Table of Contents

SixTrack **1**
SixTrack Wiki Page 1
Useful Links 1

SixTrackBeamBeam **2**
Information on Beam Beam 2

SixTrackBuild **3**
SixTrack Build 3
Basic building with gfortran on Ubuntu 12.10 3
Readme files for BOINC compilation from Laurent 3

SixTrackDoc **8**
SixTrack Programmer Manual 8
Introductions 8
Parameters 8
Global Parameters 8
Other Parameters 9
Actual Tracking 9
Synchrotron Motion 9
Flags 10
Numerical parameters 10
SINGLE ELEMENT Variables 10
General Information (SING input block): 10
Information about Specific SINGLE ELEMENT Types 11
Accelerating Cavities 12
Displacement of Elements (DISP input block) 12
AC Dipoles (DISP input block) 12
Aperture (LIMI input block) 12
Multipole Coefficients (MULT input block) 12
Current Ripples in Magnets (RIPP input block) 13
BLOC Variables 13
STRUCTURE Variables 13
Tracking in general 14
Current Ripples in Magnets 16
Particle Variables 16
Tracking in General 16
Particle Loss 17
Transfer Matrices for Linear Tracking 17
Closed Orbit Calculation 18
Beam Beam Variables 18
Leitmotifs 18
Synchrotron Motion 18
Commons 19
Relevant Ones 19
Astonishing Ones 20

SixTrackDecksBlocks **21**
SixTrack Decks 21
Decks 21
Code Blocks 26
Scripts: 32

Table of Contents

SixTrackSubRoutines **33**
Sixtrack Subroutines 33
track.f 33
Thin Lens Tracking 34
Thick Lens Tracking 34
Lost particles 34
Misc 35
sixve.f 35
Misc 35
Input Parsing 35
Some output 35
Compute values of the complex error function w(z) 36
Main program 36
Misc 36
Chromatic corrections 36
Closed orbit 36
Combination of Elements 36
Matrix Formalism 36
Misc 37
Householder transforms 38
Statistics 38
Misc 38
Misc 38
Pos processing 39
Misc 39
Beam 39
Distribution 39
Misc 39
Fit 40
Misc 40
sixvefox.f 40
Closed Orbit 40
Misc 41
Beam-Beam 41

SixTrackSubCollimat **42**

SixTrackSubadia **45**

SixTrackSubadib **46**

SixTrackSubclorb **47**
Algorithm 47
Code structure 47

SixTrackSubcomnul **49**

SixTrackSubdaten **50**
daten 50

SixTrackSubenvar **59**

Table of Contents

SixTrackSubprogram **60**

SixTrackSubthck4d **61**
thck4d 61

SixTrackSubtrauthck **63**
trauthck 63

SixTrackSubwritelin **65**

SixTrackMinutes **66**
SixTrack Minutes 66
25-Jan-2013 66

SixTrackRoadMap **67**
RoadMap 67
Physics: 67
Numerics 67
Documentation 67
Massive tracking 67
Release management: 67

SixTrackSource **69**
SixTrack Source Code organization 69
make_six options: 69
Astuce 70
Ast Files 70
Source files 71
Flags 71

SixTrack

SixTrack Wiki Page

This is a work-in-progress documentation page for SixTrack (6D particle tracking code) and related tools.

SixTrack is written in fortran. The code is organized in `.s` files that contain code blocks: these are assembled in regular `.f` fortran files by `astuce`, according to rule files `mask/*.ast` and user defined flags.

The wiki pages contain:

- SixTrackSource: information on the source code organization,
- SixTrackBuild: tips on how to build the files,
- SixTrackDoc: description of the main flow, variable and function semantics,
    - SixTrackDecksBlocks
- SixTrackSubRoutines: descriptions of the few relevant routines:
    - SixTrackSubdaten
    - SixTrackSubthck4d
    - SixTrackSubCollimat
    - SixTrackBeamBeam
    - SixTrackPostProcessing
- SixTrackMinutes: minutes of meetings,
- SixTrackRoadMap: tentative plans.

Useful Links

- SixTrack web page by Frank Schmidt;
- version of SixTrack dedicated to collimation studies;
- coupling of SixTrack to Fluka;

-- AlessioMereghetti - 16-Nov-2012

This topic: LHCAtHome > SixTrack
Topic revision: r15 - 12-Apr-2013 - RiccardoDeMaria

SixTrackBeamBeam

Information on Beam Beam

+cd beams1 !--beam-beam element !--round beam !--elliptic beam x>z !--elliptic beam z>x

+cd beams21 ktrack(i)=31 ktrack(i)=41 +cd beams22 ktrack(i)=42

+cd beams23 ktrack(i)=43

+cd beams24 !--Hirata's 6D beam-beam kick ktrack(i)=44

This topic: LHCAtHome > SixTrackBeamBeam
Topic revision: r2 - 12-Apr-2013 - JavierBarrancoGarcia

SixTrackBuild

SixTrack Build

Basic building with gfortran on Ubuntu 12.10

For linux machine with the required libraries (apt-get install libgraflib1-dev libgrafx11-1-dev libpacklib1-dev) one can use the following instructions:

Download source

```
svn co http://svn.cern.ch/guest/SixTrack/trunk/SixTrack
cd SixTrack
```

Compilation

```
 ./make_six gfortran
```

if fails:

```
cd SixTrack_4446_crlibm_gfortran_O4
gfortran  -m32 -o SixTrack_4446_crlibm_gfortran_O4 track.o sixve.o sixvefox.o dabnews.o lielib.o
```

For the 64bit the cerlib packages are broken using the multiarch mechanism. A fix can be:

```
pkgs="kernlib1 graflib1 grafx11-1 packlib1"

for ppp in $pkgs
do
apt-get download lib$ppp-gfortran:i386 lib$ppp-dev:i386
sudo dpkg -i --force-depends lib$ppp-gfortran*i386.deb
sudo dpkg -i --force-depends lib$ppp-dev*i386.deb
done
```

This creates a broken system. To fix it one can edit /var/lib/dpkg/status removing unmet dependencies.

Readme files for BOINC compilation from Laurent

```
This README describes how to compile and test SixTrack with Boinc on Windows
================================================================================

note: open MVC++ 2005 projects with MVC++ 2008

Compiling Boing for Windows (revision 26144):
---------------------------------------------

cd boinc/win_build
projects libboinc_staticcrt, libboincapi_staticcrt
set Build to release mode
add file boinc_api_fortran.cpp to libboincapi_staticcrt
build
copy libboinc_staticcrt.lib libboincapi_staticcrt.lib
  from win_build/Build/Win32/Release to sixtrack directory

cd boinc/zip
```

```
project boinc_zip
set Build to release mode
build
correct errors (collisions with MVC++ headers)
rename boinc_zip.lib to libboinc_zip.lib
copy libboinc_zip.lib from zip to sixtrack directory

e.g. from sixtrack
cp ../boinc/win_build/Build/Win32/Release/libboinc_staticcrt.lib .<0d>cp ../boinc/win_build/Build


Compiling Sixtrack for MacOSX (revision 136, version 4446):
-----------------------------------------------------------

** crlibm
cd crlibm
rm -f round_ulp.c dtostr.c log.c progress.c
rm -f *.obj
icl /arch:IA32 /O2 /fp:source /fp:strict /fp:except- /MT /I. /D_CRT_SECURE_NO_WARNINGS /Qstd=c99
check for .obj files

addition_scs.c atan.c atan_fast.c cosine.c csh_fast.c disable_xp.c division_scs.c double2scs.c dt

** sixtrack
cd ..
rm -f myboinc.f
rm -f *.obj
ifort /arch:IA32 /O2 /fp:source /fp:strict /fp:except- /MT /names:lowercase /assume:underscore /c
check for .obj files

** linking
ifort /MT /VERBOSE:LIB /exe:SixTrack_4446_crlibm_bnl_ifort_boinc_api_O2 *.obj crlibm\*.obj ..\lib

** do it again with
SixTrack_4446_crlibm_bnl_ifort_boinc_api_sse2_O2 -> /QxSSE2
SixTrack_4446_crlibm_bnl_ifort_boinc_api_sse3_O2 -> /QxSSE3



This README describes how to compile and test SixTrack with Boinc on MacOSX
============================================================================


Compiling Boing for MacOSX (revision 26141):
--------------------------------------------

svn co svn+ssh://boinc.berkeley.edu/svn/trunk/boinc (1st time only)
cd boinc
svn update
svn status -u

./_autosetup

*** edit Makefile.incl
  * add
            -I$(top_srcdir)/zip
            -I$(top_srcdir)/client
    to AM_CPPFLAGS (line 20)

  * add -m32 to AM_CFLAGS (line 32)

*** edit api/boinc_api_fortran.cpp
  * deactivate data_file functions with defines

    #ifdef DATAFILE
    void boinc_parse_init_data_file_() {
```

```
        boinc_parse_init_data_file();
    }

    void boinc_write_init_data_file_() {
        boinc_write_init_data_file();
    }
    #endif
-
  * add the function about progress (required by sixtrack) [OBSOLETE]

    void boinc_sixtrack_progress_(int* n,int* total)>---{
        double test =((double) *n) /((double) *total);
        boinc_fraction_done(test);
    }

*** edit api/Makefile.am
  * add boinc_api_fortran.cpp to api_files (line 9)

*** edit lib/procinfo_mac.cpp
  * add line 22 (after <cstdio>)
        #include <cstring>
        #include <unistd.h>

make clean
  (not the first time)

./configure --disable-server --disable-client --disable-manager --disable-fcgi --enable-libraries
            --disable-shared
DARWIN      CC=gcc-mp-4.7 CXX=g++-mp-4.7
LINUX       --build=i686-pc-linux-gnu --with-boinc-platform=i686-pc-linux-gnu

(note: CC and CXX are required to link with static version of libs like libstdc++.a)

make -k
(note: ignore the warning about openGL framework)

(note: see "ls -l */*.a" or symbolic links "ls -l ../*.a")
(checks:
ls -l ../*.a
lrwxr-xr-x  1 ldeniau  staff  20 Dec 15  2011 ../libboinc.a -> boinc/lib/libboinc.a
lrwxr-xr-x  1 ldeniau  staff  24 Dec 15  2011 ../libboinc_api.a -> boinc/api/libboinc_api.a
lrwxr-xr-x  1 ldeniau  staff  24 Dec 15  2011 ../libboinc_zip.a -> boinc/zip/libboinc_zip.a

file -L ../*.a
../libboinc.a:     current ar archive random library
../libboinc_api.a: current ar archive random library
../libboinc_zip.a: current ar archive random library
)



Compiling Sixtrack for MacOSX (revision 136, version 4446):
------------------------------------------------------------

svn co svn+ssh://svn.cern.ch/reps/SixTrack/trunk/SixTrac (1st time only)
cd SixTrac
svn update
svn status -u

*** edit make_six  [OBSOLETE]
  * add    OSTYPE=`uname -s`                (line 126)
  * add    -m32 in FCF                      (line 592, under Darwin)
  * add    -mmacosx-version-min=10.5 in FCL (line 645, under Darwin)

*** delete astuce and dafor (Linux binaries from SVN), done by "make_six clean"

./make_six clean
```

```
./make_six crlibm bnlelens ifort boinc api O2 [OBSOLETE]
./make_six bnlelens boinc api sse2 O2         [NEW]

(checks:
The following selections are ON  : tilt tracking fast crlibm api cpss boinc cr bnlelens ifort sse
The following selections are OFF : cernlib naglib da collimat nagfor g77 g95 gfortran bpm beamgas

And compiler options:
FC = ifort
FCF = -m32 -mmacosx-version-min=10.5 -O2 -fp-model source -fp-model strict -fp-model no-except
FCL = -m32 -mmacosx-version-min=10.5
)

cd SixTrack_4441_crlibm_bnl_ifort_boinc_api_sse2_O2
cd crlibm ; make ; cd ..
(warning: tan.c:14: warning: conflicting types for built-in function  tan )

*** edit Makefile.boinc
  * check for -m32
  * setup BOINCAPI = ../../..
          BOINCLIB = ../../..
          BOINCZIP = ../../..
  * remove boinc_api_fortran.cpp from SRC
  * remove boinc_api_fortran.o   from OBJS
  * comment out (provided by libboinc_api.a)
      # boinc_api_fortran.o: boinc_api_fortran.cpp
      #>gcc -m32 -c -I../.. -I../../api -I../../lib -I$(BOINCZIP) boinc_api_fortran.cpp
  * change boinc_api.o to libboinc_api.a in the linking rule dependencies
    $(CRLIBM) $(BOINCAPI)/libboinc_api.a $(BOINCLIB)/libboinc.a $(BOINCZIP)/libboinc_zip.a
  * comment out the command for linking (but not the rule itself to keep dependencies)
    the linking step is done by hand below
      SixTrack_4441_crlibm_bnl_ifort_boinc_api_sse2_O2: ...
        # ...

make -f Makefile.boinc.mac

(note: to find Gnu libraries use "g++ -m32 -print-file-name=libgcc.a")

(note: final link, adapted from the makefile, must strictly be in this form!!!)
ifort -m32 -mmacosx-version-min=10.5 -o SixTrack_4446_crlibm_bnl_ifort_boinc_api_O2 dabnews.o lie
                        crlibm/crlibm.a ../../boinc/lib/libboinc.a ../../boinc/api/libboinc_api.a
                        /opt/local/lib/gcc47/gcc/x86_64-apple-darwin10/4.7.0/../../../i386/libstd
                        -L/opt/local/lib/gcc47/gcc/x86_64-apple-darwin10/4.7.0/i386 -lpthread -lm

[original]
ifort -Wl,-map,sixtrack.map -m32 -mmacosx-version-min=10.5 -o SixTrack_4446_crlibm_bnl_ifort_boin
                        ../../boinc/lib/libboinc.a ../../boinc/api/libboinc_api.a ../../boinc/zip
                        -L/opt/local/lib/gcc47/gcc/x86_64-apple-darwin10/4.7.0/i386 -lpthread -lm

(checks:
otool -L SixTrack_4441_crlibm_bnl_ifort_boinc_api_sse2_O2
SixTrack_4441_crlibm_bnl_ifort_boinc_api_sse2_O2:>------/usr/lib/libSystem.B.dylib (compatibility

file SixTrack_4441_crlibm_bnl_ifort_boinc_api_sse2_O2
SixTrack_4441_crlibm_bnl_ifort_boinc_api_sse2_O2: Mach-O executable i386
)


Running Sixtrack tests for MacOSX (revision 130, version 4441):
---------------------------------------------------------------

cd ../../../sixtrack_test
mkdir SixTrack_4441_crlibm_bnl_ifort_boinc_api_sse2_O2
cp ../SixTrack/SixTrack/SixTrack_4441_crlibm_bnl_ifort_boinc_api_sse2_O2/SixTrack_4441_crlibm_bnl
./run_pro .. SixTrack_4441_crlibm_bnl_ifort_boinc_api_sse2_O2

(checks:
```

```
look at stderr.txt and try.out in SixTrack_4441_crlibm_bnl_ifort_boinc_api_sse2_O2_pb-d-128-141-3
diff fort.6 versions in bnl and lost
)


Putting sixtrack on my AFS public (revision 130, version 4441):
---------------------------------------------------------------

scp SixTrack_4441_crlibm_bnl_ifort_boinc_api_sse2_O2 lxplus:public/sixtrack/SixTrack_4441_crlibm_
```

