# Guide to LIBXSIF, a Library for Parsing the Extended Standard Input Format of Accelerator Beamlines

P. Tenenbaum
LCC-Note-0060
*14-Jun-2001*

**Abstract**

We describe LIBXSIF, a standalone library for parsing the Extended Standard Input Format of accelerator beamlines. Included in the description are: documentation of user commands; full description of permitted accelerator elements and their attributes; the construction of beamline lists; the mechanics of adding LIBXSIF to an existing program; and "under the hood" details for users who wish to modify the library or are merely morbidly curious.

## Contents

# 1 Version Information

This Note describes LIBXSIF version 1.2, Version Date 15-May-2001.

# 2 Introduction

The Standard Input Format (SIF) for accelerator beamline description was created in 1984 to respond to the need for a more user-friendly *lingua franca* for the accelerator world [1]. SIF had several notable improvements over previous languages (such as TRANSPORT format [2]):

- SIF permitted highly repetitive elements, from magnets to beamline symmetry units, to be defined once and reused repeatedly
- SIF permitted the beam optics to be defined independently of the beam energy, by declaring magnet strengths in quantities normalized to the energy (for example, defining the strength of a bend magnet by its total bend angle rather than its integrated magnetic field)
- SIF permitted many beamlines to be defined in a single file, and allowed the user to select which one was to be simulated
- SIF permitted the user to define named parameters, and allowed beamline element parameters to be defined in terms of arithmetic relationships between constants, named parameters, and parameters of other elements
- SIF permitted decks to be spread over multiple files, so that the actual beamline description and commands for the simulation program could be kept separate; this in turn permitted a single deck to be used without modification by many users, each of whom performed different simulations with different command files
- SIF permitted a more relaxed and intuitive syntax than the existing beamline description languages.

The standard input format was immediately adopted by the CERN simulation program MAD (Methodical Accelerator Design) [3], and was later added to the programs DIMAD [4], TRANSPORT [5], TURTLE [6], and COMFORT [7]. Nonetheless, the widespread adoption of SIF has been impeded by the absence of a single, standalone version of the deck parser that can easily be

added to any existing simulation program. In addition, the absence of linear accelerator elements in SIF has delayed its usage in programs in which such elements are critical, such as LIAR [8].

In this Note, we document a software solution to both of the problems above. LIBXSIF is a standalone library for the parsing of Extended Standard Input Format (XSIF) decks. Software engineers need add only a few calls to routines in this library to allow their programs to read and manage decks in the XSIF language. The standard is "extended" from SIF in the following ways:

- Linear accelerator elements, using the DIMAD keyword LCAVITY, are permitted
- Elements have an APERTURE attribute
- A handful of non-SIF elements, such as GKICKs, are included.

LIBXSIF is written in Fortran-90, and the latest version (v1.2) is available for Solaris, NT, and VMS-Alpha platforms; a Linux-i86 version will be released later this year. The present version of LIAR (v2.3) uses LIBXSIF, as does the present version of NLC-DIMAD (v2.8). The use of a common library to perform all XSIF parsing will greatly ease maintenance and expansion/extension of the code.

## 3 General Syntax

XSIF input is free format: blank spaces are ignored, all text (except for file names) is case-insensitive. By default, XSIF statements are confined to one 80-column line; continuation to the next line is indicated by a "&" symbol at or prior to column 80 of the current line, and all text after the continuation symbol is ignored. An exclamation point ("!") indicates a comment, and all text after the exclamation point is ignored:

```
APARAM := & all this text is ignored
!  so is all this text
1.5 + 2.5 !  complete the statement here
```

An entry in an XSIF file may be an *element*, a *beamline*, a *named parameter*, or a *command.* The syntax for these different entries is as follows:

- An element is specified by a name of up to 8 characters which is followed by a colon and an element keyword; the keyword must be at least 4 characters, and up to 8 characters of the keyword are significant. The keyword is followed by a comma, and a comma-deliniated list of element properties follows, for example:

  ```
  ELEMNAME : QUAD, L = 1.0, APERTURE = 0.01
  ```

- A beamline is specified by a name of up to 8 characters which is followed by a colon and the keyword LINE; this is followed by an equals-sign ("="), and a comma-delineated list of elements contained within parentheses. For example:

  ```
  FODO : LINE = (QF, D1, QD, QD, D1, QF)
  ```

- A named parameter is specified via one of three syntaxes: A name followed by a colon and the PARAMETER keyword, a name followed by a colon and the CONSTANT keyword, or a name followed by a colon and an equals sign and a value. Examples:

```
PAR1 :  PARAMETER = 1.0
PAR2 :  CONSTANT = 2.0
PAR3 := 3.0
```

- A command is a single word, sometimes followed by a comma-delineated argument list and in a few cases followed by an additional argument on the next line. For example:

```
USE, FODO
```

### 3.1 More on Names

Element, beamline, and parameter names may be from 1 to 8 characters in length. The first character must be a letter, and the remaining 7 characters may be a letter, a digit, a dollar sign ("$"), an underscore ("_"), or a decimal point ("."). Each element and beamline must have a unique name. Each named parameter must also have a unique name; however, since named parameters are stored separately from beamlines and elements, a named parameter may have the same name as a beamline or an element.

### 3.2 Named Parameters

A named parameter may be set equal to a numerical constant or to an arithmetic expression which includes numerical constants and other named parameters. In addition to the standard set of arithmetic operators (+,-,*,/), XSIF recognizes Fortran functions SQRT, LOG, EXP, SIN, COS, ATAN, ASIN, and ABS. Arithmetic expressions can include named parameters that have not yet been defined. XSIF also contains several named parameters that are initialized at startup and available to the user: PI, TWOPI, EMASS (in GeV), PMASS (in GeV), CLIGHT (in m/s), E (natural number), DEGRAD (radians to degrees), and RADDEG (degrees to radians). Element parameters can be used in expressions to define named parameters: in this case, the name of the element parameter is the name of the element followed by the name of the parameter enclosed in square brackets. For example, to define a named parameter NEWNAME which takes as its value the length of element OLDELT:

```
NEWNAME := OLDELT[L]
```

Once defined, a named parameter's value may be changed at will; this includes the predefined physical-constant named parameters listed above. All other named parameters which depend upon the changed named parameter will reflect this change. For example, if a parameter THREEPI is defined by the statement `THREEPI := 3*PI`, and the value of PI is changed by the statement `PI := 3.0`, then the value of THREEPI will be automatically changed to 9. Note that a parameter which is defined with the CONSTANT statement can still have its value changed; the CONSTANT keyword is provided only for compatibility with MAD.

### 3.3 Elements

Once an element is defined, its name can be used in place of the element keyword for future elements. For example, the statements:

```
QMASTER : QUAD, L=1, APERTURE = 0.01, K1=0.1
QSLAVE : QMASTER, K1 = -0.1
```

will cause the definition of a quadrupole named QSLAVE which has the same length and aperture as QMASTER but a different K1 value. In this case, redefinition of QMASTER is prohibited, but in general elements can be readily redefined just as named parameters can be.

