BNL-104879-2014-TECH

AGS/AD/Tech Note No. 463;BNL-104879-2014-IR

BNL MAD Program Notes: Upgrade to Version 7C

J. Niederer

July 1997

Collider Accelerator Department

Brookhaven National Laboratory

U.S. Department of Energy

USDOE Office of Science (SC)

DISCLAIMER

For Internal Distribution Only

Accelerator Division
Alternating Gradient Synchrotron Department
BROOKHAVEN NATIONAL LABORATORY
Upton, New York 11973

Accelerator Division
Technical Note

AGS/AD/Tech. Note No. 463

BNL MAD Program Notes

Upgrade to Version 7C

J. Niederer

July 14, 1997

BNL MAD Program Notes

Upgrade to Version 7C

J. Niederer

July 14, 1997

1. Introduction

Extensive changes have been made to the BNL enhanced version of the MAD Program used at the AGS. All names in the program command language have been increased to at least 16 characters in length to agree with CERN versions, and to help with local machine element naming practices. The prevailing CERN practice of describing a lattice in terms of locations of magnets has been added, along with corresponding lattice editing features. Other changes allow unique features of our BNL Version to be used for improved matching, in particular for CERN LHC and other applications. Similarly the graphics of the BNL Version can be used on other lattices expressed in terms of MAD Version 8 conventions without further changes. Multiple passes with iterated error distributions, and corresponding graphics, can now be obtained with new features of the BNL Runtwiss command. This new version will be called BNL MAD Version 7.C, and older versions will no longer be maintained. New dictionary versions are provided. Most changes should otherwise be transparent to present lattice descriptions and command files, except of course for the older mistakes which have been corrected. Trial execution and dictionary files may be copied from the directory: **/disk11/jn/CMad+** on the AGS *ad1* computer.

2. CERN Sequence Commands

A style of describing lattices in terms of the emplacement of magnets has evolved at CERN, along with several schemes for simplifying the repeated use of similar classes of magnets. A series of names of magnets and their positions, along with any attributes that differ from those of some common class, is called a **Sequence,** with a particular format. The details are described in the CERN Version 8 MAD Manuals. The program provides for the drifts, and repetitious attributes, such as magnet lengths and strengths, offering some conveniences over the usual **LINE** kind of description. A similar internal lattice map is generated from either of the two descriptions, which is then used by the various matching and tracking services. The idea is to begin with some standard lattice, and apply edit features to modify a copy of the standard lattice for a particular study, such as an insertion match. These mechanics preserve the master version, and offer a kind of history of what has been tried.

These features are provided in the BNL version by translating the Sequence list of statements about elements and their positions into an open ended direct access file, composed as a linked list, using the BNL MAD table buffer service. Edit commands insert, move, or delete various elements, or rotate to a new starting element, or reflect the entire lattice. All of this is done by merely changing pointers in a table. As these tables can be long enough to influence memory caching and paging, the table buffering service is used to reduce their resident size to a few thousand words. The lattice map building section, invoked by the **USE** command, follows the linked element list of the sequence table. It generates drifts and copies of standard classes of magnet as needed. This is all rather similar to the inheritance feature of cloning copies of basic magnetic elements in an object oriented environment. Sequence services are coded in the new routines *Seqdef, Seqedit* and *Seqtomap,* and their supporting routines, all in the *Line.C* Source File.

It should be remarked that the MAD Program is also undergoing considerable modification at CERN in several directions. One direction is a complete rewrite in terms of the C++ language and is well in progress. Another direction appears to be rather similar to what has been done at Brookhaven, but without the graphics and menu based control. There is some concern as to how these programs will be handled in the

future, and by whom, as specific LHC demands compete with further program development and distribution.

3. The Error Section

The Error section of BNL MAD has been rewritten so the basic error generating features can be repeated in place, and thus be reusable. Studies can now iterate error distributions with changed seeds and random selections without piling up new modules or other nuisances of the original coding. This section now calls INCLUDE files generated before compilation from the dictionary definitions of its commands to obtain its structure definitions, like most of the rest of BNL MAD.