-- RiccardoDeMaria - 11-Jan-2013

This topic: LHCAtHome > SixTrackBuild
Topic revision: r5 - 12-Feb-2013 - RiccardoDeMaria

SixTrackDoc

SixTrack Programmer Manual

Introductions

SixTrack is wonderful, but it is bloody complicated. Thus, these Twiki pages dare to be a *coder* help, useful to unravel the magics behind. In particular:

- the present section collects the most relevant parameters and variables in thematic groups, as in the `sixtrack.s` file. When possible, a brief explanation is provided. Relevant commons are reported as well.
- SixTrackSubRoutines collects all the subroutines in SixTrack: whenever of interest, the structure of the routine is shown along with a small explanation of the most important calls / commands / operations;
- SixTrackDecksBlocks collects all decks, block names, and (hopefully soon) descriptions.

Nota Bene: *all* the variables and arrays quoted in the *current* page are filled while parsing the `fort.2` / `fort.3` file, thus by the `daten` subroutine in module `sixve.f`: whenever this is not true, the respective source is explicitly quoted.

Parameters

Global Parameters

The following table lists some *global* parameters mainly related to **sizing**: in particular, the actual number of items involved in the simulation (used as upper limit on cycles, for instance), and the maximum allowed (used for array dimension - they are fixed `PARAMETER`) are shown.

description	actual number of items	max allowed
number of tracked particles	`napx`	`npart`
number of `SINGLE ELEMENT`	`il`	`nele`
number of `BLOC`	`mblo`	`nblo`
number of `SINGLE ELEMENT` in a `BLOC`	`mel(i)`	`nelb`
number of `super-period` s	`mper`	`nper`
number of entries in the `STRUCTURE`	`mbloz`	`nblz`
number of entries in the accelerator	`iu`	-
number of rippled `SINGLE ELEMENT` s	`irco`	`nele`
number of cavity locations in the `STRUCTURE`	`ncy`	-

Nota Bene:

- `napx` : while the `fort.3` file is parsed, this parameter stores the number of *couples* of primaries to be tracked (see the Sixtrack manual); in the `maincr` program in the `sixve.f` module, it is then multiplied by 2, thus storing the *actual* number of primaries to be tracked - the operation is performed just after the initialisation of the plotting. Then, it is multiplied by `imc` (see `TRAC` input block) and `mmac` (see `FLUC` input block) - this operation is performed just before I/O on unit 32;
- `iu` is assigned by the `ord` subroutine in module `sixve.f` as `mbloz*mper`; obviously, if the accelerator is described by only *one* `super-period`, then `iu` and `mbloz` coincide. `mbloz` should not be greater than `nblz-2` (as from subroutine `daten` in `sixve.f` module), whereas `iu` seems

not to have an upper limit. Keep in mind that variables describing the accelerator are sized on `nblz`, even if cycles may iterate up to `iu`, thus with the risk of exceeding `nblz`;

- `ncy` actually stores the number of `CAV` entries in the sequence - i.e. all the positions along the accelerator where a string of cavities is collapsed in only one element - or the number of *active* cavities in the sequence - i.e. all those `SINGLE ELEMENT` s in the sequence marked as cavities (i.e. `abs(kz(j)).eq.12`), with harmonic number and lag angle different from 0.0). Usually cavities are just few elements in an accelerator, thus `ncy` doesn't have an upper limit;

Other Parameters

Actual Tracking

name	description	code	input block
`e0`	total energy of the *reference particle* [MeV]		`INIT`
`pma`	rest mass of the tracked particles [MeV/c^2]		`SYNC`
`e0f`	momentum of the *reference particle* [MeV/c]	`sqrt(e0**2-pma**2)`	-
`gammar`	inverse of the relativistic gamma of the *reference particle*	`pma/e0`	-
`kanf`	index of the entry in the `STRUCTURE` where the `GO` keyword is issued		-
`numl`	Number of tracking turns in the forward direction		`TRAC`

Nota Bene:

- `pma` : the default value is the mass of the proton as stored in the parameter `pmap`, and it is overwritten by the `daten` subroutine in the `sixve.f` module, if the `SYNC` input block is issued;

Synchrotron Motion

Many parameters are stored in the code as read in the `fort.3`. Thus, the concerned values are stored in the variables described in the manual at the `SYNC` input block.

name	description	code	input block
`qs`	synchrotron tune [N/turn]		`SYNC`
`phas`	synchrotron acceleration phase [rad]		`SYNC`
`hsy(1)`	voltage of each cavity	`u0/dble(ncy)`	`SYNC`
`hsy(2)`	- not used -		-
`hsy(3)`	RF frequency of the cavity	`(two*pi)*harm/tlen`	`SYNC`

Nota Bene:

- `qs` is never used for calculation, but for dumping. Nevertheless, it is calculated as: $\frac{V_{\mathrm{RF[MV]}}}{E_{s\mathrm{[MeV]}}} \frac{h}{2\pi\beta_s^2} |\eta \cos\phi_s|$

  and the computation is performed in the `daten` subroutine, when reading the `SYNC` input block data (many temporary variables featured by counter-intuitive naming convention are involved in the calculations, among which `halc3`, i.e. the square of the tune);
- the `hsy(3)` parameter is computed in the `daten` subroutine as: $2\pi \frac{h}{t_{\mathrm{len}}}$

  (actually through the temporary variable `halc2=harm/tlen`). It is then multiplied by `ition` by the `trauthin` / `trauthck` subroutines in the `track.f` module, in order to take into account the working regime of the cavities, and then divided by 1000, in order to match the units of measurements (`hsy(3)` is multiplied by `sigmv(i)` during tracking);
- by means of the `SYNC` input block, the user actually declares the acceleration phase in degrees, stored in the `phag` variable: the conversion to radians is performed automatically by the code, in the `daten` subroutine.

Flags

name	description	0	1	input block
ithick	thin/thick lens model	*thin* lens	*thick* lens	-
nbeam	beam-beam elements	*not* present	*present*	BEAM
iout	print input data	do *not* print	*print*	PRIN
irip	activate current ripples	*off*	*on*	RIPP
irmod2	resonance compensation	*off*	*on*	RESO
ise	research of best place for resonance compensation	*off*	*on*	SEAR
isub	sub-resonance calculation	*off*	*on*	SUBR
idp	synchrotron motion	*off*	*on*	SYNC
ition	transition energy switch	(see manual)		SYNC
iprint	print the linear optics functions at	SINGLE ELEMENT	BLOC	LINE

Numerical parameters

Short-hand notations for common numerical values:

name	value		name	value		name	value		name	value		name	value
pieni	1d-38		c1m2	1.0d-2		c1m1	1.0d-1		c1m21	1.0d-21			
zero	0.0d0		c2e3	2.0d3		c1m3	1.0d-3		c1m24	1.0d-24			
half	0.5d0		c4e3	4.0d3		c1m6	1.0d-6		c1m36	1.0d-36			
one	1.0d0		c1e4	1.0d4		c1m7	1.0d-7		c1m38	1.0d-38			
two	2.0d0		c1e12	1.0d12		c1m9	1.0d-9		c5m4	5.0d-4			
three	3.0d0		c1e13	1.0d13		c1m10	1.0d-10						
four	4.0d0		c1e15	1.0d15		c1m12	1.0d-12						
c1e1	1.0d1		c1e16	1.0d16		c1m13	1.0d-13						
c1e2	1.0d2		c180e0	180.0d0		c1m15	1.0d-15						
c1e3	1.0d3		c1e6	1.0d6		c1m18	1.0d-18						

Physical parameters:

name	description	value
pmap	Proton rest mass [MeV]	938.271998d0
pmae	Electron rest mass [MeV]	0.510998902d0
crade	Classical electron radius [m]	2.817940285d-15
clight	Speed of light [m/s]	2.99792458d8

SINGLE ELEMENT Variables

i identifies a given SINGLE ELEMENT in the SING declaration part. The order of the declaration is respected.

General Information (SING input block):

name	description
bez(i), bez0(i)	first datum in element declaration, interpreted as **name**
kz(i)	second datum in element declaration, interpreted as **type**
ed(i)	first *additional* datum in element declaration
ek(i)	second *additional* datum in element declaration
el(i)	third *additional* datum, interpreted as element **length** [m]
kp(i)	additional flag (data type: integer)

sm(i)	first *additional* datum in element declaration for a *non-linear* element, interpreted as average **multipole strength**
irm(i)	index (and *not* order) of the associated Multipole Element (see #SingElVarsMult)
dki(i,1)	*horizontal* bending kick [rad] of a multipole block (type 11)
dki(i,2)	*vertical* bending kick [rad] of a multipole block (type 11)
dki(i,3)	length [m] of the dipole that is approximated by the kick of a multipole block (type 11)

Nota Bene:

- as a general rule, the actual meaning of the first and the second additional data in the element declaration (i.e. first table) changes according to the type of SINGLE ELEMENT: please refer to the SixTrack manual for further information;
- bez0(i) is used *only locally* in subroutine daten in the sixve.f module;
- kp(i) accomplishes to different tasks, according to its value (not explicitly set by the user, but by the code):
  - 0 (default value) : no particular meaning;
  - 1 : *elliptical* aperture limitation (EL);
  - 2 : no particular meaning;
  - 3 : *rectangular* aperture limitation (RE);
  - 3 and −3 : *horizontal* (HMON) and *vertical* (VMON) monitors for closed orbit corrections;
  - 4 and −4 : *horizontal* (HCOR) and *vertical* (VCOR) correctors for closed orbit corrections
  - 5 : combination of elements;
  - 6 : accelerating cavity;
    Keep in mind that the daten subroutine in the sixve.f changes the value of kp(i) of cavities into its absolute value;
- most frequently in local do loops , kpz stores the value of kp(i) and kzz stores the value of kz(i) for the current SINGLE ELEMENT;
- sm(i) is assigned by the envar(dpp) subroutine of the sixve.f module;
- summary of possible special values of kz(i) (see the instructions of the SING input block in the SixTrack manual):

value	meaning
12 / −12	RF cavity
15 / −15	wire
16 / −16	AC Dipole
20	beam-beam element
22	'phase-trombone'
23 / −23	crab cavity
26 / −26	crab cavity - multipole order 2
27 / −27	crab cavity - multipole order 3
28 / −28	crab cavity - multipole order 4
66	Fluka element

- the arrays dki(1:nele,1:3) are filled in by the daten subroutine, at the same time as the reading of the SINGLE ELEMENT declaration (i.e. parsing of fort.2 file).