Element parameters can be defined by arithmetic expressions; the syntax is exactly the same as for named parameters.

### 3.4 Beamlines

A beamline element list can contain either elements or other beamlines that have already been defined; use of an element or beamline that has not yet been defined is not permitted. Repetition of beamlines or reflection of beamlines is indicated by the * and - signs, respectively, for example:

```
FODO1A : LINE = (QF, D1, QD)
FODO1 :  LINE = (FODO1A, -FODO1A)
FODO10 :  LINE = (10*FODO1)
```

A beamline definition can also accept formal arguments, in which some of the elements in the beamline list are variables. For example:

```
FODO1A(QF,QD) : LINE = (QF, D1, QD)
FODO1 :  LINE = FODO1A(QFA,QDA)
FODO2 :  LINE = FODO1A(QFB,QDB)
```

defines two FODO half-cells which have the same drift element D1 but different F and D quads.

## 4 Commands in XSIF

XSIF recognizes the following commands: CALL, CLOSE, CONSTANT, ECHO, LINE, NLC, NOECHO, NONLC, OPEN, PARAMETER, PATH, RETURN, USE.

### 4.1 CALL

Switch input from present logical unit to another: `CALL, NN` causes logical unit NN to be used for input; NN is an integer from 1 to 99. A CALL is terminated by a RETURN statement (see 4.12). XSIF permits nested CALL/RETURN statements; up to 32 nested CALLs are permitted.

### 4.2 CLOSE

Closes a logical unit which was opened by a previous OPEN statement (see 4.9). `CLOSE, NN` causes whichever file is connected to logical unit NN to be closed.

### 4.3 CONSTANT

Declares a named parameter. `APAR : CONSTANT = 1.0` declares named parameter APAR with value 1.0. As described above, in XSIF the CONSTANT statement is synonymous with PARAMETER (see 4.10), and does not preclude changes to the value of the named parameter declared with a CONSTANT statement.

### 4.4 ECHO

Causes a copy of the input stream to be written to the error file. Antonym for NOECHO (section 4.7).

### 4.5 LINE

Declares a beamline definition: `ALINE : LINE = (...)`.

### 4.6 NLC

Causes a warning whenever an element defintion is found which does not correspond to the NLC deck standards [9]. Antonym for NONLC (section 4.8).

### 4.7 NOECHO

Disables copying of the input stream to the error file. Antonym for ECHO (section 4.4).

### 4.8 NONLC

Disables warning for elements which violate the NLC deck standards. Antonym for NLC (section 4.6).

### 4.9 OPEN

Opens a file to a selected logical unit number. Syntax:

```
OPEN, NN
filename
```

opens the named file to logical unit NN, which is between 1 and 99. The filename must be on the line after the OPEN statement. The filename can contain the $PATH variable (see PATH, section 4.11).

### 4.10 PARAMETER

Declares a named parameter: `APAR : PARAMETER = 1.0`. In general, it is more convenient to use the syntax `APAR := 1.0`.

### 4.11 PATH

Declares an alternate path for use in file operations. Syntax:

```
PATH
pathname
```

where the name of the path must be on the line following the PATH statement. In subsequent file operations, such as OPEN statements, the expression $PATH will be replaced by the specified path name. The $PATH expression can be used in the PATH command to generate extremely long absolute pathnames.

### 4.12 RETURN

Ends use of a CALL'ed file for input purposes; when a RETURN is encountered, XSIF switches to using the file that contained the CALL statement for further input. CALL/RETURN pairs can be nested to a depth of 32.

### 4.13 USE

The USE command selects a beamline to be expanded for simulation studies by the calling program. The syntax for the USE command was elucidated in the previous section. While the SYMM and SUPER arguments of the MAD USE command are recognized, they do not presently do anything!

## 5 Accelerator Elements

The list of known accelerator elements is a combination of the standard MAD elements and the standard DIMAD elements. Each element accepts two alphanumeric parameters: a 16-character string intended for use as an engineering classification, and a 24-character string intended for use as a database name. These parameters are indicated with the TYPE and LABEL keywords, respectively, and must be enclosed in single or double quotes (i.e., `QUAD, TYPE = ‘‘15Q2.85’’, LABEL = ‘‘QUAD:CB00,1010’’`). Neither TYPE nor LABEL strings must be unique. The present version of LIBXSIF requires that the first character in a TYPE or LABEL string be a letter, while the remainder may be any combination of letters, digits, periods (“.”), colons (“:”), underscores (“_”), and dollar signs (“$”). All other element parameters require a number or an expression that evaluates to a number, and all element parameters default to zero if not explicitly defined, with the following exceptions:

- If the TILT parameter is given without a numerical value, a rotation about the longitudinal axis from normal to skew orientation is assumed (i.e., 90 degrees for a bend, 45 degrees for a quad, 30 degrees for a sextupole, etc.)
- LFILE and TFILE parameters expect a file name enclosed within single or double quotes; up to 80 characters is permitted (including the terminating quotes); use of the $PATH expression is permitted (see previous section).

### 5.1 DRIFT

| | |
|---|---|
| L | is the length. |

### 5.2 SBEND

This is a sector bend magnet.

| | |
|---|---|
| L | is the path length through the magnet (not the magnet rectangular length). |
| ANGLE | is the bend angle. |
| K1 | is the quadrupole component of the field; see section 5.4. |
| E1 | is the entrance edge angle. |
| E2 | is the exit edge angle. |
| TILT | is the tilt angle. |
| K2 | is the sextupole component of the field; see section 5.5. |

| | |
|---|---|
| H1 | is the entrance pole face curvature. |
| H2 | is the exit pole face curvature. |
| HGAP | is the entrance half gap size. |
| FINT | is the entrance fringe field integral, which defaults to 0.5. |
| HGAPX | is the exit half gap size. If HGAPX is not given a value, it defaults to the value of HGAP. |
| FINTX | is the exit fringe field integral. If FINTX is not given a value, it defaults to the value of FINT. |

## 5.3 RBEND

The rbend is a parallel faced dipole magnet. Its parameters are the same as those of the sbend. Parameters E1 and E2 are not provided by the user and are set by the program to half the value of the bend angle.

