parameters in the two equations is

$$\begin{aligned} K_n + i\,S_n &= \widetilde{K}_n e^{-i(n+1)\phi_n}, \\ |\widetilde{K}_n| &= \sqrt{K_n^2 + S_n^2}, \\ \tan((n+1)\,\phi_n) &= \frac{-S_n}{K_n} \quad . \end{aligned} \tag{8.6}$$

8.1.2 Scaled Multipoles

The conversion from the $K_{n,coef}$ and $S_{n,coef}$ multipole coefficients associated with a physical element via a `<scaled_multipole>` node to the actual multipole strengths is given by:

$$\begin{aligned} K_n &= K_{n,coef} \cdot F \cdot \frac{r_0^{n_{\rm ref}}}{r_0^n} \\ S_n &= S_{n,coef} \cdot F \cdot \frac{r_0^{n_{\rm ref}}}{r_0^n} \end{aligned} \tag{8.7}$$

where r_0 is set by the `radius` attribute of an element, and F and $n_{\rm ref}$ are set automatically depending upon the type of element as shown in Table 8.1.

Element	F	$n_{\rm ref}$
<kicker>	$\sqrt{\texttt{x\_kick}^2 + \texttt{y\_kick}^2}$	0
<electric_kicker>	$\sqrt{\texttt{x\_kick}^2 + \texttt{y\_kick}^2}$	0
<bend>	G	0
<quadrupole>	$\widetilde{K}$	1
<solenoid>	KSOL	1
<sol_quad>	$\widetilde{K}$	1
<sextupole>	$\widetilde{K}$	2
<octupole>	$\widetilde{K}$	3

Table 8.1: F and $n_{\rm ref}$ for various elements.

8.1.3 Solenoidal Fields

A pure solenoidal field is described by its coefficient in the equations of motion, k_s, and the longitudinal derivative of that coefficient, $k_s' \equiv dk_s/ds$. The deflection experienced by a beam passing through a solenoid is given by:

$$\begin{aligned} \Delta p_x &= L\left[-\frac{1}{2}k_s' y + (k_s + s k_s')\, p_y\right], \\ \Delta p_y &= L\left[\frac{1}{2}k_s' x - (k_s + s k_s')\, p_x\right]. \end{aligned} \tag{8.8}$$

From these equations, the relationship between the solenoid coefficient k_s and the magnetic fields in the solenoid can be derived:

$$\begin{aligned} \frac{q\,B_x}{P_0} &= -\frac{x}{2}k_s', \\ \frac{q\,B_y}{P_0} &= -\frac{y}{2}k_s', \\ \frac{q\,B_s}{P_0} &= (k_s + s k_s'). \end{aligned} \tag{8.9}$$

In these expressions, the origin of coordinate s is assumed to be at the center of the solenoid.

8.2 Taylor Maps

A transport map $\mathcal{M} : \mathcal{R}^6 \to \mathcal{R}^6$ through an element or a section of a lattice is a function that maps the starting phase space coordinates $\mathbf{r}(\text{in})$ to the ending coordinates $\mathbf{r}(\text{out})$

$$\mathbf{r}(\text{out}) = \mathcal{M}\,\mathbf{r}(\text{in}) \tag{8.10}$$

$\mathcal{M}$ is made up of six functions $\mathcal{M}_i : \mathcal{R}^6 \to \mathcal{R}$. Each of these functions maps to one of the $r(\text{out})$ coordinates. These functions can be expanded in a Taylor series and truncated at some order. Each Taylor series is in the form

$$r_i(\text{out}) = \sum_{j=1}^{N} C_{ij} \prod_{k=1}^{6} r_k^{e_{ijk}}(\text{in}) \tag{8.11}$$

Where the C_{ij} are coefficients and the e_{ijk} are integer exponents. The order of the map is

$$\text{order} = \max_{i,j}\left(\sum_{k=1}^{6} e_{ijk}\right) \tag{8.12}$$

8.3 Wigglers

As discussed in Section 3.3.20, `<wiggler>` elements are split into two classes: map type and periodic type. The map type `Wigglers` are modeled using the method of Sagan, Crittenden, and Rubin[6]. In this model the magnetic field is written as a sum of terms B_i

$$B(x,y,z) = \sum_i B_i(x,y,z;C,k_x,k_y,k_z,\phi_z) \tag{8.13}$$

Each term B_i is specified using five numbers: (C,k_x,k_y,k_z,ϕ_z). A term can take one of three forms: The first form is

$$\begin{aligned} B_x &= -C\frac{k_x}{k_y}\sin(k_x x)\sinh(k_y y)\cos(k_s s+\phi_s) \\ B_y &= \quad C\quad\cos(k_x x)\cosh(k_y y)\cos(k_s s+\phi_s) \\ B_s &= -C\frac{k_s}{k_y}\cos(k_x x)\sinh(k_y y)\sin(k_s s+\phi_s) \\ &\text{with } k_y^2 = k_x^2 + k_s^2\ . \end{aligned} \tag{8.14}$$