Information about Specific SINGLE ELEMENT Types

The following paragraphs list the variable storing additional information for specific types of SINGLE ELEMENT. Local Nota Bene are provided when necessary.

Accelerating Cavities

name	description	code
phasc(i)	lag phase of the cavity	el(i)*rad
itionc(i)	regime of the cavity	kz(i)/abs(kz(i))
hsyc(i)	'frequency' of the cavity	((two*pi)*ek(j))/tlen

Nota Bene:

- a SINGLE ELEMENT flagged as cavity is considered as *active* only if the harmonic number ed(i) and the lag angle ek(i) are *both* different from 0.0. Keep in mind that cavities are always treated by SixTrack as elements of zero length;
- while the phasc(i) variable is assigned for *any* declared cavity, the itionc(i) and hsyc(i) variables are assigned *only* for *active* cavities. In particular:
  hsyc(i)=((two*pi)*ek(j))/tlen;
- possible values of itionc(i):
  - 0: default;
  - 1: above transition;
  - −1: below transition;

Displacement of Elements (DISP input block)

name	description
xpl(i) and xrms(i)	value and rms of *horizontal* displacement [mm,mm]
zpl(i) and zrms(i)	value and rms of *vertical* displacement [mm,mm]

AC Dipoles (DISP input block)

name	description
nturn1(i)	number of turns free of excitation at the beginning of the run (first additional datum in element declaration)
nturn2(i)	number of turns to ramp up the excitation amplitude from 0 to ACdipAmp (second additional datum in element declaration)
nturn3(i)	number of turns of constant excitation amplitude (third additional datum in element declaration)
nturn4(i)	number of turns to ramp down the excitation amplitude (fourth additional datum in element declaration)

Aperture (LIMI input block)

name	description
apx(i)	*horizontal* aperture limitation [mm]
apz(i)	*vertical* aperture limitation [mm]
ape(1,i)	apz(i)**2 [mm^2]
ape(2,i)	apx(i)**2 [mm^2]
ape(3,i)	apx(i)**2*apz(i)**2 [mm^4]

Multipole Coefficients (MULT input block)

Despite the following arrays are sized on nele, the main index im identifies a given Multipole Coefficient element, and *not* a given SINGLE ELEMENT. i cycles on the multipole order. The order of the declaration is respected in case of both im and i. *Mapping*: irm(j) (j identifies the SINGLE ELEMENT to which the current Multipole im is associated - see #SingElVarsGen).

name	description	code
r00(im)	reference radius [mm]	r00(im)=r0
benkc(im)	bending strength of the dipole [mrad]	benkc(im)=benki
bk0(im,i)	B-value	(benki*bk0d)/r0a
ak0(im,i)	A-value	(benki*ak0d)/r0a
bka(im,i)	B-rms	(benki*bkad)/r0a
aka(im,i)	A-rms	(benki*akad)/r0a
nmu(im)	max multipole order	

Nota Bene:

- r0 and benki are temporary variables used for parsing the first line declaring the Multipole Element in the fort.3 file;
- bk0d, bkad, ak0d and akad are temporary variables used for parsing the line declaring the strengths of the multipole order i for the current Multipole Element. r0a is basically r00(im)**(i-1);

Current Ripples in Magnets (RIPP input block)

name	description
ramp(i)	amplitude of the ripple, i.e. max kick strength
rfre(i)	frequency of the ripple [number of turns]
rzph(i)	initial phase []

Nota Bene:

- nrel(j) stores the index in the SINGLE ELEMENT list of the j th element declared in the RIPP input block, acting thus as *mapping*. This is actually used only at printout level (a dedicated one for rippled elements) in the daten subroutine (almost at its end), and in the maincr program, when the values are copied in the respective arrays of entries in the accelerator structure (see #StructEntryVarsRipp);

BLOC Variables

i identifies a given BLOC in the BLOC declaration part. The order of the declaration is respected. j identifies a given SINGLE ELEMENT in the SING declaration part (see #SingleElement).

name	description
bezb(i)	name of current BLOC
elbe(i)	length of current BLOC [m]
beze(i,j)	name of the current SINGLE ELEMENT in the current BLOC
mtyp(i,j)	index (in the array of SINGLE ELEMENT s) of the current SINGLE ELEMENT in the current BLOC

Nota Bene:

- keep in mind that mel(i) is the total number of *linear* SINGLE ELEMENT contained in the current BLOC - see #GlobalParameters;

STRUCTURE Variables

The STRUCTURE of the accelerator is a sequence of entries, either BLOC of *linear* elements or single *non-linear* elements. i identifies a given entry in the sequence. The order of the declaration is respected.

Tracking in general

name	description
ic(i)	numerical identifier of the current entry
ktrack(i)	index in GOTO statements, based on the **type** of current entry
strack(i)	BLOC : **physical length** [m] / *non-linear* SINGLE ELEMENT : smiv(1,i)*1c??? (often renamed to stracki)
tiltc(i)	cos(extalign(i,3)*c1m3)
tilts(i)	sin(extalign(i,3)*c1m3)
strackc(i)	strack(i)*tiltc(i) (except for dipole edge, ek(IX)*tilts(i))
stracks(i)	strack(i)*tilts(i) (except for dipole edge, ed(IX)*tilts(i))
strackx(i)	used for dipole edge elements and solenoids
strackz(i)	used for dipole edge elements and solenoids
xsiv(1,i)	displacement of current entry in the *horizontal* direction
zsiv(1,i)	displacement of current entry in the *vertical* direction

Nota Bene:

- possible values of ic(i):
  - ic(i)=l+nblo : the current entry i is the *non-linear* SINGLE ELEMENT l;
  - ic(i)=l : the current entry i is the *linear* BLOC l;
- most frequently in local do loops , ix stores the value of ic(i) for the current entry in the sequence;
- ktrack(i) is is assigned by the subroutines trauthin or trauthck in the track.f module. Possible values (to be completed! - the condition reported for values between 11 and 30 are actually coded via GOTO statements: the real condition is thus reported):

value	meaning	condition
0	*non-linear* element	default value
1	BLOC of *linear* elements	not ic(i).gt.nblo
2	accelerating cavity	abs(kp(ic(i)-nblo)).eq.6
3	phase-trombone / matrix element	kz(i).eq.22
11	up-right bending kick (i.e. horizontal)	kz(i).eq.1
12	up-right quadrupole kick	kz(i).eq.2
13	up-right sextupole kick	kz(i).eq.3
14	up-right octupole kick	kz(i).eq.4
15	up-right decapole kick	kz(i).eq.5
16	up-right dodecapole kick	kz(i).eq.6
17	up-right 14th pole kick	kz(i).eq.7
18	up-right 16th pole kick	kz(i).eq.8
19	up-right 18th pole kick	kz(i).eq.9
20	up-right 20th pole kick	kz(i).eq.10
21	skew bending kick (i.e. vertical)	kz(i).eq.-1
22	skew quadrupole kick	kz(i).eq.-2
23	skew sextupole kick	kz(i).eq.-3
24	skew octupole kick	kz(i).eq.-4
25	skew decapole kick	kz(i).eq.-5
26	skew dodecapole kick	kz(i).eq.-6
27	skew 14th pole kick	kz(i).eq.-7
28	skew 16th pole kick	kz(i).eq.-8
29	skew 18th pole kick	kz(i).eq.-9
30	skew 20th pole kick	kz(i).eq.-10

31	see local Nota Bene	
32	see local Nota Bene	
33	multipole block - pure *hor* kick with _non_-zero length approximated by the kick	kz(i).eq.11
34	multipole block - *hor* kick with _non_-zero length approximated by the kick with higher pole orders	kz(i).eq.11
35	multipole block - pure *hor* kick with *zero* length approximated by the kick	kz(i).eq.11
36	multipole block - *hor* kick with *zero* length approximated by the kick with higher pole orders	kz(i).eq.11
37	multipole block - pure *ver* kick with _non_-zero length approximated by the kick	kz(i).eq.11
38	multipole block - *ver* kick with _non_-zero length approximated by the kick with higher pole orders	kz(i).eq.11
39	multipole block - pure *ver* kick with *zero* length approximated by the kick	kz(i).eq.11
40	multipole block - *ver* kick with *zero* length approximated by the kick with higher pole orders	kz(i).eq.11
41	beam-beam element	nbeaux(imbb(i)).eq.1
42	beam-beam element	nbeaux(imbb(i)).eq.2
43	beam-beam element	nbeaux(imbb(i)).eq.3
44	Hirata's beam-beam element	kzz.eq.20.and.parbe(ix,2).gt.0d0
45	Wire	kz(i).eq.15
51	AC Dipole	kz(i).eq.16
52	AC Dipole	kz(i).eq.-16
53	crab cavity	kz(i).eq.23
54	crab cavity	kz(i).eq.-23
55	dipedge element	kz(i).eq.24
56	solenoid	kz(i).eq.25
57	crab cavity - multipole order 2	kz(i).eq.26
58	crab cavity - multipole order 2	kz(i).eq.-26
59	crab cavity - multipole order 3	kz(i).eq.27
60	crab cavity - multipole order 3	kz(i).eq.-27
61	crab cavity - multipole order 4	kz(i).eq.28
62	crab cavity - multipole order 4	kz(i).eq.-28

- for values of ktrack(i) greater than 45, strack(i), strackc(i) and stracks(i) are not assigned;
- the actual physics needed for values of ktrack(i) greater/equal than/to 57 are implemented *only* in the thin6d subroutine in the track.f module;
- if the value of ktrack(i) is 31, then the present entry is something of *zero length*, with no effect on particle tracking but *aperture check*. In particular, possible meanings are:

code	meaning
strack(i).le.pieni	BLOC of zero length
kz(i).eq.0	inactive *non-linear* element
abs(smiv(1,i)).le.pieni	??

- if the value of ktrack(i) is 32, the present element is a MULTIPOLE with order higher than the dipole one and length approximated by the kick different from 0.0: un-like for 31, this value triggers some active code during tracking;

- the values of `tiltc(i)` and `tilts(i)` are initialised in the `comnul` subroutine to `1.0` and `0.0` respectively, and then set to non-default values by the `ord` subroutine;
- The `xsiv(nmac,nblz)` and `zsiv(nmac,nblz)` are filled in by the `maincr` program, basically right after the calculation of the linear optics. For every entry in the accelerator structure, they store a random transverse displacement computed using the parameters input with the `DISP` input block. They are then used during tracking in non-linear `SINGLE ELEMENT` s, in addition to the information about the tilt, in order to compute magnetic kicks.

Current Ripples in Magnets

These variables store the same information as declared via the `RIPP` input block: they are assigned just before I/O on unit 32, looping over all entries composing the accelerator structure: once the entry `i` is identified as the `SINGLE ELEMENT jj`, the respective data are copied (see #SingElVarsRipp).

name	description	code
`rsmi(i)`	amplitude of the ripple, i.e. max kick strength	`rsmi(i)=ramp(jj)`
`rfres(i)`	frequency of the ripple [number of turns]	`rfres(i)=rfre(jj)`
`rzphs(i)`	initial phase []	`rzphs(i)=rzph(jj)`

Particle Variables

`j` identifies a given particle. All these variables are assigned by the `maincr` subroutine, in the `sixve.f` module.