## 5.4 QUADRUPOLE

The strength parameter is defined by K1 $\equiv B_{\rm pole}/(aB\rho)$, where $B_{\rm pole}$ is the pole-tip magnetic field and $a$ is the aperture radius.

| | |
|---|---|
| L | is the length. |
| K1 | is the strength. |
| TILT | is the tilt angle. |
| APERTURE | is the magnet aperture. |

## 5.5 SEXTUPOLE

The strength parameter is defined by K2 $\equiv 2B_{\rm pole}/(a^2B\rho)$, where $B_{\rm pole}$ is the pole-tip magnetic field and $a$ is the aperture radius.

| | |
|---|---|
| L | is the length. |
| K2 | is the strength. |
| TILT | is the tilt angle. |
| APERTURE | is the magnet aperture. |

## 5.6 QUADSEXT

Combined function quadrupole-sextupole, with strengths defined as in sections 5.4 and 5.5.

| | |
|---|---|
| L | is the length. |
| K1 | is the quadrupole strength. |
| K2 | is the sextupole strength. |
| TILT | is the tilt angle; if entered with no argument, an angle of 45 degrees is used. |
| APERTURE | is the magnet aperture. |

## 5.7 OCTUPOLE

The strength parameter is defined by K3 $\equiv 6B_{\rm pole}/(a^3 B\rho)$, where $B_{\rm pole}$ is the pole-tip magnetic field and $a$ is the aperture radius.

| | |
|---|---|
| L | is the length. |
| K3 | is the strength. |
| TILT | is the tilt angle. |
| APERTURE | is the magnet aperture. |

## 5.8 MULTIPOLE

This is an arbitrary high-order multipole.

| | |
|---|---|
| L or LRAD | is the length. |
| K0L - K20L | are the integrated strengths, with KnL $\equiv n!B_{\rm pole,n}L/(a^n B\rho)$, and $B_{\rm pole,n}$ is the contribution to the pole-tip field of the n'th multipole. |
| T0 - T20 | are the tilt angles. If tn is entered without a value, half-turn/2(n+1) is assumed. |
| KZL | is the integrated longitudinal strength, KZL = $B_z L/B\rho$. |
| KRL | is the integrated radial strength, KRL = $B_r L/B\rho$. |
| THETA | is the beam crossing angle with respect to the symmetry axis of the solenoidal component. |
| Z | is the distance in z to the point at which the beam and solenoidal axes meet. |
| SCALEFAC | is a dimensionless strength factor, used to scale all the strengths together. |
| TILT | is the overall tilt angle. |
| APERTURE | is the magnet aperture. |

NOTE : Parameters KZL, KRL, THETA, and Z are used to simulate a beam passing at an angle through a solenoidal field slice.

## 5.9 DIMULTIPOLE

This element is the "traditional" DIMAD multipole. It is used to preserve compatibility with existing DIMAD decks, since the keywords of the MULTIPOLE have been changed to allow compatibility with MAD.

| | |
|---|---|
| L | is the length. If the length is zero, the strengths are interpreted as integrated strengths. |
| K0 - K20 | are the integrated strengths. |
| T0 - T20 | are the tilt angles. If Tn is entered without a value, half-turn/2(n+1) is assumed. |
| SCALEFAC | is a dimensionless strength factor, used to scale all the strengths together. |
| TILT | is the overall tilt angle. |
| APERTURE | is the magnet aperture. |

## 5.10 SOLENOID

Note that the KS value is defined to be $B_S/B\rho$ in the MAD standard, which differs from the TRANSPORT definition by a factor of 2.

| | |
|---|---|
| L | is the length. |
| KS | is the solenoid strength. |
| K1 | is the quadrupole strength. |
| TILT | is the tilt angle |
| APERTURE | is the magnet aperture. |

## 5.11 RFCAVITY

This is a storage-ring RF cavity.

| | |
|---|---|
| L | is the length. |
| VOLT | is the cavity voltage, in MeV |
| LAG | is the phase lag of the cavity with respect to a nominal particle at the start of the machine, in radians/$2\pi$. |
| FREQ | is the frequency of the cavity in MHz. |
| HARMON | is the harmonic number of the cavity. |
| ENERGY | is the energy.(GeV). |
| ELOSS | This is the energy loss factor of the cavity, in V/coulomb. |
| LFILE | This is a filename of up to 78 characters enclosed in double quotes. The file contains longitudinal wakefield data for the cavity, in units of V/coulomb/meter. The filename may use the $PATH construct to refer to an alternate path. |
| TFILE | This is a filename of up to 78 characters enclosed in double quotes. The file contains transverse wakefield data for the cavity, in units of V/coulomb/m$^2$. The filename may use the $PATH construct to refer to an alternate path. |
| NBIN, BINMAX | Used by some simulation programs for histogramming particles according to their longitudinal coordinates for wakefield computations. |
| APERTURE | is the aperture. |

## 5.12 LCAVITY

This element is a cavity or structure for linear acceleration.

| | |
|---|---|
| L | is the length. |
| E0 | is the injection energy, typically supplied only for the first cavity. |
| DELTAE | is the energy gain on crest without beam loading in MeV. |
| PHI0 | is the phase offset for a reference particle in radians/$2\pi$. A positive phase indicates that the RF crest is ahead of the bunch. |
| FREQ | is the frequency in MHz. |
| ELOSS | is the cavity beam loading factor in V/Coulomb. |
| LFILE | is a 78-character filename, enclosed in double quotes. The file contains the cavity's longitudinal wakefield Green's Function in V/C/m. The filename may use the $PATH construct to refer to an alternate path (see Control Flow, below). |

| | |
|---|---|
| TFILE | is a 78-character filename, enclosed in double quotes. The file contains the cavity's transverse wakefield Green's Function in V/C/m$^2$. The filename may use the $PATH construct to refer to an alternate path. |
| NBIN, BINMAX | Used by some simulation programs for histogramming particles according to their longitudinal coordinates for wakefield computations. |
| APERTURE | is the aperture. |

### 5.13 ROLL, SROT

This element performs a rotation of the coordinate system about the longitudinal axis.

| | |
|---|---|
| ANGLE | is the rotation angle. A positive angle means the new coordinate system is rotated clockwise about the s- axis with respect to the old system. |

### 5.14 ZROT, YROT

This element performs a rotation of the coordinate system about the vertical axis. The angle must be small.

| | |
|---|---|
| ANGLE | is the rotation angle. A positive angle means the new coordinate system is rotated clockwise about the local z-axis with respect to the old system. |

### 5.15 HKICK, VKICK

These are horizontal and vertical steering magnets.

| | |
|---|---|
| KICK | a horizontal (vertical) kick of size kick |
| TILT | rotation angle about the longitudinal axis |

### 5.16 GKICK

This element is a general kick.

| | |
|---|---|
| L | is the length. |
| DX | is the change in x. |
| DXP | is the change in x'. |
| DY | is the change in y. |
| DYP | is the change in y'. |
| DL | is the change in path length. |
| DP | is the change in dp/p. |
| ANGLE | is the angle through which the coordinates are rotated about the longitudinal axis. |
| DZ | is the longitudinal displacement. |
| V | is the extrance-exit parameter of the kick, which signals to some programs whether the kick is applied at the beginning or end of the element and whether it is applied on all turns. |

| | |
|---|---|
| T | is the momentum dependence parameter. The kicks dx' and dy' can be thought of as misalignment errors or as angle kicks of orbit correctors. In the first case (T=0) they are momentum independent. When T=1 the kicks dx' and dy' vary inversely with momentum. |

## 5.17 HMONITOR, VMONITOR, MONITOR

These elements are horizontal, vertical, and horizontal and vertical monitors, respectively.

| | |
|---|---|
| L | is the monitor length. |
| XSERR,<br>YSERR,<br>XRERR,<br>YRERR | are the x and y systematic and random errors. |

## 5.18 BLMONITOR, PROFILE, WIRE, SLMONITOR, IMONITOR, INSTRUMENT

These are different types of instrumentation.

| | |
|---|---|
| L | is the length. |

## 5.19 MARKER

A marker is a drift element of zero length. It has no parameters.