The second form is

$$\begin{aligned} B_x &= \quad C\frac{k_x}{k_y}\sinh(k_x x)\sinh(k_y y)\cos(k_s s+\phi_s) \\ B_y &= \quad C\quad\cosh(k_x x)\cosh(k_y y)\cos(k_s s+\phi_s) \\ B_s &= -C\frac{k_s}{k_y}\cosh(k_x x)\sinh(k_y y)\sin(k_s s+\phi_s) \\ &\text{with } k_y^2 = k_s^2 - k_x^2\ , \end{aligned} \tag{8.15}$$

The third form is

$$\begin{aligned} B_x &= \quad C\frac{k_x}{k_y}\sinh(k_x x)\sin(k_y y)\cos(k_s s+\phi_s) \\ B_y &= \quad C\quad\cosh(k_x x)\cos(k_y y)\cos(k_s s+\phi_s) \\ B_s &= -C\frac{k_s}{k_y}\cosh(k_x x)\sin(k_y y)\sin(k_s s+\phi_s) \\ &\text{with } k_y^2 = k_x^2 - k_s^2\ . \end{aligned} \tag{8.16}$$

The relationship between k_x, k_y, and k_z ensures that Maxwell's equations are satisfied. Since the field is given by analytic equations, Lie algebraic techniques can be use to construct Taylor maps to arbitrary order.

8.4 Wakefields

Wakefield effects are divided into short–range (within a bunch) and long–range (between bunches).

8.4.1 Short–Range Wakes

Short-range wakefields are represented in *AML* in a tabular format. The longitudinal monopole energy kick dE for the i^{th} macroparticle is computed from the equation

$$\delta(i) = \frac{-e\,L}{v\,P_0}\left(\frac{1}{2}W_\parallel^{SR}(0)\,|q_i| + \sum_{j\neq i} W_\parallel^{SR}(z_j - z_i)\,|q_j|\right) \tag{8.17}$$

where v is the particle velocity, e is the charge on an electron, q is the macroparticle charge, L is the cavity length, z_i and z_j are the longitudinal positions of the i^{th} and j^{th} macroparticles, respectively, and $W_{\parallel}^{SR}$ is the short–range longitudinal wakefield function. Note that $W_{\parallel}^{SR}(\Delta z < 0) \equiv 0$ for this definition of the wake function.$W_{\parallel}^{SR}$ has units of V/C/m.

The transverse kick $dp_x(i)$ for the i^{th} macroparticle due to the dipole short–range transverse wakefield is modeled with the equation

$$dp_x(i) = \frac{e\,L\,\sum_j |q_j|\,x_j\,W_{\perp}^{SR}(z_j - z_i)}{v\,P_0} \tag{8.18}$$

There is a similar equation for $dp_y(i)$. $W_{\perp}^{SR}$ is the transverse short–range wake function. Note that $W_{\perp}^{SR}(\Delta z \leq 0) \equiv 0$ for this definition of the wake function. $W_{\perp}^{SR}$ has units of V/C/m^2.

8.4.2 Long–Range Wakes

Following, Chao[9] Long–range wakefields are characterized by a set of cavity modes. The wake function W for a mode is given by

$$W(z) = \frac{c}{2}\left(\frac{R}{Q}\right)\,e^{kz/2Q}\,\sin(k\,z) \tag{8.19}$$

where the wake number k is related to the mode frequency ω by

$$k = \omega/c \tag{8.20}$$

Note that z is negative for trailing particles so $W(z)$ is a sinusoid with exponential decay as expected.