Tracking in General

name	explanation	code
`xv(1,j)`	x - horizontal position [mm]	
`yv(1,j)`	x' - horizontal direction [1E-3]	
`xv(2,j)`	y - vertical position [mm]	
`yv(2,j)`	y' - vertical direction [1E-3]	
`sigmv(j)`	Path length difference [mm]	
`ejv(j)`	Total energy [MeV]	`sqrt(ejfv(j)**2+pma**2)`
`ejfv(j)`	Momentum [MeV/c]	`sqrt(ejv(j)**2-pma**2)`
`dpsv(j)`	$\frac{\Delta p/p}{}$	`(ejfv(j)-e0f)/e0f`
`dpsv1(j)`	$1000\ \frac{\Delta p/p}{1+\Delta p/p}$	`(dpsv(j)*c1e3)/(one+dpsv(j))` or `(dpsv(j)*c1e3)*oidpsv(j)`
`dpd(j)`	$1+\Delta p/p$	`one+dpsv(j)`
`dpsq(j)`	$\sqrt{1+\Delta p/p}$	`sqrt(dpd(j))`
`oidpsv(j)`	$\frac{1}{1+\Delta p/p} = \frac{p}{p_{(j)}}$	`one/(one+dpsv(j))`
`rvv(j)`	$E_{(j)}/pc_{(j)} * pc/E = \beta/\beta_{(j)}$	`(ejv(j)*e0f)/(e0*ejfv(j))`

Nota Bene:

- the path length difference `sigmv(j)` is actually $\sigma = s - v_0 \cdot t$, as stated in the SixTrack manual, where s is the straight length of the element traveled by the particle, v_0 is the speed of the *reference* particle, and t is the time the particle needs to cover its path in the element, with its own speed;
- whenever dealing with a particle distribution **from scratch**, the arrays storing information about the longitudinal motion should be filled as well, namely `sigmv(j)`, `ejv(j)`, `ejfv(j)`, `dpsv(j)` and `oidpsv(j)`. The variable `dpsv1(j)` is then filled in subroutine `trauthin` / `trauthck` (basically few lines before calling the subroutines performing the actual tracking), whereas the variable `rvv(j)` is filled in the `maincr` program, few lines before calling `trauthin` /

`trauthck`. On the contrary, when the energy/momentum of a particle **is changed**, all these variables must be consistently updated.

Particle Loss

name	explanation
`nlostp(j)`	index of the lost particle
`pstop(nlostp(j))`	boolean flag for particle loss
`nnumxv(nlostp(j))`	revolution at which the loss has occurred
`numxv(nlostp(j))`	revolution at which the loss has occurred
`aperv(nlostp(j),1)`	horizontal aperture at loss location
`aperv(nlostp(j),2)`	vertical aperture at loss location
`iv(nlostp(j))`	entry in the `SEQUENCE` at which the loss has occurred
`ixv(nlostp(j))`	index of the `SINGLE ELEMENT` at which the loss has occurred
`xvl(1,nlostp(j))`	x - horizontal position of the lost particle [mm]
`yvl(1,nlostp(j))`	x' - horizontal direction of the lost particle [1E-3]
`xvl(2,nlostp(j))`	y - vertical position of the lost particle [mm]
`yvl(2,nlostp(j))`	y' - vertical direction of the lost particle [1E-3]
`dpsvl(nlostp(j))`	$\Delta p/p$ of the lost particle
`ejvl(nlostp(j))`	Total energy of the lost particle [MeV]
`sigmv(nlostp(j))`	Path length difference of the lost particle

Nota Bene:

- `pstop(j)`, `nnumxv(j)` and `numxv(j)` are initialised by means of a loop over `npart` in the `maincr` program in the `sixve.f` module to `.false.`, `numl` and `numl`, respectively, just after I/O on unit 32; all other variable but `nlostp(j)` are initialised at the beginning of the `maincr` program;
- when a particle hits an aperture limit, `pstop(j)` is set to `.TRUE.`;
- `nlostp(j)` is initialised by means of a loop over `npart` at the begining of the `trauthck` / `trauthin` subroutines in the `track.f` module to `j`. During tracking, if a particle gets lost, all the arrays describing the particles would have a hole: thus, particles are grouped again. Thus, after one particle is lost, `nlostp(j)` may not be equal to `j` anymore;

Transfer Matrices for Linear Tracking

In linear elements (thus in case of a `DRIFT`, a `DIPOLE`, a `QUADRUPOLE` or a `BLOC` of linear elements), the tracking is performed through the usual linear transfer matrices:

```
|z  | = | R11 R12 | |z  | + |D  | delta
|z'|   | R21 R22 | |z'|   |D'|
```

The matrices are stored in the multi-dimensional matrix `bl1v[1:6][1:2][1:npart][1:nblo]`. The first index `[1:6]` spans over the elements of the `R` matrix and the `D` array:

index	matrix element
1	`R11`
2	`R12`
3	`R21`
4	`R22`
5	`D`
6	`D'`

The second index `[1:2]` spans over the planes:

- 1: *horizontal*;
- 2: *vertical*;

The third index `[1:npart]` spans over the number of particles, and the last one `[1:nblo]` over the `BLOC` in the lattice structure of the accelerator.

It has to be kept in mind that MAD-X creates a `fort.2` file for SixTrack, dumping all the consecutive linear elements in one `BLOC`, and all the non linear elements in-between: thus, the CPU time required by matrix multiplications is drastically reduced.

Closed Orbit Calculation

The 4D closed orbit calculation is performed by the subroutine `clorb`. The 6D closed orbit calculation is performed using differential algebra (DA), the closed orbit in 4D is taken as an initial guess with the two remaining components set to zero - unless `iclo6` is set to 5 or 6 in the TRAC input block. In that case an initial guess for the 6D closed orbit is read in from fort.33.

name	explanation
`clo(1:2)`	Closed orbit `x` and `y`
`clop(1:2)`	Closed orbit `x'` and `y'`
`clo6(1:3)`	Same as above but including longitudinal coordinates.
`clop6(1:3)`	-
`iclo6r`	Switch to trigger read-in of initial guess for the 6D closed orbit from fort.33 iclo6r.eq.1, guess read from fort.33 iclo6r.eq.0, initial guess taken as 4D closed orbit with `clo6(3)` and `clop6(3)` set to zero. However, this is activated by setting `iclo6` to 5 or 6 in the TRAC input block.

Beam Beam Variables

nbb : number of beam beam elements	
sigman(j,i)	= j=1,2 i=1,nbb=
sigman2	=sigman**2
sigmaq	=sigman(1)/sigman(2), sigman(2)/sigman(1) =

Leitmotifs

This section covers some 'leitmotifs' of the code, i.e. it follows the code behavior concerning a certain theme.

Synchrotron Motion

Synchrotron motion (no matter if during acceleration or not) is triggered when the two following conditions are met:

- there is at least one cavity entry in the sequence of the accelerator. MADX usually takes care of correctly inserting the cavities in the accelerator description, according to the logical flag `CAVALL` of the `SIXTRACK` command (see the MADX manual for further information):
  - if the flag is *not triggered*, i.e. *by default*, MADX will collapse all the cavities in *only one entry* of the sequence declaration called `CAV`, without creating a corresponding entry in the `SINGLE ELEMENT` declaration;
  - if the flag *is triggered*, MADX will dump an entry in the sequence declaration for *each* cavity actually present. `SINGLE ELEMENT` entries will be dumped accordingly. No `CAV` entry will be actually dumped;

- the SYNC input block is issued in the fort.3 file. MADX dumps the needed settings in the fc.3.aux file.

Being responsible for parsing the input information, the daten subroutine in the sixve.f module takes care of the setup of the synchrotron motion:

- declaration of SINGLE ELEMENT s: if a cavity is found, its phase is stored in the phasc(i) variable, el(i) is forced to 0.0, and kp(i) is set to 6 (see end of the local GOTO loop). Moreover, if the current one is an *active cavity*, the local counter ncy2 is increased, and the itionc(i) variable is set;
- declaration of the accelerator STRUCTURE : if a CAV entry is found, the local icy counter is increased;
- SYNC input block: the idp flag is swapped from its default value of 0 to 1 *only* in case at least a CAV entry or an *active* cavity is found in the declaration of the accelerator STRUCTURE. This input block is also responsible of computing the following quantities:
  - qs, phas, hsy(1) and hsy(3), in case *no* SINGLE ELEMENT is declared as active cavity;
  - hsyc(j), in case at least one SINGLE ELEMENT is declared as *active* cavity;
- post-processing of input data: in case a SINGLE ELEMENT is declared as cavity (no matter if active or not), its phase phasc(i) is converted into rad and kz(i) is changed with its absolute value.

Thus, if the SYNC input block is not issued or there are no cavities in the accelerator structure, no synchrotron motion is performed. Moreover, if *no active* cavity is found, a dummy SINGLE ELEMENT named CAV is added to the list, no matter if the SYNC input block is present or not: this fake element is never dumped by SixTrack in the echo of SINGLE ELEMENT s (check the il1 variable in the code).

When triggering tracking, the idp or ition can disentangle if the 4d or the 6d tracking should be performed, and phas disentangle between tracking with or without acceleration. During tracking, if the fake cavity is found, the general parameters are used for describing the synchrotron motion (see #SyncMotionGlobalParameters), otherwise those of the current cavity are used (see #SyncMotionSingleElementParam).

Nota Bene: in case cavities are declared as SINGLE ELEMENT but they are *not active*, the fake CAV element will be created and added to the list. If the SYNC input block is issued, 6d tracking is performed, but no actual synchrotron motion takes place: the parameters of each SINGLE ELEMENT declared as cavity are 0.0. The CAV entry would be active, but it's not in the STRUCTURE declaration. Thus, if you trigger CAVALL in MADX, pay attention that all the parameters describing the cavity are different from 0.0.

Commons

Relevant Ones

- accelerator structure:

```
common/str /il,mper,mblo,mbloz,msym(nper),kanf,iu,ic(nblz)
```

- description of linear elements:

```
common/ell /ed(nele),el(nele),ek(nele),sm(nele),kz(nele),kp(nele)
```

- displacement of elements:

```
common/pla /xpl(nele),xrms(nele),zpl(nele),zrms(nele)
```

- parameters for synchrotron motion:

```
 common/syn/qs,e0,pma,ej(mpa),ejf(mpa),phas0,phas,hsy(3),          &
&crad,hsyc(nele),phasc(nele),dppoff,sigmoff(nblz),tlen,            &
&iicav,itionc(nele),ition,idp,ncy,ixcav
```

- linear optics:

```
 common/linop/bez(nele),elbe(nblo),bezb(nblo),ilin,nt,iprint,      &
&ntco,eui,euii,nlin,bezl(nele)
```

Astonishing Ones

```
 common/main1/                                                     &
&ekv(npart,nele),fokqv(npart),aaiv(mmul,nmac,nblz),                &
&bbiv(mmul,nmac,nblz),smiv(nmac,nblz),zsiv(nmac,nblz),             &
&xsiv(nmac,nblz),xsv(npart),zsv(npart),qw(2),qwc(3),clo0(2),       &
&clop0(2),eps(2),epsa(2),ekk(2),cr(mmul),ci(mmul),xv(2,npart),     &
&yv(2,npart),dam(npart),ekkv(npart),sigmv(npart),dpsv(npart),      &
&dp0v(npart),sigmv6(npart),dpsv6(npart),ejv(npart),ejfv(npart),    &
&xlv(npart),zlv(npart),pstop(npart),rvv(npart),                    &
&ejf0v(npart),numxv(npart),nms(npart),nlostp(npart)


 common/main2/ dpd(npart),dpsq(npart),fok(npart),rho(npart),       &
&fok1(npart),si(npart),co(npart),g(npart),gl(npart),sm1(npart),    &
&sm2(npart),sm3(npart),sm12(npart),as3(npart),as4(npart),          &
&as6(npart),sm23(npart),rhoc(npart),siq(npart),aek(npart),         &
&afok(npart),hp(npart),hm(npart),hc(npart),hs(npart),wf(npart),    &
&wfa(npart),wfhi(npart),rhoi(npart),hi(npart),fi(npart),hi1(npart),&
&xvl(2,npart),yvl(2,npart),ejvl(npart),dpsvl(npart),oidpsv(npart), &
&sigmvl(npart),iv(npart),aperv(npart,2),ixv(npart),clov(2,npart),  &
&clopv(2,npart),alf0v(npart,2),bet0v(npart,2),ampv(npart)


 common/main3/ clo6v(3,npart),clop6v(3,npart),hv(6,2,npart,nblo),  &
&bl1v(6,2,npart,nblo),tas(npart,6,6),qwcs(npart,3),di0xs(npart),   &
&di0zs(npart),dip0xs(npart),dip0zs(npart),xau(2,6),cloau(6),       &
&di0au(4),tau(6,6),tasau(npart,6,6),wx(3),x1(6),x2(6),fake(2,20)
```