## 5.20 ECOLLIMATOR, RCOLLIMATOR

An ecollimator is elliptic, and an rcollimator is rectangular. The particles are checked at the entrance and at the exit of the collimator.

| | |
|---|---|
| L | is the length. |
| XSIZE<br>YSIZE | are the x and y collimator apertures. The default apertures are 1 meter. |

## 5.21 ARBITELM

This is the arbitrary element.

| | |
|---|---|
| L | is the length. |
| P1 - P20 | are the parameters. |

## 5.22 MTWISS

This is a general transfer matrix expressed in terms of its matched Twiss parameters and phase advance.

| | |
|---|---|
| L | is the length. |
| MUX,<br>BETAX,<br>ALPHAX, | |

MUY,
BETAY,
ALPHAY — are the twiss parameters for this transfer matrix. betax and betay have default values of 1.

### 5.23 MATRIX

This element is a general transfer matrix.

R$ij$
T$ijk$ — are the matrix elements. $i$,$j$, and $k$ range from 1 to 6, but $j$ is always less than or equal to $k$.

## 6 How to Use LIBXSIF

The standalone XSIF library contains 76 object files at present. Fortunately, most of these object files contain subroutines which are only needed by other XSIF subroutines, and the user only needs to use a handful of them in the program which is to call XSIF. Once parsing is complete, it is necessary for the calling program to transfer the element definitions and their order in the USEd beamline to its own internal data structures; typically this can be accomplished in a single subroutine.

### 6.1 Fortran-90 Modules

The data used by XSIF is distributed amongst a number of Fortran-90 shared-data structures called Modules. These are integrated into the subroutines that require them by "USE Association:" the required modules are named in USE statements that appear prior to the IMPLICIT statement. The subroutine that calls the XSIF parsing routines needs to USE the following modules: XSIF_INOUT, XSIF_INTERFACES, XSIF_ELEMENTS. The subroutine that transfers data from the XSIF data structures to the calling-program data structures needs to USE the following modules: XSIF_SIZE_PARS, XSIF_ELEMENTS, XSIF_ELEM_PARS.

### 6.2 Initializing and Activating the Parser

The following subroutines and functions are relevant for the initialization and activation of the parser:

#### 6.2.1 XSIF_IO_SETUP

This is an INTEGER*4 function which takes 9 arguments:

- A CHARACTER*(*) which is the path and filename of the XSIF deck
- A CHARACTER*(*) which is the path and filename for the error stream
- A CHARACTER*(*) which is the path and filename of XSIF's standard output
- An INTEGER*4 which is the logical unit number to be used for the deck
- An INTEGER*4 which is the logical unit number to be used for the error stream
- An INTEGER*4 which is the logical unit number to be used for the standard output

- An INTEGER*4 which is the logical unit number to be used for any warnings or error messages from XSIF_IO_SETUP itself
- A LOGICAL*4 which indicates whether the deck is to be echoed to the error stream and standard output
- A LOGICAL*4 which indicates whether NLC coding standard warning messages are desired.

XSIF_IO_SETUP opens the three desired files to the desired unit numbers (the deck with status "OLD," the error and the standard output with status "REPLACE"); if successful, it executes subroutines RDINIT and CLEAR (see below), and sets the value of INTER (interactive) via function INTRAC (see below). If all is successful, XSIF_IO_SETUP will return 0; otherwise it returns error value XSIF_PARSE_NOOPEN (defined in module XSIF_INOUT), which indicates that one of the 3 files could not be opened.

### 6.2.2 RDINIT

A subroutine which initializes various I/O variables and writes the XSIF version information to the error stream, the standard output, and the screen.

### 6.2.3 CLEAR

A subroutine which initializes variables related to the element and named parameter data structures in XSIF.

### 6.2.4 INTRAC

At one time this function indicated whether the master program was running interactively; the present version always returns FALSE. This is only used by the parser if a syntax error has occurred; non-interactive mode permits it to scan the deck for additional syntax errors before aborting (see section 6.2.5).

### 6.2.5 XSIF_CMD_LOOP

This is an INTEGER*4 function. It executes the main loop that reads new lines of the XSIF file, determines what form of input the line represents (command, element, beamline definition, or named parameter) and responds accordingly. This function takes a single, optional argument: the name of a LOGICAL*4 function. XSIF_CMD_LOOP will continue to parse the input deck (and any other files accessed from that deck via the OPEN/CALL commands) until one of the following occurs:

- A function or subroutine in XSIF sets the value of global variable XSIF_STOP to TRUE, in which case all functions and subroutines in the call chain will complete execution normally and XSIF_CMD_LOOP will pass execution back to the calling routine
- A fatal read error occurs, in which case the global variable FATAL_READ_ERROR is set to TRUE and all subroutines and functions in the call chain will abort execution and pass control back to the routine that called XSIF_CMD_LOOP as quickly as possible
- All input data in all files accessed by the XSIF parser is exhausted (either by RETURN statements or by end-of-file), at which time control is returned to the calling routine.

If all parsing was completed without errors, XSIF_CMD_LOOP will return 0. Otherwise it will return the error values XSIF_PARSE_ERROR, indicating a syntax error in the deck, or XSIF_FATALREAD, indicating a fatal read error; both of these signals are defined in XSIF_INOUT. Note that if a parsing error is encountered, XSIF_CMD_LOOP will enter "scanning mode," in which it continues to parse the deck but nonetheless returns a bad status. This permits a single pass through XSIF to detect most (if not all) of the syntax errors, rather than detecting and aborting on one error per pass.

The optional function argument, referred to in XSIF_CMD_LOOP as XSIF_EXTRA_CMD, permits the XSIF parser to manage commands or elements which are not part of the library as described in this Note. If XSIF_EXTRA_CMD is present, XSIF_CMD_LOOP will call it before executing its standard command-handling routines. XSIF_EXTRA_CMD takes as arguments the name of the most recent command keyword (CHARACTER*8) and its length (INTEGER*4), the name of the most recent beamline element (CHARACTER*8) and its length (INTEGER*4), and an error flag (LOGICAL*4). If XSIF_EXTRA_CMD returns TRUE, then XSIF_CMD_LOOP assumes that the last command was handled by XSIF_EXTRA_CMD, it will bypass the built-in command handler and proceed to read the next statement. If XSIF_EXTRA_CMD sets its last argument to TRUE, then XSIF_CMD_LOOP knows that a syntax error has occurred within XSIF_EXTRA_CMD, and it will enter "scanning mode."