Assuming that the macroparticle generating the wake is offset a distance a along the x–axis, A trailing macroparticle will see a kick

$$d\mathbf{p}_{\perp}(i) = \frac{-e\,I_m\,W_m(z)\,m\,r^{m-1}\,\left(\hat{\mathbf{r}}\cos m\theta - \hat{\theta}\sin m\theta\right)}{v\,P_0} \tag{8.21}$$

$$\delta = \frac{-e\,I_m\,W'(z)\,r^m\,\cos m\theta}{v\,P_0} \tag{8.22}$$

where m is the order of the mode, r and θ represent the transverse position of the trailing macroparticle in polar coordinates, and the mode strength I_m is given by:

$$I_m = q\,a^m \tag{8.23}$$

The form of the transverse kick $d\mathbf{p}_{\perp}$ is the same as for a multipole of order m. Note that for $m = 1$, the deflection experienced by a trailing particle is not dependent on the transverse position of the trailing particle and is in the x direction. This represents a dipole mode wakefield.

For dipole and higher order modes ($m \geq 1$), each mode has two polarizations which are rotated with respect to one another by an angle $\pi/(2m)$. For a cavity with perfect symmetry, the two polarizations of a given mode would have the same frequency, Q, and R/Q. In reality, the two polarizations can have different values for these parameters. In addition, the axes of the modes can be rotated with respect to the nominal accelerator coordinate system: for example, the two polarizations of the dipole mode need not generate x deflections for x offsets and y deflections for y offsets, but can more generally generate deflections at an angle θ_m for offsets of the beam along the axis at angle θ_m, and deflections at an angle $\theta_m + \pi/2$ for offsets of the beam along the axis at angle $\theta_m + \pi/2$. Systems in which the two polarizations

of the dipole mode have different frequencies and are not aligned with the accelerator axes can couple x offsets of the leading macroparticle into y deflections of the trailing macroparticle.

In summary, each mode has two polarizations, which are each characterized by an R/Q, Q, ω, m, and θ_m; the angle between the two polarizations is $\pi/(2m)$. *AML* requires that the two polarizations for each mode be represented as separate `<mode>` children of the `<wake_modes>` node.

The modal R/Q has units of Ω/m^{2m}.

Notice that R/Q is defined so that it includes the cavity length. Thus the long–range wake equations, as opposed to the short–range ones, do not have any explicit dependence on L. Also notice that there is an overall sign difference between how the long–range and short–range transverse wakes are defined.

To make life more interesting different people define R/Q differently. A common practice is to define an R/Q "at the beam pipe radius". In this case the above equations must be modified to include factors of the beam pipe radius.

8.5 Coupling and Normal Modes

The coupling formalism used by *AML* is taken from the paper of Sagan and Rubin[7]. The main equations are reproduced here. A one–turn map $\mathbf{T}(s)$ for the transverse two–dimensional phase space $\mathbf{x} = (x, x', y, y')$ starting and ending at some point s can be written as

$$\mathbf{T} = \mathbf{V}\,\mathbf{U}\,\mathbf{V}^{-1}, \tag{8.24}$$

where $\mathbf{V}$ is symplectic, and $\mathbf{U}$ is of the form

$$\mathbf{U} = \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{pmatrix}. \tag{8.25}$$

Since $\mathbf{U}$ is uncoupled the standard Twiss analysis can be performed on the matrices $\mathbf{A}$ and $\mathbf{B}$. The normal modes are labeled a and b and if the one–turn matrix $\mathbf{T}$ is uncoupled then a corresponds to the horizontal mode and b corresponds to the vertical mode.

$\mathbf{V}$ is written in the form

$$\mathbf{V} = \begin{pmatrix} \gamma\mathbf{I} & \mathbf{C} \\ -\mathbf{C}^{+} & \gamma\mathbf{I} \end{pmatrix}, \tag{8.26}$$

where the symplectic conjugate is

$$\mathbf{C}^{+} = \begin{pmatrix} C_{22} & -C_{12} \\ -C_{21} & C_{11} \end{pmatrix}. \tag{8.27}$$

Since we demand that $\mathbf{V}$ be symplectic we have the condition

$$\gamma^2 + ||\mathbf{C}|| = 1, \tag{8.28}$$

and $\mathbf{V}^{-1}$ is given by

$$\mathbf{V}^{-1} = \begin{pmatrix} \gamma\mathbf{I} & -\mathbf{C} \\ \mathbf{C}^{+} & \gamma\mathbf{I} \end{pmatrix}. \tag{8.29}$$

$\mathbf{C}$ is a measure of the coupling. $\mathbf{T}$ is uncoupled if and only if $\mathbf{C} = \mathbf{0}$.