-- AlessioMereghetti - 18-Nov-2012

This topic: LHCAtHome > SixTrackDoc
Topic revision: r43 - 02-Jul-2013 - AlessioMereghetti

SixTrackDecksBlocks

SixTrack Decks

Decks

- adia
- adib
- anfb
- aux
- avepol
- averaged
- beam
- betalf
- block
- blockdis
- bran
- checkpt
- chroma
- clor
- clorb
- clorb2
- combel
- comcfu
- comnul
- compcjg
- cor
- couplean
- cpart
- ctoi
- ctor
- ctord
- ctorflo
- daabs
- daadd
- daall
- daallno
- dacad
- dacct
- dacctt
- dacdi
- dacex
- dacext
- dacfu
- dacfui
- dacfuit
- dacfur
- dacfurt
- dacfut
- dachk
- daclr

- daclrd
- dacma
- dacmu
- dacmud
- dacmut
- dacom
- dacon
- dacop
- dacopd
- dacsu
- dacycle
- dadal
- dadal1
- dadcd
- dadeb
- dader
- dadert
- dadic
- dadiv
- daeps
- daexc
- daexct
- daexter
- daexx
- daexxt
- daflo
- daflod
- dafun
- dafunt
- dagauss
- dainf
- daini
- dainv
- dainvt
- dakey
- daliesix
- dalin
- dalind
- dalint
- dallsta
- damch
- damono
- damul
- damulin
- damult
- dancd
- danorm2
- danorm2t
- danormr
- danormrt
- danot
- danum
- daorder
- dapac

- dapek
- dapek0
- dapin
- dapint
- dapoi
- dapok
- dapok0
- dapokzer
- dapos
- dapri
- daprid
- daprimax
- daran
- dare
- darea
- daread
- dashift
- dasqr
- dasqrt
- dasub
- dasuc
- daswap
- daten :read input data from fort.3 and/or fort.2
- datra
- datrash
- datrashn
- davar
- davar0
- decoup
- dehash
- dfilt
- dhdj
- dhdjflo
- difd
- distance
- dlie
- dumps
- eig
- envada
- envar
- envardis
- envars
- envarsv
- errf
- errff
- error
- etall
- etall1
- etallnom
- etcct
- etcctpar
- etcjg
- etcom
- etctr

- etdiv
- etini
- etinv
- etmtree
- etpin
- etpoi
- etppulnv
- etppush
- etppush2
- etred
- etrtc
- ety
- ety2
- etyt
- exp1d
- expflo
- expflod
- expnd2
- facflo
- facflod
- fexpo
- fexpo1
- filt
- filtres
- flofac
- flofacg
- flowpara
- flush
- getdanot
- gettura
- gofix
- h2pluflo
- hash
- hdf5
- hyper
- idprset
- initpert
- inputres
- intd
- itoc
- join
- killnonl
- lib
- liefact
- lieinit
- lienot
- liepeek
- linopt
- loesd
- lubksb
- ludcmp
- maincr
- mainda
- mapflol

- mapnorm
- mapnormf
- matinv
- matrix
- midbflo
- movearou
- movemul
- mtree
- mulnd2
- mydainf
- mydaini
- myrinv
- nagdumy
- nuanaflo
- nwrtbnl
- nwrtcoll
- orbit
- ord
- orderflo
- pertpeek
- phasad
- planar
- plotdumy
- postpr
- ppush
- ppush1
- ppushlnv
- ppushpr
- prolong
- prresflo
- qmod
- ranecu
- reelflo
- resex
- respoke
- resvec
- rext
- rmod
- rotflo
- rotiflo
- rtoc
- rtocd
- rtocflo
- runcav
- runda
- search
- setidpr
- simil
- subre
- subsea
- sumpos
- sympl3
- synoda
- take

- taked
- tra
- transver
- trx
- trxflo
- umlau
- umlauf
- umschr
- wireda
- xgam
- xgbm

Code Blocks

- acdip1
- acdipkick
- alignf
- alignl
- alignsa
- alignsb
- alignu
- alignva
- alignvb
- alloc
- alloctot
- beam
- beam11
- beam11of
- beam11s
- beam12
- beam12f
- beam12of
- beam12s
- beam13
- beam13f
- beam13of
- beam13s
- beam21
- beam21of
- beam21s
- beam22
- beam22f
- beam22of
- beam22s
- beam23
- beam23f
- beam23of
- beam23s
- beama1
- beama1of
- beama1s
- beama2
- beama2f
- beama2of

- beama2s
- beama3
- beama3f
- beama3of
- beama3s
- beama4
- beama4f
- beama4o
- beama4of
- beama4s1
- beama4s2
- beamco
- beamcof
- beamcoo
- beamcou
- beamdim
- beamr1
- beamr1f
- beamr1of
- beamr2
- beamr2f
- beamr2of
- beamr2s
- beamr3
- beamr3f
- beamr3o
- beamr3of
- beamr3s1
- beamr3s2
- beams1
- beams21
- beams22
- beams23
- beams24
- beamwzf1
- beamwzf2
- bnlin
- bnlout
- bnltwiss
- bpmdata
- choice
- clor
- close
- coast
- collpara
- commadh1
- commadh2
- commadha
- commd1da
- common
- common1
- common2
- commonas
- commonc

- commond1
- commond2
- commondl
- commonds
- commonl
- commonm1
- commonmn
- commons
- commont1
- commont2
- commonta
- commontr
- commonxz
- commphin
- commtim
- crab1
- crabkick
- crco
- crcoall
- crlibco
- dabinc
- daini
- dainicom
- dalin
- dalin1
- dalin2
- dalin3
- dalin4
- dalin5
- dalino
- daname
- dano
- dascr
- database
- dbcollim
- dbcommon
- dbdaten
- dblinopt
- dbmaincr
- dbmkdist
- dbpencil
- dbthin
- dbtrthin
- dump1
- dump2
- dump3
- filtr
- funint
- ii
- info
- integratedex
- interac
- istable
- kicka0

- kicka01h
- kicka01v
- kicka02h
- kicka02v
- kicka03h
- kicka03v
- kicka04h
- kicka04v
- kicka05h
- kicka05v
- kicka10h
- kicka10v
- kickadpe
- kickaso1
- kickb01h
- kickb01v
- kickbxxh
- kickbxxv
- kickf01h
- kickf01v
- kickfdpe
- kickfho
- kickfso1
- kickfxxh
- kickfxxv
- kickl01h
- kickl01v
- kickldpe
- kicklso1
- kicklxxh
- kicklxxv
- kickq0
- kickq01h
- kickq01v
- kickq02h
- kickq02v
- kickq03h
- kickq03v
- kickq04h
- kickq04v
- kickq05h
- kickq05v
- kickq10h
- kickq10v
- kickqdpe
- kickqso1
- kicks01h
- kicks01v
- kicksho
- kicksxxh
- kicksxxv
- kicku01h
- kicku01v
- kickudpe

- kickuso1
- kickuxxh
- kickuxxv
- kickv01h
- kickv01v
- kickvdpe
- kickvho
- kickvso1
- kickvso2
- kickvxxh
- kickvxxv
- kispa10h
- kispa10v
- lost1a
- lost1b
- lost1c
- lost2
- lost2a
- lost3a
- lost3b
- lost4
- lost5a
- lost5b
- lost5c
- lostpart
- mul
- mul4v0
- mul4v01
- mul4v02
- mul4v03
- mul4v04
- mul4v05
- multb0
- multb01
- multb02
- multb03
- multb04
- multb05
- multb10
- multf01
- multf02
- multf03
- multf04
- multf05
- multl0
- multl01
- multl02
- multl03
- multl04
- multl05
- multl10
- multl11
- multl12
- multl13

- mults0
- mults01
- mults02
- mults03
- mults04
- mults05
- mults10
- multu0
- multu01
- multu02
- multu03
- multu04
- multu05
- open
- parbeam
- parnum
- parpro
- phas1so1
- phas2so1
- phas3so1
- printing
- resfile
- reson
- rhicelens
- rvet0
- rvet1
- rvet2
- save
- solenoid
- sqrtfox
- sqrtfox0
- sqrts
- sqrtv
- stable
- stra0
- stra01
- stra02
- stra03
- stra04
- stra05
- stra10
- stra11
- stra12
- stra13
- stra14
- stra2dpe
- thcklin
- timefct
- trom0
- trom01
- trom02
- trom03
- trom04
- trom05

- trom10
- trom20
- trom30
- trom40
- trom41
- trom42
- tunedef
- tunerad
- umlalid
- umlalid1
- vecflow
- version
- wire
- wirektrack

Scripts:

```
cat *.s | grep '^+if' | grep -o ' [a-z0-5]*' | sort -u
cat *.s | grep '^+dk' | grep -o ' [a-z0-5]*' | sort -u | wc
```

-- RiccardoDeMaria - 26-Sep-2012

This topic: LHCAtHome > SixTrackDecksBlocks
Topic revision: r1 - 26-Sep-2012 - RiccardoDeMaria

SixTrackSubRoutines

Sixtrack Subroutines

This is a summary of the subroutines defined in SixTrack.

track.f

This module contains all the subroutines actually responsible for the tracking and the aperture check. Other subroutines in support to the tracking are coded as well.

Two different sets of subroutines are available, depending on the model of accelerator, i.e. if in *thin* lens or in *thick* lens. Each set has four different routines:

- `trauxxxx` : a general subroutine, responsible for some final variable initialisation and calling the proper tracking routine;
- `xxxx4d` : subroutine dedicated to tracking in 4d;
- `xxxx6d` : subroutine dedicated to tracking in 6d;
- `xxxx6dua` : subroutine dedicated to tracking in 6d with acceleration.

The 4d tracking is performed whenever the `SYNC` input block is *not* issued in the `fort.3` file or when it is issued with `ition` (set if above or below transition energy) set to `0`, i.e. whenever no synchrotron motion is requested. On the contrary, the 6d tracking is performed if the same `SYNC` input block is present with `ition` set to `1` or `-1`, and the 6d tracking with acceleration is performed if the phase of the cavity `phag` is different from `0.0` (the actual check is `abs(phas).ge.pieni`). Please refer to the SixTrack manual for further information.

The most relevant difference between the *thin* and the *thick* lens models is the collapsing of consecutive linear `SINGLE ELEMENT` s in one `BLOC`. The most direct consequence is that the tracking through any `BLOC` in case of *thick* lens is treated by means of the usual linear matrix formalism, kept in memory by the `bl1v(1:6,1:2,1:npart,1,nblo)`, `al(1:6,1:2,j,1:nele)` and `as(1:6,1:2,j,1:nele)` variables: the first one is actually used in case of 4d tracking, whereas the other two are used in case of 6d tracking, w/o acceleration - see Matrix Formalism for the actual computation of these variables. In particular, the `as(1:6,1:2,j,1:nele)` multi-dimensional array seems to store the matrix elements useful for longitudinal dynamics, whereas the `al(1:6,1:2,j,1:nele)` multi-dimensional array seems to store the matrix elements useful for transverse dynamics. As expected, in case of a *thin* lens model of the accelerator, each `BLOC` contains only a single drift: each `BLOC` is thus treated simply as a drift, thus the transverse coordinates are updated with the path length travelled by the particle. While the `bl1v(1:6,1:2,1:npart,1,nblo)` and `hv(1:6,1:2,1:npart,1,nblo)` variables (the latter is used for computing matrices of any `BLOC`) are initialised at the beginning of the main program `maincr`, the `as(1:6,1:2,j,1:nele)` and `al(1:6,1:2,j,1:nele)` variables are initialised in the subroutine `comnul`.

It should be bared in mind that the only subroutine for tracking implementing also the RF Crab Cavities (`RF CC`) is the `thin6d` one, which implements also the collimation version of SixTrack.