4. Cycling Error Distributions with Runtwiss

Twiss tracking over iterated error distributions can now be done through new features of the **Runtwiss** command. This is a menu or batch operated command that will cycle program parameters on Vary commands, and / or various Error kinds of commands. For each parameter change, it performs a Twiss tracking run, optionally draws the Twiss computation, then makes another change, and repeats through the repertoire of prescribed changes. This command is quite effective in showing the range of expected spread in variables such as the optical functions as a given parameter is varied.

New attributes for the **Runtwiss** command are:

Errorlist	List of the names of **Ealign** and / or **Efield** alignment and field error description commands to be applied. (**Xmenu** form of list.) For each new value of a seed for the random generator, all errors implied by the list of error commands are evaluated, and a Twiss pass is performed.
Seedlist	Optional list of integers to be used as seeds for the MAD random number generator - one seed per error and Twiss pass. (**Mvector** form - integer vector)
Nseed(2)	1. Number of trials (passes) if *Seedlist* not given. Small integer. 2. Increment to be added to current seed. Large integer. Default is 7777777

If operated in the menu mode, the error calculations are obtained by selecting the *Run Errors* button on the main popup menu. The passes can be single stepped using the *E Steps* button near the bottom of the menu. *E Steps* is a toggle which reverses the current setting of the single step option, which is initially off.

An example of the error sequence iteration for a spallation source booster model is given in Figure 1. (Splash) The Figure shows the scatter expected among 50 runs for a configuration in which Gaussian field errors are applied to various quads

The Monitor classes of lattice elements will now optionally record a copy of the optical functions during each Twiss pass. This allows orbit readings to be tabulated over a series of orbit passes by the **Runtwiss** command. A given monitor will record orbit functions when the logical flag *Optics* is set on the statement that defines the monitor. A table is obtained by defining a **Snoopy** data gathering command and its various lists and buffers, and attaching it to a **Runtwiss** session by means of a **Rundata** command. An example data setup for the Splash example is appended.

5. Savebeta and Beta0

The **Savebeta** command causes a snapshot of the principal orbit functions to be taken at a particular point in an **FTwiss** run. It delivers output to a **Beta0** module, which is suitable as input to **FTwiss, FMatch,** and **FConstraint** commands. The functions recorded on the **Beta0** module are: Betx, Bety, Alfx, Alfy, Dx, Dy, Dpx, Dpy, Mux, Muy, T, Pt, Wx, Wy, Phix, Phiy, Ddx, Ddy, Ddpx, Ddpy, Dmux, and Dmuy. The command and its use corresponds to the CERN v8 descriptions.

Attributes for **Savebeta** are:

Label	The name of the **Beta0** module to be generated.
Place	The place in the lattice where the optical functions are to be gathered. Normally this is the name of a lattice element. For a Beam Line, the end of the line is used.

6. General Coding Changes

Most of this upgrade also involves code changes intended to make names of program variables, and coding patterns and styles more consistent throughout the program. As programs age, they get harder to maintain and accept changes and new features safely unless this kind of periodic housekeeping and busy work is done. In principle, this also makes it easier for others besides the authors to penetrate the code of a program. The longer name lengths have caused considerable complication in the code, as they introduce name / character data groups which no longer have the same 8 byte lengths in the data base as the numeric data groups. About half of the approximately 150,000 lines of code have been changed in some way during the last two months, albeit much of it cosmetically for better clarity. Users of the program should benefit from increased speed, corrected mistakes, and fewer crashes.

7. Coding Style

The internal coding of BNL MAD, being based on an object oriented data base, has to deal with various data types, often in a rather invasive and unsatisfactory manner. The particular BNL style used leads to fast, reliable execution of the major computations, but also to considerable complications that some think are often handled better by other programming languages. Without laboring these matters, several naming conventions have been adopted to reduce the confusion caused by intermixing the various types of data in the code and the data base interface. The intent is to spare the casual reader of the code the nuisance of looking up the data types while trying to understand what a given statement is trying to do. With proper attention to coding style, and particularly in naming variables, it is hopefully evident whether a given statement is dealing with decimal or integer arithmetic, logical expressions, or names and character strings.