### 6.2.6 XSIF_IO_CLOSE

This subroutine closes the XSIF error file, the XSIF output stream file, and all deck files that were opened via CALL statements. At the present time, XSIF_IO_CLOSE does not close files that were opened via XSIF OPEN statements but never CALLed. XSIF_IO_CLOSE takes no arguments.

### 6.2.7 XUSE2

This is an INTEGER*4 function that takes the name of a beamline (CHARACTER*8) as its argument and attempts to expand it (executing XUSE2 with the name of a beamline is equivalent to placing that beamline's name in a USE statement in the XSIF deck). XUSE2 will return 0 if it succeeded in expanding the named beamline, and it returns XSIF_PARSE_ERROR if the the name in question does not correspond to a beamline. If any beamline is successfully expanded by XSIF at any time (by a USE statement in the deck or an XUSE2 call), then the LOGICAL*4 variable LINE_EXPANDED is set to TRUE. This is a useful way for the calling routine to tell if there is an expanded beamline ready for simulations or not.

## 6.3 Extracting the Beamline and Elements

If XSIF is to be added to an existing program, it is unlikely that the existing program will be capable of using the XSIF data structures directly to store information that it needs to generate the beamline, matrices, etc. Even if it proves possible, it may not be desirable to do it this way. Thus, most programs will require a routine that extracts the expanded beamline, its elements, and their parameters. Consequently, we include helpful tips on how this can be done. All of the data read by the XSIF parser is in structures in the XSIF_ELEMENTS module.

### 6.3.1 Evaluation of Parameters

In the XSIF data structures, the arithmetic relationships between the various defined parameters are preserved. Before extracting the beamline and elements, it is necessary to calculate the numerical

values of all parameters. This can be done by calling (in order) INTEGER*4 function PARCHK and subroutines PARORD and PAREVL.

PARCHK examines the parameters to make sure that all of the named parameters have been defined; since undefined parameters default to zero value, unexpected and unpleasant things can happen if an important parameter depends upon one that is accidentally undefined and therefore zero by default. PARCHK takes one argument, a LOGICAL*4; if this argument is TRUE, then undefined parameters are treated as errors, otherwise they are treated as warnings. If no undefined parameters are detected, PARCHK returns 0. If one or more undefined parameters are detected, appropriate messages are sent to the error file and PARCHK returns either a negative number (if undefined parameters are considered errors) or a positive number (if they are considered warnings). In either case the absolute value of the return is XSIF_PAR_NODEFINE, which is defined in module XSIF_INOUT.

PARORD examines the relationships between the parameters and determines the correct order for evaluating them; this information is stored in a table. PAREVL goes down the table generated by PARORD and evaluates the numerical value of all parameters. Both subroutines produce warnings if obvious problems, such as circular definitions or divide-by-zero, are encountered.

### 6.3.2 Storage of the Beamline and Element Data

The list of elements in beamlines is stored in INTEGER*4 array ITEM. The index of the first element of the expanded beamline is in global variable NPOS1, and the index of the last element is NPOS2-1. The element information is encoded in the following structures:

- The I'th entry in ITEM is the index number of an element in a beamline
- IETYP(ITEM(I)) is the type of element (quad, bend, etc), according to the parameters in XSIF_ELEM_PARS (ie, if IETYP(ITEM(I)) == 5, then ITEM(I) is a quad since parameter MAD_QUAD in XSIF_ELEM_PARS == 5)
- IEDAT(ITEM(I),1) is a pointer into the parameter list, PDATA, of the first element parameter of ITEM(I). Data for a given element is stored sequentially in PDATA, and the order of the parameters for each type of element is shown in XSIF_ELEM_PARS. Since for all elements except MARKERs and various kinds of rotations the length is the first parameter, for example, PDATA(IEDAT(ITEM(I),1)) is typically the length of element ITEM(I).
- The name, type, and label of an element are KELEM(ITEM(I)), KETYP(ITEM(I)), and KELABL(ITEM(I)), respectively.

Section 7 contains more information on the storage of information within XSIF's data structures.

## 6.4 Examples

We recognize that in many cases a working example is the best form of documentation. At present, there are two programs available that make use of LIBXSIF: LIAR and DIMAD version 2.8. The source codes for these programs are available for perusal at the NLC Accelerator Physics web site: `http://www-project.slac.stanford.edu/lc/local/AccelPhysics/codes/nlc_simulation_codes.htm`.

The LIAR call to LIBXSIF is in the context of a LIAR command, READ_XSIF:

```
read_xsif, file = ‘‘linac.xsif’’,
line = ‘‘ELIN1’’,
energy = 10,
echo = .f.
```

tells LIAR to parse a file named `linac.xsif,` to then expand for simulation a beamline named `ELIN1` with an initial energy of 10 GeV, and to suppress duplication of the input stream in the output and error streams. All of the interfacing to LIBXSIF is in the subroutine READ_XSIF and the function XSIF_EXPAND, both in the file read_xsif.f.

DIMAD is a much older program than LIAR, and has for over a decade had a very different "look and feel." In this case, DIMAD was designed to have its commands and its deck inputs in a single input stream: DIMAD's standard input parser would handle the command, DIMAT, that signalled the end of deck parsing and the beginning of simulation activity. This "look and feel" was preserved by use of an additional function which handles DIMAD's additions to the standard input; this function is passed to XSIF_CMD_LOOP as an argument, as described in section 6.2.5. The interface to LIBXSIF is in file madin_new.f; beamline expansion is handled by subroutine DIMATD, in dimatd.f. DIMAD eschews use of XSIF_IO_SETUP and XSIF_IO_CLOSE, since these routines are primarily intended for use in programs that keep their XSIF input sequence separate from their command streams (which DIMAD does not); consequently, DIMAD has calls to RDINIT, CLEAR, and INTRAC within MADIN_NEW.

# 7 Technical Information

The information in the preceding sections, coupled with examples from the implementation of XSIF calls in LIAR and DIMAD, should be sufficient (we hope!) to permit users to add LIBXSIF capabilities at will to their simulation programs. Such an approach relieves the users of LIBXSIF of the responsibility for reading and understanding each of the source code files in the present version of LIBXSIF. Nonetheless, some additional technical information on the inner workings of LIBXSIF is potentially desirable to those who wish to expand or modify said workings, or simply to have a better understanding of the system.

## 7.1 Global Data Storage – LIBXSIF Module Files

LIBXSIF stores its global variables in Modules, which must be accessed by USE statements in subroutines that seek access to the values.

### 7.1.1 Module XSIF_SIZE_PARS

Contains parameter definitions for statically-allocated arrays of various types in other XSIF modules. All parameters are of type INTEGER*4.

### 7.1.2 Module XSIF_ELEM_PARS

This contains parameter definitions required for parsing the various classes of elements. Most of the information is in the form of "dictionaries:" arrays of CHARACTER*8 strings. The remainder is in the form of INTEGER*4 parameters.