Many of the subroutines collected in this module have the `nthinerr` error variable. This variable assumes values with specific meanings:

- `0` computation is regular;

- `3000` error while dumping particle information on unit `90 - mod(ia2-1, 32)`. This implies that some post-processing is skipped;
- `3001` all particles have been lost.

Thin Lens Tracking

Tracking in case of a *thin* lens model of the accelerator.

- trauthin(nthinerr) [809 lines]: Elena Benedetto's version has modifications on this part
- thin4d(nthinerr) [1400 lines];
- thin6d(nthinerr) [1499 lines]: it integrates also the collimation routines
- thin6dua(nthinerr) [1496 lines];

Thick Lens Tracking

Tracking in case of a *thick* lens model of the accelerator.

- trauthck(nthinerr) [806 lines];
- thck4d(nthinerr) [1416 lines];
- thck6d(nthinerr) [1511 lines];
- thck6dua(nthinerr) [1519 lines];

Lost particles

This bunch of subroutines is responsible for the aperture check. In general, all the routines have the same structure: the real difference is the type of aperture check that is performed. When tracking through a `BLOC`, all six subroutines dedicated to actual tracking (see #ThinLensTracking and #ThickLensTracking) will *skip* the aperture check, since a `BLOC` is a small sequence of linear elements, represented by only one transport matrix. All the following subroutines but the first one are called by the six subroutines dedicate to actual tracking (see #ThinLensTracking and #ThickLensTracking).

- lostpart(nthinerr) [293 lines]: aperture check against a *general rectangular* aperture, of dimension specified by the variables `aper(1)` and `aper(2)`, read from input bloc `ITER`. This subroutine is called by `thin6d`, `thin6dua` and `thck6dua` (most probably this routine could be dropped);
- lostpar2(i,ix,nthinerr) [298 lines]: aperture check against a *general rectangular* aperture, of dimension specified by the variables `aper(1)` and `aper(2)`, read from input bloc `ITER`. This subroutine differs from the previous one because the element where the particle is lost is saved, and the printout logging the loss is modified accordingly;
- lostpar3(i,ix,nthinerr) [298 lines]: aperture check against the *rectangular* aperture of the `ix SINGLE ELEMENT`, i.e. entry `i` in the `STRUCTURE` declaration;
- lostpar4(i,ix,nthinerr) [299 lines]: aperture check against the *ellyptical* aperture of the `ix SINGLE ELEMENT`, i.e. entry `i` in the `STRUCTURE` declaration;

Misc.

- synuthck [632 lines]: Update matrix elements for *linear* optics which depend on $^{\Delta p/p}$ of only *linear* `SINGLE ELEMENT` s. Basically, the subroutine loops over all the `SINGLE ELEMENT` s, and in case of a *linear* element of *non-zero length*, it loops over all the particles to re-compute those parameters explicitly dependent on the momentum - it actually updates selected entries of the `as(1:6,1:2,1:napx,1:il)` and `al(1:6,1:2,1:napx,1:il)` matrix variables in common `syos`, for the concerned elements. More information about matrices for linear optics in section Matrix Formalism.

- ripple(n) [202 lines]: ripple of quadrupole power supply, translated into a ripple in the tune;

- writebin(nthinerr) [230 lines]: output

- dist1: distance?

- write6(n) [246 lines]: output

sixve.f

Misc.

- errf(xx,yy,wx,wy) [77 lines]: Double precision complex error function. Seems to be based on an algorithm by W. Gautschi (*Efficient Computation of the Complex Error Function, SIAM Journal on Numerical Analysis, Vol. 7, No. 1 (Mar., 1970), pp. 187-198)*

- wzsubv(napx,vx,vy,vu,vv) [251 lines]: ?? vector version

- wzsub(x,y,u,v) [138 lines]: ??

- adia(numx,e0f) [152 lines]: Adiabatic energy increase. Input (1) `numx`, the current turn number (2) `e0f`, momentum of reference particle.

- adib(e0f) [146 lines]: Adiabatic energy decrease. Input (1) `e0f`, momentum of reference particle.

Input Parsing

- daten [2172 lines]: Read input data from `fort.2` and `fort.3`

- intepr(i,j,ch,ch1) [74 lines]: Input parsing helper

- splitfld(errno,nunit,lineno,nfields,nf,chars,fields) [73 lines]: Splits the chars input into space separated fields, up to `nfields` maximum. The number of fields is returned in `nf`.

- spliterr(errno,nunit,lineno,nfields,nf,lf,chars) [28 lines]: Reports any errors encountered while parsing the input files.

Some output

- write4 [172 lines]: Write modified geometry file to unit 4

Compute values of the complex error function w(z)

- wzset [60 lines]: ??
- mywwerf(x,y,wr,wi) [73 lines]: ??
- ranecu(rvec,len,mcut) [61 lines]: ??

Main program

- program maincr [1624 lines]: Main
- comnul [585 lines]: set all commons to 0

Misc.

- distance(x,clo,di0,t,dam) [87 lines]: phase-space distances for post-processing
- betalf(dpp,qw) [296 lines]: calculation of opt parameters at starting position

Chromatic corrections

- chroma [241 lines]: Chroma for 5 energy values
- chromda [269 lines]: Chromatic correction via da

Closed orbit

- clorb(dpp) [189 lines]: Calculation of the closed orbit
- clorb2(dpp) [179 lines]: As clorb, but don't write output

Combination of Elements

- combel(iql) [159 lines]: Combination of elements

Matrix Formalism

These subroutines are responsible for building the matrix formalism for linear beam dynamics, only in case of a ***thick lens*** model of the accelerator. In general, `envar*` subroutines perform the calculation for each *linear* `SINGLE ELEMENT`, while `block*` subroutines perform the calculation for each `BLOC` of *linear* elements.

Nota Bene:

1. as stated in the SixTrack manual, *linear* `SINGLE ELEMENT` s are identified as of *non-zero* length, whereas *non* linear elements have a *zero* length;
2. the subroutines `envardis(dpp,aeg,bl1eg,bl2eg)` and `blockdis(aeg,bl1eg,bl2eg)` are used *only* in the `linopt` subroutine, as a support to `envar(dpp)` and `block` subroutines, for further calculations: indeed, the subroutine `envardis(dpp,aeg,bl1eg,bl2eg)` is called with a value of `dpp` increased by `ded` (set to `c1m9` by the `daten` subroutine in the `sixve.f` module). This is also a possible reason why matrices are stored in temporary variables and not in a common;
3. the subroutine `synuthck` in the `track.f` module is responsible for updating those matrix elements of any *linear* `SINGLE ELEMENT`, which depend on $^{\Delta p/p}$;

4. in order to propagate the computation performed by the subroutine `envarsv(dpsv,oidpsv,rvv,ekv)` or by `synuthck` on each *linear* `SINGLE ELEMENT` to each `BLOC`, the subroutine `blocksv` must be called just afterwards.

- envar(dpp) [333 lines]: The computed matrices are stored in the `a(1:il,1:2,1:6)` multi-dimensional matrix variable, in common `mat`. Moreover, this subroutine assigns the `sm(i)` array. At the very end, this subroutine calls the `block` subroutine.

- envardis(dpp,aeg,bl1eg,bl2eg) [333 lines]: Exact copy of the previous subroutine, but the matrices are stored in the multi-dimensional matrix variables listed in the interface - meaning of variables is kept. The `sm(i)` variable is not modified. At the very end, this subroutine calls the `blockdis(aeg,bl1eg,bl2eg)` subroutine.

- envars(j,dpp,rv) [416 lines]: it computes the matrices for *all* linear `SINGLE ELEMENT` s, but only for particle `j`. `dpp` seems to be `dpsv(j)`, whereas `rv` seems to be `rvv(j)`. Basically, the subroutine loops over all the `SINGLE ELEMENT` s, and in case of a linear element, it computes *all* the parameters - actually `as(1:6,1:2,j,1:il)` and `al(1:6,1:2,j,1:il)` matrix variables in common `syos`, for the concerned elements. More information about matrices for linear optics in section Matrix Formalism. Differently from the two previous subroutines, it doesn't call any other subroutine for computing / updating the matrix formalism for the `BLOC` s. This subroutine is *never* used.

- envarsv(dpsv,oidpsv,rvv,ekv) [529 lines]: Same purpose as the previous subroutine, but it loops over `napx` particles. The organisation of the code is slightly different from the previous subroutine (to be noted that this subroutine doesn't call the previous one inside a loop over the particles, but it implements from scratch the calculation), despite it fills the *same* multi-dimensional *matrix variables* in the *same common*. The meaning of `dpsv`, `oidpsv`, `rvv` can be found in SixTrackDoc#ParticleVariables, whereas `ekv(npart,nele)` seems to store `ek(i)` (see SixTrackDoc#SingleElement), repeated for each particle (`i` identifies a given `SINGLE ELEMENT` in the `SING` declaration part): it is filled by the `maincr` program in the `sixve.f` module, and never touched anymore. As the previous subroutine, it doesn't call any other subroutine for computing / updating the matrix formalism for the `BLOC` s.

- block [169 lines]: It computes the matrices for each *linear* `BLOC`: those matrices are stored in the `bl1(1:mblo,1:2,1:6)` and `bl2(1:mblo,1:2,1:6)` multi-dimensional matrix variables, in the `mat` common. The former variable contains the matrix for each `BLOC` when particles travel through it in the same order as in the `BLOC` declaration, whereas the latter contains the matrix for each `BLOC` travelled in the order opposite to the declaration one.

- blockdis(aeg,bl1eg,bl2eg) [174 lines]: Exact copy of the previous subroutine, but the matrices are stored in the multi-dimensional matrix variables listed in the interface - meaning of variables is kept.

- blocksv [249 lines]: It computes the matrices for each *linear* `BLOC` and stores them in the `bl1v(1:6,1:2,1:napx,1,mblo)` multi-dimensional matrix variable. The computation is based on the `al(1:6,1:2,j,1:il)` variable only. See SixTrackDoc#TransferMatrixForLinearTracking.

Misc.

- prror(ier) [592 lines]: Print error

- linopt(dpp) [1085 lines]: Linear parameters at the position of every element of block

- writelin(nr,typ,tl,p1,t,ixwl) [203 lines]: write linear optic parameters

- cpltwis(typ,t,etl,phi) [210 lines]: twiss parameters
- loesd (rmat, vec,dimakt,dimtot,kod) [82 lines]: Solution for linear equation (vec2), vec1=vec2*mat
- matrix(dpp,am) [153 lines]: ??
- corrorb [408 lines]: Correction of closed orbit
- putorb(xinc,nx,npflag) [213 lines]: put orbit corrections
- orbinit [176 lines]: Init random number for correctness

Householder transforms

- htls(a,b,m,n,x,ipiv,r,iter,rms,ptp) [227 lines]: ??
- htal(a,m,n,k,beta) [30 lines]: ??
- htbl(a,b,m,n,k,beta) [28 lines]: ??
- htrl(a,b,m,n,k,rho) [31 lines]: ??
- htul(a,m,n,k,sig,beta) [30 lines]: ??

Statistics

- calrms(r,m,rms,ptp) [31 lines]: Calculate rms
- function maxmin (a,n,m) [23 lines]: max and min of an array

Misc.

- ord [326 lines]: Organization of blocks
- phasad(dpp,qwc) [881 lines]: Adjustement of x-phaseadvance between 2 positions
- qmod0 [361 lines]: Adjustment of the Q-values plus an additional adjustment of a x-phaseadvance between two positions in the machine
- qmodda(mm,qwc) [351 lines]: Adjustments of Q-values via differential algebra (D.A)
- umlauf(dpp,ium,ierr) [845 lines]: One turn transformation

Misc.

- resex(dpp) [1614 lines]: Calculation of driving terms for resonances (for RMCD)
- rmod(dppr) [419 lines]: strength of correction elements
- search(dpp) [199 lines]: find positions for correction elements
- subre(dpp) [1976 lines]: Calculation of resonance and subresonance driving terms
- detune(iv,ekk,ep,beta,dtu,dtup,dfac) [76 lines]: detuning

- subsea(dpp) [1586 lines]: Calculation of driving terms

Pos processing

- postpr(nfile) [1949 lines]: post processing
- sumpos [93 lines]: summarize post processing results

Misc.

- decoup [302 lines]: decoupling with matrix
- fft(ar,ai,m,n) [83 lines]: FFT
- caconv(a,b,c) [31 lines]: ??
- cphase(k,a,b,c,d,i,j,ie) [58 lines]: ??
- cinvar(a,b,c,d,j,e,xinv,invx) [64 lines]: ??
- sinpro(a,b,c,d,e) [39 lines]: ??
- join [183 lines]: ??