It is useful to normalize out the $\beta(s)$ variation in the the above analysis. Normalized quantities being denoted by a bar above them. The normalized normal mode matrix $\overline{\mathbf{U}}$ is defined by

$$\overline{\mathbf{U}} = \mathbf{G}\,\mathbf{U}\,\mathbf{G}^{-1}, \tag{8.30}$$

Where $\mathbf{G}$ is given by

$$\mathbf{G} \equiv \begin{pmatrix} \mathbf{G}_a & \mathbf{0} \\ \mathbf{0} & \mathbf{G}_b \end{pmatrix}, \tag{8.31}$$

with

$$\mathbf{G}_a = \begin{pmatrix} \frac{1}{\sqrt{\beta_a}} & 0 \\ \frac{\alpha_a}{\sqrt{\beta_a}} & \sqrt{\beta_a} \end{pmatrix}, \tag{8.32}$$

with a similar equation for $\mathbf{G}_b$. With this definition, the corresponding $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$ (cf. Eq. (8.25)) are just rotation matrices. The relationship between $\mathbf{T}$ and $\overline{\mathbf{U}}$ is

$$\mathbf{T} = \mathbf{G}^{-1}\,\overline{\mathbf{V}}\,\overline{\mathbf{U}}\,\overline{\mathbf{V}}^{-1}\,\mathbf{G}, \tag{8.33}$$

where

$$\overline{\mathbf{V}} = \mathbf{G}\,\mathbf{V}\,\mathbf{G}^{-1}. \tag{8.34}$$

Using Eq. (8.31), $\overline{\mathbf{V}}$ can be written in the form

$$\overline{\mathbf{V}} = \begin{pmatrix} \gamma\mathbf{I} & \overline{\mathbf{C}} \\ -\overline{\mathbf{C}}^{+} & \gamma\mathbf{I} \end{pmatrix}, \tag{8.35}$$

with the normalized matrix $\overline{\mathbf{C}}$ given by

$$\overline{\mathbf{C}} = \mathbf{G}_a\,\mathbf{C}\,\mathbf{G}_b^{-1}. \tag{8.36}$$

The normal mode coordinates $\mathbf{a} = (a, a', b, b')$ are related to the laboratory frame via

$$\mathbf{a} = \mathbf{V}^{-1}\,\mathbf{x}. \tag{8.37}$$

In particular the normal mode dispersion $\eta_a = (\eta_a, \eta'_a, \eta_b, \eta'_b)$ is related to the laboratory frame dispersion $\eta_x = (\eta_x, \eta'_x, \eta_y, \eta'_y)$ via

$$\eta_a = \mathbf{V}^{-1}\,\eta_x. \tag{8.38}$$

Bibliography

[1] Documentation on the GNU Lesser General Public License can be found at: http://www.gnu.org/licenses/lgpl.html.

[2] Information on XML can be found at: http://www.w3.org/XML.

[3] H. Grote, F. C. Iselin, *The MAD Program User's Reference Manual*, Version 8.19, CERN/SL/90-13 (AP) (REV. 5) (1996). Can be obtained at: http://mad.home.cern.ch/mad (under MAD version 8).

[4] F. C. Iselin, *The MAD program Physical Methods Manual*, unpublished, (1994). Can be obtained at: http://mad.home.cern.ch/mad (under MAD version 8).

[5] L. M. Healy, *Lie Algebraic Methods for Treating Lattice Parameter Errors in Particle Accelerators*. Doctoral thesis, University of Maryland, unpublished, (1986).

[6] D. Sagan, J. Crittenden, and D. Rubin. "A Symplectic Model for Wigglers," Part. Acc. Conf. (2003).

[7] D. Sagan and D. Rubin "Linear Analysis of Coupled Lattices," Phys. Rev. ST Accel. Beams **2**, 074001 (1999).

[8] R. Assmann, et. al., "LIAR — A Computer Program for the Modeling and Simulation of High Performance Linacs," Version 3 (2003). Web page at: http://www.slac.stanford.edu/accel/nlc/local/AccelPhysics/codes/liar/web/liar.htm.

[9] Alexander Chao, *Physics of Collective Beam Instabilities in High Energy Accelerators*, Wiley, New York (1993).

[10] P. Tenenbaum, "LIBXSIF, A Stand alone Library for Parsing the Standard Input Format," Proc. 2001 Part. Acc. Conf. p. 3093 — 95 (2001).

[11] R. Talman, "Multiparticle Phenomena and Landau Damping," in AIP Conf. Proc. **153** pp. 789–834, M. Month and M. Dienes editors, American Institute of Physics, New York (1987).

[12] E. Forest, *Beam Dynamics: A New Attitude and Framework*, Harwood Academic Publishers, Amsterdam (1998).