The first dictionary, DKEYW, is of all element keywords recognized by XSIF (DRIFT, SBEND, RBEND, etc.). Its dimension is set by NKEYW to 36, indicating that XSIF recognizes 36 distinct types of elements. Any keyword which is to be recognized by XSIF as a type of element must have an entry in DKEYW.

Table 1: Parameters in XSIF_SIZE_PARS.

| Name | Value | Purpose |
|---|---|---|
| MAXPOS | 49,152 | Number of elements in fully-instantiated USEd beamline |
| MAXELM | 32,768 | Number of distinct beamline elements definable |
| MAXPAR | 40,000 | Number of Named and Element parameters definable |
| MAXLST | 49,152 | Number of entries in all beamline lists |
| MAXERR | 100 | Not used in XSIF at present, to be removed |
| MXCALL | 32 | Number of nested CALL statements permitted |
| MX_WAKEFILE | 16 | Number of distinct wakefield files permitted |
| MX_WAKE_Z | 100 | Number of entries permitted in a wakefile |
| MXLINE | 1000 | Related to LINE definitions, function unclear |
| ETYPE_LENGTH | 16 | Size of TYPE strings |
| ELABL_LENGTH | 24 | Size of LABEL strings |

The remaining dictionaries are of the names of element parameters which correspond to a particular element type: for example, DQUAD is a list of the possible parameters that a quadrupole may have (L, K1, TILT, APERTURE). The dimension of DQUAD is set by parameter NQUAD. Note that there are not 36 element dictionaries, although there are 36 element types recognized by XSIF; this is because in many cases multiple element classes can use the same parameter dictionary (for example, ECOLLs and RCOLLs both use the DCOLL dictionary).

XSIF_ELEM_PARS also contains 36 INTEGER*4 parameters which are the element type numbers used to store the element type in XSIF_ELEMENTS (see section 7.1.3). For example, MAD_DRIFT is set to 1; therefore any drift elements are stored with a 1 in the element-type data structure.

Finally, XSIF_ELEM_PARS contains a set of LOGICAL*4 parameters which indicate whether various element types or element keywords are part of the NLC coding standard. NLC_KEYW contains 36 entries, in the same order as the order of element types in DKEYW; the values of NLC_KEYW are either TRUE if the corresponding element type is part of the NLC standard or FALSE if it is not; for example, NLC_KEYW(2) is FALSE because DKEYW(2) is RBEND, and RBENDs are not an accepted part of the NLC standard. Similarly, NLC_PARAM is a LOGICAL*4 2-dimensional array with dimensions NKEYW by NBEND (ie, number of parameters for an RBEND or SBEND element), which can indicate whether a particular parameter for a given element class is accepted in the standard. The array is set entirely to FALSE in XSIF_ELEM_PARS, but appropriate initializations are performed in subroutine CLEAR. For example, NLC_PARAM(9,3) is FALSE while NLC_PARAM(9,1), NLC_PARAM(9,2), NLC_PARAM(9,4), and NLC_PARAM(9,5) are TRUE; this is because SOLENOID is an accepted element type (type 9), and solenoid arguments L, KS, TILT, and APERTURE (arguments 1, 2, 4, 5, respectively) are part of the NLC standard while K1 (argument 3) is not part of the standard (but is accepted by XSIF).

### 7.1.3 Module XSIF_ELEMENTS

Module XSIF_ELEMENTS contains the data structures that store element and named parameter definitions. The way in which LIBXSIF uses these parameters will be discussed in further detail in the next section. Table 2 summarizes the variables in XSIF_ELEMENTS.

In addition to the variables in Table 2, XSIF_ELEMENTS contains two lists of wakefield file names which are read in as part of RFCAV and LCAV elements: LWAKE_FILE (for longitudinal

Table 2: Variables in XSIF_ELEMENTS.

| Name | Type | Dimensions |
|---|---|---|
| KELEM | CHARACTER*8 | MAXELM |
| KETYP | CHARACTER(ETYPE_LENGTH) | MAXELM |
| KELABL | CHARACTER(ELABL_LENGTH) | MAXELM |
| KTYPE | CHARACTER(ETYPE_LENGTH) | 1 |
| KLABL | CHARACTER(ELABL_LENGTH) | 1 |
| IETYP | INTEGER*4 | MAXELM |
| IEDAT | INTEGER*4 | MAXELM×3 |
| IELIN | INTEGER*4 | MAXELM |
| IELEM1, IELEM2 | INTEGER*4 | 1 |
| ELEM_LOCKED | LOGICAL*4 | MAXELM |
| KPARM | CHARACTER*8 | MAXPAR |
| IPTYP | INTEGER*4 | MAXPAR |
| IPDAT | INTEGER*4 | MAXPAR×2 |
| IPLIN | INTEGER*4 | MAXPAR |
| IPARM1, IPARM2 | INTEGER*4 | 1 |
| PDATA | REAL*8 | MAXPAR |
| IPNEXT | INTEGER*4 | MAXPAR |
| IPNEXT | INTEGER*4 | 1 |
| IUSED | INTEGER*4 | 1 |
| ILDAT | INTEGER*4 | MXLIST×6 |
| ITEM | INTEGER*4 | MAXPOS |

wakes) and TWAKE_FILE (for transverse wakes); up to MX_WAKEFILE of each can be stored, and NUM_LWAKE and NUM_TWAKE keep track of how many of each are present. XSIF_ELEMENTS also contains NLC_STANDARD, a logical variable which is set when elements and/or their parameters are checked against the standard, and IKEYW_GLOBAL, an auxiliary variable used in managing the MAD class construct (ie, defining an element to be an instance of another element rather than an instance of a base element class such as QUAD). Finally, there is a logical which is set true when a beamline has been successfully expanded and which is false otherwise, LINE_EXPANDED.

#### 7.1.4 Module XSIF_INOUT

This module primarily contains information related to input and output for the library. It also contains the status/error flags for the library and (for historical reasons) a set of MAD machine structure flags that are not used by LIBXSIF. Table 3 summarizes these variables.

#### 7.1.5 Module XSIF_CONSTANTS

Contains variables and parameters used to define XSIF's intrinsic named parameters (PI, TWOPI, DEGRAD, RADDEG, E, EMASS, PMASS, CLIGHT). These are made into the first eight named parameters (see next section), and initialized to values in XSIF_CONSTANTS.

#### 7.1.6 Module XSIF_INTERFACES

Contains explicit Fortran-90 style interfaces to functions and subroutines in LIBXSIF which require them: ARRCMP (compares two arrays of CHARACTER*1), ARR_TO_STR (converts a character array into a string), XPATH_EXPAND (returns a pointer to the present value of the PATH expression), XSIF_CMD_LOOP (master command manger for LIBXSIF). Any subroutine or function that makes use of these routines must access XSIF_INTERFACES via USE association.

### 7.2 How LIBXSIF Works

Most of the inner workings of the XSIF parser are fairly straightforward, but a few of its operations are quite complicated internally. Here we briefly describe a few of its more interesting pieces of internal logic.