Beam

- beamint(np,track,param,sigzs,bcu,ibb,ne,ibtyp,ibbc) [64 lines]: Hinata's 6d beam
- boost(np,sphi,cphi,tphi,salpha,calpha,track) [70 lines]: Hinata's 6d beam boost
- sbc(np,star,cphi,nsli,f,ibtyp,ibb,bcu,track,ibbc) [150 lines]: Synchro beam coll.
- boosti(np,sphi,cphi,tphi,salpha,calpha,track) [75 lines]: Inverse boost
- bbf(sepx,sepy,sigxx,sigyy,bbfx,bbfy,bbgx,bbgy,ibtyp) [105 lines]: ??
- stsld(star,cphi,sphi,sigzs,nsli,calpha,salpha) [69 lines]: longitudinal position of the strong slice to all??

Distribution

- function gauinv(p0) [123 lines]: Inverse of integrated normal distribution

Misc.

- kerset(ercode,lgfile,limitm,limitr) [87 lines]: kernlib
- rinv(n,a,idim,ir,ifail) [115 lines]: replaces A by inverse
- dinv(n,a,idim,ir,ifail) [117 lines]: ??
- f010pr(name,n,idim,k,kprnt) [42 lines]: print error
- rfact(n,a,idim,ir,ifail,det,jfail) [81 lines]: ??
- dfact(n,a,idim,ir,ifail,det,jfail) [81 lines]: ??

- rfeqn(n,a,idim,ir,k,b) [53 lines]: ??
- dfeqn(n,a,idim,ir,k,b) [53 lines]: ??
- rfinv(n,a,idim,ir) [67 lines]: ??
- dfinv(n,a,idim,ir) [66 lines]: ??
- tmprnt(name,n,idim,k) [39 lines]: kernlib tnprnt

Fit

- lfit(x,y,l,key,a,b,e) [60 lines]: wheighted straight line fit
- lfitw(x,y,w,l,key,a,b,e) [57 lines]: wheighted straight line fit

Misc.

- logical function myisnan(arg1,arg2) [8 lines]: Compares the two input arguments `arg1` and `arg2`. If these are **NOT** equal then `.true.` is returned, otherwise `.false.` is returned. Called by the functions `acos_rn(x)`, `asin_rn(x)` and `atan2_rn(y,x)`.
- datime(nd,nt) [23 lines]: Called once by the main program (maincr) to get the current date and time. This information is used in the printout of e.g. "*SIXTRACK starts on: 02nd of April 2013, 59 minutes after 13.*" in the beginning of a run. NB: The part that does the printout needs to be updated after the year 2099 due to '20' being printed as a fixed string.
- timest(r1) [13 lines]: Start time.
- timex(r1) [10 lines]: Calculate elapsed time. Used together with `timest(r1)` in the `program maincr` to calculate the elapsed time for preparating calculations, tracking computations and total time used for the simulations.
- abend(cstring) [12 lines]: Prints out an error message of the form "*SIXTRACK STOP/ABEND*" followed by the input string `cstring`. This is then followed by a Fortran **stop** statement, which terminates the run. Called from various other subroutines.

sixvefox.f

This module contains differential algrebra (D.A) versions of some subroutines found in `sixve.f`. It uses the D.A package in `dabnews.f` to a great extent. The main purpose of these subroutines is in the calculation of the 4D/6D closed orbit using the D.A approach.

Closed Orbit

- envada [1421 lines]: Calculation of momentum depending element matrices and change of path length for each particle. Used for **thick elements**. Specially prepared for new D.A (six-dimensional version)
- envquad(i,ipch) [471 lines]: Calculation of momentum depending element matrices and change of path length for each particle. Used for **thick elements**. Specially prepared for new D.A (six-dimensional version)
- umlauda [6245 lines]: Central loop for 6D closed orbit calculation.
- clorda(nn,idummy,am) [423 lines]: Calculation of the 6D closed orbit

- mydaini(ncase,nnord,nnvar,nndim,nnvar2,nnord1) [86 lines]: Calculation of the 4D closed orbit including delta.
- synoda [247 lines]: Synchrotron oscillations, specially prepared for new D.A

Misc.

- wireda [946 lines]: This subroutine sends a particle with coordinates (x,a,y,b,d) through the map of a straight current wire.
- errff(xx,yy,wx,wy) [456 lines]: Modification of the wwerf subroutine, this version is for map production using Berz's D.A package.

Beam-Beam

- beaminf(track,param,sigzs,bcu,ibb,ne,ibbc [89 lines]: Hirata's 6D beam-beam from BBC. DA version.
- boostf(sphi,cphi,tphi,salpha,calpha,track) [256 lines]: ???
- sbcf(star,cphi,nsli,f,ibb,bcu,track,ibbc) [542 lines]: ???
- boostif(sphi,cphi,tphi,salpha,calpha,track) [310 lines]: ???
- bbff(sepx,sepy,sigxx,sigyy,bbfx,bbfy,bbgx,bbgy) [369 lines]: ???

-- DavidSinuela - 11-Sep-2012 -- RiccardoDeMaria - 26-Sep-2012 -- AlessioMereghetti - 17-Nov-2012

SixTrackSubCollimat

===+ Collmation Routines

```
mynp= nloop samples of napx particles).
Loop over particle sample (j=1,int(mynp/napx00))
call...

Thin6D routine
+ Check if FIRSTRUN
 read collimator database
 initialize random number generator
 generate random tilt/offsets
 + Cycle over elements (j=1 .. iu)
  + Check if the element is a collimator
   recognize an element as a collimator from the element name beginning bez(myix)(1:2) or bez(myi
   recognize the collimator type from the element name bez(myix)(9:11) or bez(myix)(8:10)
   assign to each element its aperture as defined in the fort.3 file
   + cycle over collimators in DB (i=1 .. dbncoll)
     if the element(j) corresponds to the collimator(j) in the database -- db_name(i)=bez(myix)
     look for the minimum collimator gap
    + Check if pencil beam
      set the pencil beam to start form the minimum-gap collimator
 re-initialize random generator (call ranxlugo)
 initialize some particle arrays (secondary, tertiary...)
 initialize global efficiency arrays neff(k) neffx(k) neffy(k)
 initialize some collimator arrays (?) cn_impact(j) cn_absorbed(j) csum(j) csqsum(j)

+ Cycle over particles (j=1,napx)
 initialize some arrays (secondary, tertiary...)
 initialize particle name ipart(), flukaname(j)

+ Cycle over number of turns (do 660 n=1,numl)
 zero counter for efficiency calculations (totals=0)
 if(irip.eq.1) call ripple(n) (??)
 if(mod(numx,nwri).eq.0) call writebin(nthinerr) (??)
 if(nthinerr.ne.0) return (??)
 + Cycle over n. of elemenst (do 650 i=1,iu)
  + Cycle over particles (j=1,napx)
   if particles are absorbed ((part_abs(j).gt.0), or the coordinates are large enough (100m, 100m
   put to zero each coordinate
  if (firstrun) save the coordinates of the particle 1 to xbob, ybob, xpbob, ypbob to check the o
  + some "sixtack stuff" not understood
  assign myktrack=1 to any element whose name correspond to a colimator-type element (check bez(m
  ONLY if the element is a collimator (myktrack=1), I continue to label 10... (goto(10,30,740,650
  define stracki as drift lenght
  + If I have a collimator (a.k.a Label 10)
   + Check if collimation is on AND if the element is a collimator from the element name beginnin
    recognize the collimator type from the element name bez(myix)(9:11) or bez(myix)(8:10)
    assign to each element its aperture as defined in the fort.3 file
    set collimator lenght to zero (c_lenght=0)
    + Check if first run
     + Check if 0 <rselect < 64
      + Cycle over number of particles (j = 1, napx)
       Transform particle coordinates
       xj = (xv(1,j)-torbx(ie))/1d3;
       xpj = (yv(1,j)-torbxp(ie))/1d3;
       yj = (xv(2,j)-torby(ie))/1d3;
       ypj = (yv(2,j)-torbyp(ie))/1d3;
       pj = ejv(j)/1d3)
       then, for each element, the sum of normalized amplitued for each particle is calculated.
    Look for adeguate DB information, i.e.:
    + Cycle over collimator in the database
```

```
 if the element(j) corresponds to the collimator(j) in the database, then the variable FOUND=
 the number of the associated collimator in the database is called ICOLL
+ If the collimator is in the database
 If indicated in the fort.3 (do_nsig=false), assign apertures from the database
 define some parameters for the beta-beating studies
 If indicated in the fort.3 (do_nominal) use the beta functions in the colliamtor database
 inluding beta beating and calculate final beta functions at the collimator bx_dist,by_dist
 If indicated in the fort.3 (do_write_dist) and the collimator is the selected one, write col
 If first turn, write some output (colldb)
 + If the collimator is NOT for RHIC
  store all the characteristics defined in the database (c_lenght, c_tilt, c_material, c_offs
  additional calculation for the crystal (more parameters must be defined)
  calculate all different apertures (nominal aperture, calc_aperture, pencil aperture)
  calculate x,xp,y,yp for pencil beam
  in case the pencil beam is generated at the seleceted colliamtor, change the collimator til
 + ELSE If the collimator for RHIC
  store all the characteristics defined in the database (c_lenght, c_tilt, c_material, c_offs
  crystals are NOT defined for RHIC
  aperture calculations are different
 If first turn, write some more output (collgaps)
 Define the full aperture in meters c_aperture=2*calc_aperture
 + Cycle over number of particles(j = 1, napx)
  Define the particle coordinates! (rcx,rcxp,rcy,rcyp,rcp)
  Check if the drift lenght is zero (if not ABORT)
  drift back of half collimator lenght
  give flukaname (flukaname(j) = ipart(j)+100*samplenumber)
 if indicated in fort.3 (do_oneside) OR if it a crystal/romanpot/scraper/tcdq, set the collim
 + If the collimator is in the database
  if the collimator is RHIC collimator: call collimate_RHIC
  elseif the collimator is crystal collimator: call collimate_CRY
  elseif the collimator is an electron lens: call collimate_ELENS
  elseif the collimator is sliced: divide in slices, and call collimate2 for each slice
  else: call collimate2
 + Cycle over particles (do j = 1, napx)
  + If particle has hit the collimator((part_hit(j).eq.(100000000*ie+iturn)))
   if the original drift lenght was =0, track back of half lenght
   if the collimator is a crystal, and the variable write_c_out is true (hardcoded!), write c
   if the collimator is an elens, and the variable write_elens_out is true (hardcoded!), writ
   copy data back to original vector (from rc coordinates to xv coord.)
   Update of the energy variables to adapt to a possible energy change ejfv,rvv,dpsv,oidpsv,d
  Else get back the original particle coordinates
  + If first run
   + Check if 0 <rselect < 64 (part_abs(j).eq.0)
     If the first particle at the first turn, set sum_ax and sum_ay to zero
    + If the particle has not been absorbed
     calculate normalized coordinates and amplitudes          Update the calues of the variab
    Else set the particle normalized amplitudes to zero
    sampl(ie) = totals
    ename(ie) = bez(myix)(1:16)
  + If tha particle has just hit a collimator
   + If indicated in the fort.3 (dowrite impacts)
    write output file 46 all_impacts.dat
   + If the particle has NOT been absorbed
    + If indicated in the fort.3 (dowrite impacts)
     write output file 47 all_absorptions.dat
    write output file 38 tracks2.dat
   + If the particle has NOT been absorbed
    Update the variables for secondary, tertiary...
   + If indicated in the fort.3 (dowritetracks)
    + If the particle has NOT been absorbed
     + If the particle is in tertiary or secondary halo AND its displacemente is lower than 9
       write the output file 38 tracks2.dat both before and after the collimator.
   + If the particle impact is < 0.9 (??)
    update the impact variable n_impact,sqsum,cn_impact,csum,csqsum
   + If the particle has been absorbed
    update the absorbed variables n_absorbed,cn_absorbed,n_tot_absorbed,iturn_last_hit,iturn_
 + If there has been at least one impact
```

```
      calculate impacts average and sigma
    + If there has been at least one impact on icoll
      calculate impacts average and sigma
    + If the collimator is the selected one(db_name1(icoll)(1:10).eq.name_sel(1:10) )and it is t
      initialize some variables
     + Cycle over particles(j = 1, napx)
      + If the particle has just hit the collimator
       Update variable num_selhit,num_selabs
     Initialize variables (n_impact,sum, sqsum)
     + Cycle over particles(j = 1, napx)
      + If the particle has just hit the collimator
       Update variables (n_impact,sum, sqsum)
       calculate number of hits n_impact
        If selected in the fort.3 (dowrite_impact) write output file 49 (impact.dat)
     + If there has been at least one impact
       calculate impacts average and sigma
     print out some data about selected collimator
  +Else (if not a collimator or collimation is off)
   Consider the element as a drift
  GOTO 650
```