7.1. Data Base Names

The names of BNL MAD data base variables generally have a prefix, an underscore, and a name part that is the same as the name of the variable in the dictionaries that define the MAD user language. This naming scheme has been adapted from a quite similar one used for the SLAC control system software. These names are generated by a step in a UNIX make procedure that reads the dictionary that defines the user language, and converts most of the entries into INCLUDE files that resemble Fortran structure definitions. Such variables also have an index, which is used in the sense of a pointer to the first member of the data base structure holding the variable. This kind of an index - "pointer" normally addresses 8 byte data base words, and in the original MAD style, begins with the letter "i".

quad_l(ielem)

quad_k1(ielem)

quad_tilt(ielem)

Most such names of variables have a corresponding integer equivalence as part of the structure, so any pointers and tags which are part of the data group can be referenced. Data base names of integers all begin with the letter "m". A typical data group, integer or real, has the form

Number	
Tag	Pointer

Pointers such as *ielem* also have to deal with 4 byte variables, such as reals and integers, as well as 8 byte variables, such as decimal numbers. The convention used to name 4 byte pointers is to repeat the first character of the corresponding 8 byte structure pointer:

mquad_k1(iielem + kktag)

mquad_k1(iielem + kkptr)

This first example refers to the *tag* associated with the data group for the quad_k1 variable. A tag is a small integer which is set if the variable was found on the statement which defined this particular quad. There is a specific value of the tag for each data type, integer, real, name, etc. The second example refers to a pointer that shows the program how to derive the value of k1 from a parameter expression. All names of numerical values are in lower case.

7.2. Names, Characters

All variables which either are of a name data type in the data base, or are declared as a simple character data type in Fortran, are supposed to have their first character written in capitals, to note the data type. This applies also to Character, String, Word, Beam, Place, Vari, and Namegroup data types of the data base, all of which involve at least one 8 character word.

7.3. Real * 4

Four byte reals are used mainly for graphics, as the SGI graphics routines used at BNL deal with relatively small screen dimensions in pixels, and expect four byte arguments. An effort has been made to prefix names of these variables with the letter "r" in the two graphics section source files, but not consistently in the data base. Pointer references to such 4 byte names in the data base also have the first letter of the pointer repeated:

axis_wends(iiaxis)

This 4 byte situation could be handled better, but involves a lot of work in parts of the program which are not likely to be visited very often.

7.4. Logicals

Logical variables in the data base are stored usually as 8 byte reals (= 0. or 1.), and often are not obviously of logical data type from their name. Within the program most logicals not from the data base have an underscore somewhere in the name of the variable, such as *er_flag*. The main exception is *error*.

8. Manuals

Manuals covering BNL changes to the MAD Program are found in the directory **/disk11/jn/Docum+** of the AGS *ad1* computer. These are written in the older UNIX roff format. These manuals can be printed and viewed with the UNIX based commands:

alias teq. 'cat * | tbl | eqn | psroff -t -ms > ppp; lp ppp'
alias teqV. 'cat * | tbl | eqn | psroff -t -ms > ppp; xpsview ppp'