#### 7.2.1 Named Parameters and Element Parameters

All named parameters and element parameters are stored in a single set of arrays, all with dimension MAXPAR: named parameters are stored from the bottom of the array in the order parsed, while element parameters are stored in the top of the array in the inverse of parsing order. The pointer for the named parameters is IPARM1, that for the element parameters is IPARM2 (ie, when IPARM1 == IPARM2, the parameter table is full).

All parameters are stored in a fully-symbolic manner: the actual numerical values of parameters are never evaluated in LIBXSIF operations. This is accomplished by use of the IPTYP and IPDAT arrays. When a parameter is first defined its type in IPTYP is set to -1. If the parameter is an element TILT parameter with no argument (ie taking the default value), its IPTYP is set to -2. If the parameter is set equal to a number (ie, GAMMA := 100), then IPTYP is set to 0 and the number is stored in PDATA. If the parameter is set equal to an expression (ie, GAMMA := G1 - G2), IPTYP is set to an integer from 1 to 21, where each integer corresponds to the operation (binary or unary) in the expression: 1 through 4 correspond to +,-,*,/, respectively; 11 and 12 correspond to unary + and -, respectively; 13 through 21 correspond to SQRT, LOG, EXP, SIN,

Table 3: Variables in XSIF_INOUT.

| Name | Type | Description |
|---|---|---|
| XSIF_VERSION | CHARACTER*16 | Parser version information |
| XSIF_VERS_DATE | CHARACTER*11 | Parser version date (VMS-format) |
| KDATE | CHARACTER*11 | Date (VMS-format) of the XSIF run |
| KTIME | CHARACTER*8 | Time (HH:MM:SS) of the XSIF run |
| IDATA | INTEGER*4 | logical unit for input stream |
| IPRNT | INTEGER*4 | logical unit for output stream |
| IECHO | INTEGER*4 | logical unit for error stream |
| ISCRN | INTEGER*4 | logical unit for CRT (set to 6) |
| ILINE | INTEGER*4 | number of lines parsed |
| ILCOM | INTEGER*4 | used as pointer into IELIN and IPLIN |
| ICOL | INTEGER*4 | column position of parser in present line |
| IMARK | INTEGER*4 | used for writing blank spaces in warning messages |
| NWARN | INTEGER*4 | number of warnings so far |
| NFAIL | INTEGER*4 | number of fatal errors so far |
| SCAN | LOGICAL*4 | indicates "scan mode" |
| ERROR | LOGICAL*4 | indicates error has occurred |
| ENDFIL | LOGICAL*4 | indicates unexpected EOF encountered |
| INTER | LOGICAL*4 | indicates interactive mode (always FALSE) |
| KTEXT | CHARACTER*80 | contains most recent line read from input |
| KLINE | CHARACTER*1(81) | EQUIVALENCEd to above, col 81 is semicolon |
| KTEXT_ORIG | CHARACTER*80 | same as KTEXT but original case preserved |
| KLINE_ORIG | CHARACTER*1(81) | same as KLINE but original case preserved |
| NOECHO | INTEGER*4 | used to indicate whether to echo input stream |
| LEV | INTEGER*4 | used in evaluating parameter expressions |
| IOP | INTEGER*4(50) | used in evaluating parameter expressions |
| IVAL | INTEGER*4(50) | used in evaluating parameter expressions |
| IO_UNIT | INTEGER*4(MXCALL) | contains stack of CALL'ed IO units |
| NUM_CALL | INTEGER*4 | depth of CALL stack |
| PATH_PTR | CHARACTER | pointer to present PATH expression |
| PATH_LCL | CHARACTER*1 | local path (".") |
| LOTOUP | CHARACTER*26 | used in conversion to upper case |
| UPTOLO | CHARACTER*1(26) | used in conversion to upper case |
| FATAL_READ_ERROR | LOGICAL*4 | indicates fatal read error has occurred |
| XSIF_STOP | LOGICAL*4 | indicates that parsing should terminate ASAP |
| NLC_STD | LOGICAL*4 | indicates that NLC standard warnings are desired |
| XUSE_FROM_FILE | LOGICAL*4 | indicates USE statement in input stream |

COS, ATAN, ASIN, ABS, TAN, respectively. The operands of the expression are pointed to by IPDAT(*,1) and (*,2): for the expression GAMMA := G1 - G2, for example, IPTYP would be 2 (binary -), while IPDAT for GAMMA would point to the entries for G1 and G2 in the parameter table; for unary operations, only IPDAT(*,2) is used. In the case of more complicated expressions (ie GAMMA := G1 - SIN(G2/G3)), the expression is broken into a series of unary and binary operations, and temporary parameters are assigned: in the case above there would be a temporary parameter with IPTYP = 14 and IPDAT pointing to G2 and G3; the entry for GAMMA would be IPTYP = 2 and IPDAT pointing at G1 and the temporary parameter.

For all parameters, IPLIN records the input line in which the parameter was defined.

### 7.2.2 Elements and Beamlines

Elements and beamlines are stored in a set of arrays quite analogous to the arrays that store named parameters and element parameters. Array IELIN records the input line in which a beamline or element is defined (or redefined); array KELEM records the name; arrays IETYP and IEDAT record necessary parameters for the element or beamline. In addition, there is an array, ELEM_LOCKED, which is of type LOGICAL and indicates that the corresponding element may not be redefined; this is the case if the element is used as a template for a class of other, similar elements. Also, the Type and Label character strings are stored in the KETYP and KELABL arrays.

When an element is stored in the arrays, IETYP is the element-type parameter from module XSIF_ELEM_PARS which corresponds to the element's type (i.e., 1 for drift, 2 for RBEND, etc.). IEDAT(*,1) and IEDAT(*,2) point into the parameter data arrays, at the first and last parameters of the element; the element's parameters are stored sequentially, so all the parameters between these belong to the element in question. IEDAT(*,3) is not used by elements. Elements are stored from the bottom of the arrays, and IELEM1 is the pointer to the last-filled slot in these arrays.

Like elements, beamlines are stored in the bottom of the IE* arrays. For a beamline, IETYP is set to zero. IEDAT(*,1) and IEDAT(*,2) point to any formal parameters for the beamline, which are stored antisequentially in the top of the IE* arrays, and pointed to by IELEM2 (ie, when IELEM1 == IELEM2, the element table is full). For example, an XSIF statement `BL1(SF,SD): LINE = (...)` would result in BL1 being stored in the bottom of the IE* arrays, SF and SD being stored at the top of the IE* arrays, and IEDAT(*,1) and IEDAT(*,2) for BL1 would point to the entries for SD and SF, respectively, at the top of the IE* arrays. For formal parameters, IETYP, IEDAT(*,1), and IEDAT(*,2) are all zero. For beamlines and formal arguments, IEDAT(*,3) points into the beamline's first entry in the master beamline list table, ILDAT, which we describe below.

All beamline lists are stored in ILDAT, which has dimensions MAXLST by 6; IUSED points to the last-allocated entry in ILDAT. Like elements and parameters, XSIF stores beamline lists symbolically. The functions of the entries in ILDAT change depending on whether the list entry in question is a "head cell" (entry that denotes the beginning of a line or sub-line), a named element or beamline, a sub-list element, or an actual argument of a beamline that was defined with formal arguments. The machinery is complex, and will not be further discussed here.

When a beamline is selected by the USE command, it is expanded into the ITEM array, which has dimension MAXPOS. This is the fully-instantiated list of actual elements that make up the expanded beamline. The pointers to the first and last entry of the expanded beamline in ITEM are NPOS1 and NPOS2. Most of the entries in ITEM are pointers into the IE* list of elements; if the expanded beamline contains sub-lines, these are also expanded and their beginnings and end marked by special marker cells in ITEM. This is why algorithms which strip the information out of ITEM for use in other programs' native beamline storage system test each entry in ITEM to see whether it is, in fact, a beamline element.

## 7.3 Useful Subroutines and Functions in XSIF

LIBXSIF contains many routines which are useful both in the preparation of an XSIF interface to an existing program and in more general terms. We briefly describe some of these below.

### 7.3.1 Subroutine ELMDEF

ELMDEF is the subroutine which controls the parsing and storage of beamline elements. As such it is an excellent example of how to write a parsing routine for non-standard elements which the user might want to add to the XSIF standard.

### 7.3.2 Subroutine PARAM

PARAM is the subroutine which manages the parsing and storage of named parameters.

### 7.3.3 Subroutines LINE and DECLST

These subroutines manage the parsing and storage of beamline element lists, including LINE definitions.

### 7.3.4 Subroutines XUSE and EXPAND, Function XUSE2

Subroutine XUSE manages the expansion of a beamline specified in a USE command; most of the actual work of expansion is performed by EXPAND. XUSE2 is an INTEGER*4 function which takes as argument the name of a beamline to be expanded. If expansion is successful, XUSE2 returns zero, otherwise XSIF_PARSE_ERROR is returned. Upon successful expansion of a beamline, LINE_EXPANDED is set to TRUE.

### 7.3.5 Function PARCHK

Examines all named parameters to determine whether any undefined parameters exist. Takes one argument, a LOGICAL*4. PARCHK is INTEGER*4 and returns zero if all parameters are defined; if undefined parameters are detected, warning messages are printed to the screen and the error stream, and a value of either +1 or -1 is returned; the former value is returned if PARCHK's argument is FALSE, while the latter is returned if PARCHK's argument is TRUE. This allows the user to decide whether an undefined parameter is an ERROR or a WARNING.

### 7.3.6 Subroutine PARORD

PARORD examines the functional relationships between parameters in XSIF and determines the order in which they must be numerically evaluated; the order is stored in array IPNEXT in XSIF_ELEMENTS. This prevents subroutine PAREVL (see section 7.3.7) from attempting to determine the numerical value of a parameter which depends on another parameter which has not yet been computed.

### 7.3.7 Subroutines DECEXP and PAREVL

Subroutine PAREVL goes down the list stored in IPNEXT and evaluates the numeric values of all parameters; these are stored in PDATA in XSIF_ELEMENTS. DECEXP translates parameter expressions into the numerical codes stored in the IP* tables; these codes permit the algebraic

expressions to be stored for evaluation by PAREVL. Consequently, if a new mathematical function is to be added to XSIF, both DECEXP and PAREVL must be modified.

### 7.3.8 Subroutine RDLOOK

RDLOOK performs lookup of a CHARACTER*8 string in a dictionary of CHARACTER*8 strings. RDLOOK takes as arguments: the CHARACTER*8 string which is to be looked up, its length (as INTEGER*4), the dictionary (an array of CHARACTER*8 strings), the minimum and maximum indices of interest in the dictionary (as INTEGER*4), and the position of the string of interest in the dictionary (as INTEGER*4); this is set to zero if the string is not found.

### 7.3.9 Subroutines RDINIT and CLEAR

RDINIT initializes various values related to logical I/O units used by XSIF, and CLEAR clears and initializes parameters related to element parsing.

### 7.3.10 Function XSIF_IO_SETUP and Subroutine XSIF_IO_CLOSE

XSIF_IO_SETUP performs all necessary configuration of input and output (including calling RDINIT and CLEAR), and returns zero if successful or an error message otherwise; XSIF_IO_CLOSE, conversely, attempts to gracefully close all files opened by XSIF.

### 7.3.11 Function XSIF_CMD_LOOP

This executes the master loop which examines XSIF input for commands, parameters, elements, etc. It returns zero if no errors occurred, otherwise it returns an error signal.

## 8 Where to Get LIBXSIF

All of LIBXSIF (library files, source files, module files, etc.) can be downloaded over the Internet from:
`http://www.slac.stanford.edu/accel/nlc/local/AccelPhysics/codes/xsif`

This web page will allow access to several subfolders including `doc` (this Note plus the NLC Coding Standards document), `src` (source files), `bin` (Solaris binary `libxsif_sun.a` and module files), `binnt` (Windows NT binary `xsif.lib` and module files), and `binaix` (legacy AIX binaries, since SLAC no longer supports AIX). A Linux version of LIBXSIF is expected in the second half of 2001.

## 9 Acknowledgements

The authors would like to thank the authors of MAD and DIMAD, Linda Hendrickson, Francois Ostiguy, and Mark Woodley for their kind assistance.

## References

[1] D.C. Carey and F.C. Iselin, "A Standard Input Language for Particle Beam and Accelerator Computer Programs," Proceedings of the 1984 Summer Study on the Design and Utilization of the Superconducting Super Collider, Snowmass, Colorado (1984).

[2] K.L. Brown *et al,* "TRANSPORT: A Computer Program for Designing Charged Particle Beam Transport Systems," SLAC-Report-91 Rev. 2 (1977).

[3] H. Grote, "The MAD Program User's Reference Manual," CERN/SL/90-13 (AP) Rev. 5 (1996). See also CERN-LEP-TH notes 83-30, 85-15, 85-38, and 87-33.

[4] R.V. Servranckx *et al*, "User's Guide to the Program DIMAD," SLAC-Report-285 (1990).

[5] D.C. Carey *et al*, "Third-Order TRANSPORT with MAD Input," SLAC-Report-530 (1998).

[6] D.C. Carey *et al*, "TURTLE with MAD Input," SLAC-Report-544 (1999).

[7] C. Hawkes and M.J. Lee, "Recent Upgrading of the Modeling Program COMFORT," SLAC-CN-342 (1986).

[8] R. Assmann *et al,* "LIAR: A Computer Program for the Modeling and Simulation of High Performance Linacs," SLAC-AP-103 (1997).

[9] P. Tenenbaum and M. Woodley, "Next Linear Collider Beamline Decks Coding Standards, Rev. 5," at:
`http://www.slac.stanford.edu/accel/nlc/local/lattice/documentation/deckstandards.ps` (1999).