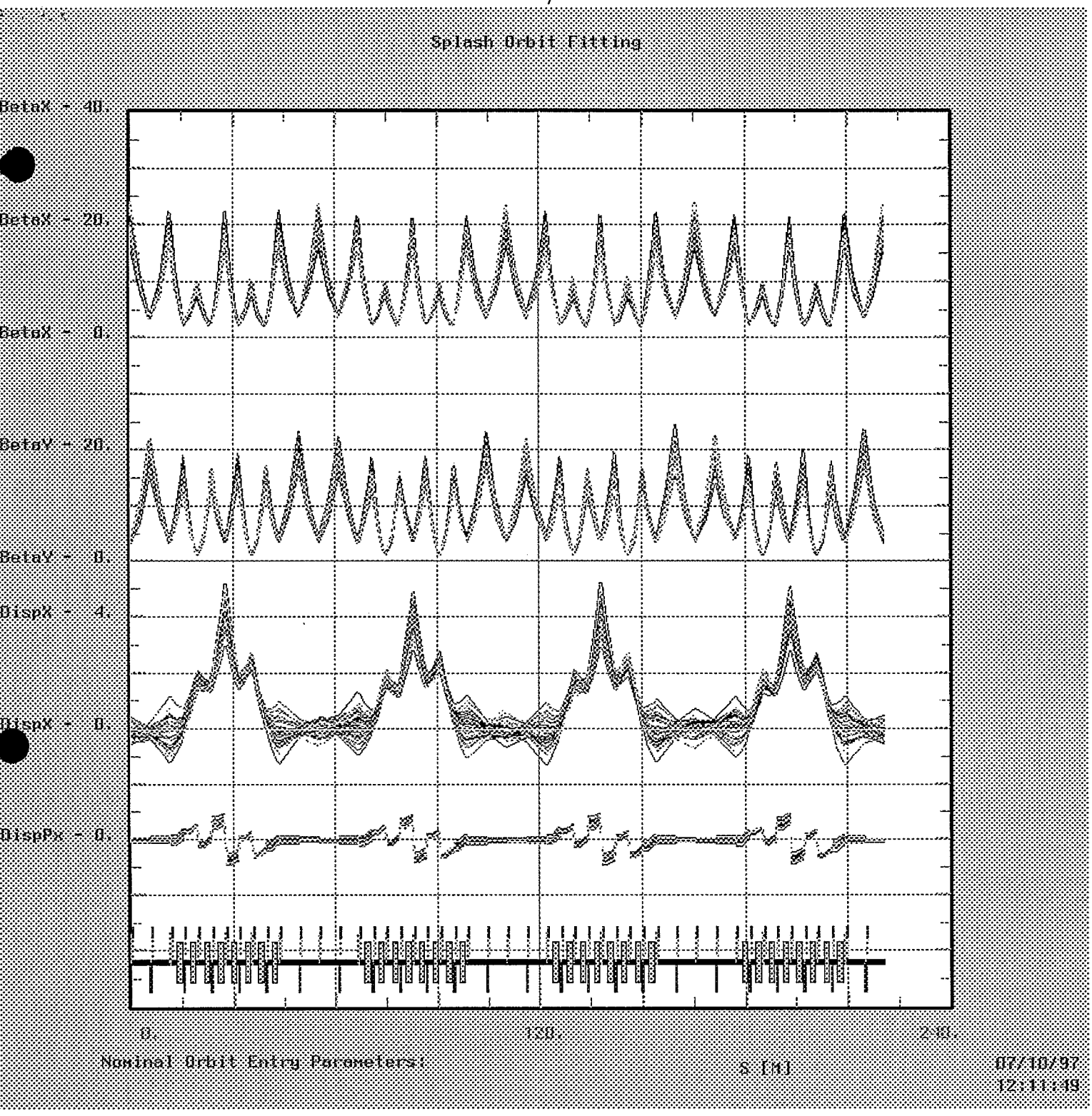
teq. Madc.man

The current upgrades have so far been covered in:

Madc.man
Fmatch.man
Twissdriver.man
Snoopy.man

```
!!    Demo File for Error Iterations on Splash Lattice
!           File = "Madc.data"
!             July 7, 1997

! Field Error Definitions, for Quads.
Ef_1   Efield, QHA, DKL(1) = 0.01 * TGAUSS(2.5) * 0.50 * KHA / Brho
Ef_2   Efield, QVA, DKL(1) = 0.01 * TGAUSS(2.5) * 0.50 * KVA / Brho
Ef_3   Efield, QHC, DKL(1) = 0.01 * TGAUSS(2.5) * 0.50 * KHC / Brho
Ef_4   Efield, QVS, DKL(1) = 0.01 * TGAUSS(2.5) * 0.50 * KVS / Brho
Ef_5   Efield, QHE, DKL(1) = 0.01 * TGAUSS(2.5) * 0.50 * KHE / Brho

Store

!     Twiss, with dP to Be Varied.
Pdelta     Param    = 0.
Ftw_1      Ftwiss,  Drawcom = DrawC5,  Deltap = Pdelta

!     Variables to Be Tabulated. Beta at Horizontal Monitors.
"Var_list1"   Varlist,   N1 = 48,                                          &
  PUEA1(1:-2)[BETAX],     PUEA2(1:-2)[BETAX],     PUEA3(1:-2)[BETAX],    &
  PUEA4(1:-2)[BETAX],     PUEA5(1:-2)[BETAX],     PUEA6(1:-2)[BETAX],    &
     .........

!     Labels for Snoopy Data Fields, 48 Monitors.
 "Lab_list1"   Xmenu,     N1 = 48,                                          &
   PUEA1,     PUEA2,     PUEA3,     PUEA4,    &
   PUEA5,     PUEA6,     PUEA7,     PUEA8,    &
      ........

   Snoopy - Data Gathering Service.
"SnoopSP1"  Snoopy   Var       = "Var_list1",    Labels = "Lab_list1",    &
                     Table     = "SNP_quads",                             &
                     Types     = "SNP_typs1",                             &
                     Buffer    = "SNPbuff1",                              &
                     Freq      = 1
!   Buffer Description - Holds Data Gathered by Snoopy at End of Twiss Pass.
 "SNPbuff1" Buffer    Tabletyp = "#Snoopy#",      Words   = 1,             &
                      Variable =  48,             Items   = 1,             &
                      Events   =  20,             Records = 5,             &
                      Spares   = 3000,   -All,    Take,                    &
                      Write,              Clear, -Recover,    Direct

!   List of Snoopy Commands to Be Operated during Twiss. Attach to Rundata.
 "SNP_list1"   Xmenu,       "SnoopSP1"
!   Data Types Vector for Snoopy Command.  Passed to Snoopdumps.
 "SNP_typs1"   Mvector      N1 = 48, 48 ^ 0

!      Snoop Dump Command, to Print Group of 12 Monitors from Table.
 "SNP_dump1a"  Snoopdump,   Snoop = "SnoopSP1",    Fact = "SP1_facts",    &
                            Mask  = "SP1_masksa",  lcol = 160
!   Scale Factors for Printing Snoopy Fields.  Default Here.
 "SP1_facts"   Rvector,     N1 = 48,  48 ^ 1.
!   Masks for Selecting Snoopy Fields to Print. 1 - 12 Here.
 "SP1_masksa"  Mvector,     N1 = 48,  48 ^ 0,  Mvalue(1)  = 12 ^ 1
```

```
!       Vary for Momentum Sensitivity.
 _deltap"    Fvary, Pdelta,    Step = .001,       Fill,   Nsteps = 21
"Vvv_list"   Xmenu  = "V_deltap"

!    Drawing List for Runtwiss Rundata. Draw from Vector, Schema, etc
  Drw_list  Xmenu =   DrawC5,    T.schema,  T1.orbx,   T2.orbx

!    Define Rundata to Gather Various Lists for Runtwiss.
  Rdtw    Rundata    Model    = MAD,       Machine = Circ,                &
          Drawlist   = Drw_list,                                          &
          Varylist   = "Vvv_list",                                        &
          Ftwiss     = Ftw_1,                                             &
          Snooplist  = "SNP_list1",                                       &
          Sliders    = SL_Pdelt
!    Error List, Practice Seed List for Runtwiss.
 "Err_list"   Xmenu   = Ef_1,  Ef_2,  Ef_3,  Ef_4,  Ef_5
 "Seed_list"  Mvector = 123456789,  234567891, 345678912, 456789123,     &
                        567891234

!    Define a Runtwiss Command with Errors,
     Rtww    Runtwiss  Menu, Plotdef = "Pdef.ax",                         &
                       Errorlist     = "Err_list",                       &
                       Nseed         = 50, 77777777
!                      Seedlist      = "Seed_list"
     Endstore

        Enable Rundata, Operate Runtwiss Session.
     Rdtw ;           Rtww

!    Print Table Listings by Groups of 12 Monitors.
    "SNP_dump1a"  ; "SNP_dump1b"  ; "SNP_dump1c"  ; "SNP_dump1d"

      RETURN
```


Splash Orbit Fitting
BetaX = 40.
BetaX = 20.
BetaX = 0.
BetaY = 20.
BetaY = 0.
DispX = 4.
DispX = 0.
DispPx = 0.
0.
150.
240.
Nominal Orbit Entry Parameters:
S [m]
07/10/97
12:11:49