This topic: LHCAtHome > SixTrackSubCollimat
Topic revision: r2 - 14-Dec-2012 - ValentinaPrevitali

SixTrackSubadia

The input parameters are

- `numx`, the current revolution number of the simulation
- `e0f`, the momentum of the reference particle [MeV/c]

Basic structure

```
subroutine adia(numx, e0f)
  ! Declaration of variables
  ...

  ! If a certain number of turns at the flat bottom
  ! is (nde(1)) or for energy ramping (nde(2)) is set
  ! then the phase is set to zero and the subroutine exited
  if(numx.eq.1) phas0 = phas
  if(numx.le.nde(1)) phas = zero
  if(numx.le.nde(1)) return
  if(numx.le.nde(2)) phas = zero
  if(numx.le.nde(2)) return

  ! Set synchrotron phase to phase0
  ! Calculate new reference energy and reference momentum
  ! (also see below)
  phas = phas0
  e0 = e0 + hsy(1)*sin_rn(phas)
  e0f = sqrt(e0**2-pma**2)

  return
end
```

$E_0 \rightarrow E_0 + V \cdot \sin(\phi)$

$P_0 = \sqrt{E^2 - m_p^2}$

This topic: LHCAtHome > SixTrackSubadia
Topic revision: r1 - 05-Apr-2013 - MattiasFjellstrom

SixTrackSubadib

The input parameter is

- `e0f`, the momentum of the reference particle [MeV/c]

Basic structure

```
subroutine adib(e0f)
  ! Variable declarations

  ! Check if phas0 is less than 1d-38
  if(abs(phas0).le.pieni) return

  ! Calculate new reference energy
  ! and reference momentum
  e0 = e0 + hsy(1)*sin_rn(phas)
  e0f = sqrt(e0**2 - pma**2)
  return
end
```

$E_0 \rightarrow E_0 + V \cdot \sin(\phi)$

$P_0 = \sqrt{E_0^2 - m_p^2}$

This topic: LHCAtHome > SixTrackSubadib
Topic revision: r1 - 05-Apr-2013 - MattiasFjellstrom

SixTrackSubclorb

The basic structure/algorithm for calculating the closed orbit is described below. [1]

Algorithm

The following quantities will be used in the description:

$\mathbf{M}$ is the one-turn transfer matrix.
$\mathbf{I}$ is theidentity matrix.
$\vec{z}$ either stands for $\vec{x} = (x_{co}, x'_{co})$ or $\vec{y} = (y_{co}, y'_{co})$.
$\frac{\Delta p}{p}$ is the momentum deviation.
$\vec{D}$ is the dispersion vector.

(1) The calculation is done iteratively. An initial guess for the closed orbit is done (as the *off-momentum closed orbit*)

$\vec{z}_0 = \frac{\Delta p}{p}\vec{D}$

(2) Using the following relations

$\vec{z}_0 + \Delta\vec{z} = \mathbf{M}(\vec{z}_0 + \Delta\vec{z})$
$\vec{z}_1 = \mathbf{M}\vec{z}_0$

the deviation from the closed orbit is calculated as

$\Delta\vec{z} = [\mathbf{M} - \mathbf{I}]^{-1}(\vec{z}_0 - \vec{z}_1)$

(3) The new value of $\vec{z}$ is then calculated as $\vec{z}_{new} = \vec{z}_{old} + \Delta\vec{z}$, the process is then repeated.

When the deviation $\Delta\vec{z}$ is within the desired precision of the closed orbit calculation specified in the iteration error input block (ITER) the iteration terminates.

[1] *RACETRACK: A computer code for the simulation of nonlinear particle motion in accelerators*, http://frs.web.cern.ch/frs/report/race.pdf

Code structure

Set the initial guess of clo(l) and clop(l)

```
do 10 l=1,2
  clo(l) = dpp*di0(l)
  clop(l) = dpp*dip0(l)
  dx(l) = 1e6
  dy(l) = 1e6
10 continue
```

Construct the one turn matrix and peform the one-turn-transformation (UMLAUF)

```
call envar(dpp)
call umlauf(dpp, 1, ierr)
ierro = ierr
if(ierro.ne.0) return
```

Loop over the number of iterations for the closed orbit calculation (itco)

```
do 40 ii=1,itco
```

Check if the current guess is within desired limits. The limits are dma and dmap, specified in the ITER input-block

```
  dcx = abs(dx(1))
  dcxp = abs(dy(1))
  dcz = abs(dx(2))
  dczp = abs(dy(2))
  if(dcx.le.dma.and.dcz.le.dma.and.dczp.le.dmap.and.dczp.le.dmap) goto 50
```

Set (x,y) to (clo,clop) and save old values to (x0,y0)

```
  do 20 l=1,2
     x(1,l) = clo(l)
     y(1,l) = clop(l)
     x0(l) = x(1,l)
  20 y0(l) = y(1,l)
```

Update the one-turn matrix and do the one-turn-transformation (a call to umlauf is done inside of the matrix-subroutine)

```
  call matrix(dpp, am)
  if (ierro.ne.0) return
```

Calculate the difference with the old values (i.e dx,dy) and calculate the new values of (clo,clop)

```
 do 30 l=1,2
    ll = 2*l
    x1(l) = x(1,l)
    y1(l) = y(1,l)
    det = two - am(ll-1,ll-1) - am(ll,ll)
    dx(l) = x0(l) - x1(l)
    dy(l) = y0(l) - y1(l)
    dclo(l) = (dx(l)*(am(ll,ll)-one) - dy(l)*am(ll-1,ll)) / det
    dclop(l) = (dy(l)*(am(ll-1,ll-1)-one) - dy(l)*am(ll-1,ll)) / det
    clo(l) = clo(l) + dclo(l)
    clop(l) = clop(l) + dclop(l)
  30 continue
40 continue

if (ncorru.ne.1) write(*,10000) itco

50 cor = c1e3*sqrt(dcx**2 + dcz**2)

if (iout.eq.1.and.ncorru.ne.1) then
  write(*,10010) dpp, clo(1), clop(1), clo(2), clop(2), ii, cor
endif

return
end subroutine
```

This topic: LHCAtHome > SixTrackSubclorb
Topic revision: r3 - 16-Apr-2013 - MattiasFjellstrom

SixTrackSubcomnul

As a reference list, here are the variables that are set to zero. The array variables are of different size depending on if they are related to a particle or an element (or something else)

```
ncorru, ncorrep, nrturn, ithick, ierro, il, iclo6, iclo6r, mper, mblo,
mbloz, kanf, iu, itra, napx, numl, numlr, ird, imc, niu(1), niu(2), idp,
irew, iorg, itco, itcro, itqv, ichrom, iqmod, iqmod6, ilin, iqmodc,
ichromc, ilinc, ntco, nt, iprint, iclo, icoe, ise, mesa, mp, m21, m22,
m23, ise1, ise2, ise3, isub, nta, nte, ipt, irmod2, nre, nur, nch, nqc,
npp, ipos, iconv, imad, nstart, nstop, iskip, iav, iwg, ivox, ivoz, ires,
ifh, idis, icow, istw, iffw, irip, irco, idial, nord, nvar, nvar2, ndimf,
nordf, nvarf, nord1, nsix, nvar2, ncor, idptr, nbeam, ibb6d, ibeco,
ibtyp, lhc, ibbc, iver, ibidu, inorm, imod1, imod2, icorr, nctype, namp,
nmom, nmom1, nmom2, weig1, weig2, dpmax, dpda_da, dpda1_da, sigmda_da,
ej1_da, ejf1_da, rv_da, pi, pi2, pisqrt, rad, chi0, chid, dp1, idfor,
rat, qs, e0, crad, dppoff, tlen, pma, phas0, phas, ition, dpscor, sigcor,
benki, dma, dmap, dkq, dqq, de0, ded, dsi, dech, dsm0, amp0, qxt, qzt,
eui, euii, tam1, tam2, totl, dphix, dphiz, qx0, qz0, dres, dfft, preda,
partnum, emitx, emity, emitz, gammar, sigz, sige, damp, ampt, tlim,
time0, time1, nde(i), is(i), idz(i), amp(i), bet0(i), alf0(i), clo(i),
clop(i), aper(i), di0(i), dip0(i), cro(i), sigma0(i), qwsk(i), betx(i),
betz(i), alfx(i), alfz(i), iq(i), hsy(i), qw0(i), clo6(i), clop6(i),
clon(i), wxys(i), corr(i,i1), corr(1,1), corr(1,2), nwr(i), ipr(i),
nrr(i), nu(i), nskew(i), dtr(i), ire(i), msym(i), ta(i,j), exz(i,j),
rtc(i1,i2,i3,i4), rts(i1,i2,i3,i4), tasau(i,i1,i2), kz(i), kp(i), irm(i),
imtr(i), nmu(i), kpa(i), isea(i), nrel(i), ncororb(i), iratioe(i),
itionc(i), dki(i,1), dki(i,2), dki(i,3), ed(i), el(i), ek(i), sm(i),
xpl(i), xrms(i), zpl(i), zrms(i), benkc(i), r00(i), apx(i), apz(i),
ape(1,i), ape(2,i), ape(3,i), ramp(i), rfre(i), rzph(i), ratioe(i),
hsyc(i), phasc(i), ptnfac(i), wirel(i), acdipph(i), crabph(i),
crabph2(i), crabph3(i), crabph4(i), a(i,i3,i4), al(i4,i3,i1,i),
as(i4,i3,i1,i), bk0(i,i1), ak0(i,i1), bka(i,i1), aka(i,i1), parbe(i,i1),
mel(i), mstr(i), elbe(i), bl1(i,i1,i2), bl2(i,i1,i2), mtyp(i,j), ic(i),
mzu(i), icext(i), icextal(i), extalign(i,1), extalign(i,2),
extalign(i,3), sigmoff(i), tiltc(i), tilts(i), imbb(i), xsi(i), zsi(i),
smi(i), smizf(i), rsmi(i), rfres(i), rzphs(i), aai(i,i1), bbi(i,i1),
aaiv(i3,i2,i), bbiv(i3,i2,i), zfz(i), rvf(i), sigm(i), dps(i), ej(i),
ejf(i), x(i,i1), y(i,i1), icomb0(i1), icomb(i,i1), ratio(i,i1), hmal(i),
cotr(i,i1), rrtr(i,i1,i2), sigman(j,i), sigman2(j,i), sigmanq(j,i),
clobeam(j,i), beamoff(j,i), bbcu(i,j), bbcu(i,11), xx_da(i1), yy_da(i1),
alda_da(i1,i2), asda_da(i1,i2), aldaq_da(i1,i2), asdaq_da(i1,i2),
smida_da(i)
```

This topic: LHCAtHome > SixTrackSubcomnul
Topic revision: r2 - 08-Apr-2013 - MattiasFjellstrom

SixTrackSubdaten

daten

Read input data from `fort.2` and `fort.3`

After the declaration of all the variables and commons, all the variables are initialised. Afterwards, the `fort.3` file is parsed: every keyword has its own dedicated lines, reached by `GO TO` statements. In the particular case the `GEOM` keyword is found, the `fort.2` file is parsed. Once the `fort.3` file is fully parsed and the keyword `ENDE` is found, some initialisation is performed. A summary printout can be requested. `FORMAT` statements close the subroutine declaration.

```
      subroutine daten

c     declaration of variables and commons;

c     initialisation of variables, among which:
c     - line number when reading multi-line junk:
      iclr=0

c     understand if the geometry description of the accelerator
c       is contained in fort.2 or fort.3

  110 read(3,10000,end=1530,iostat=ierro) idat
c     dedicated parts of code for every info not related to geometry
      if(idat(1:1).eq.'/') goto 110
      if(idat.eq.sing) goto 120
      if(idat.eq.bloc) goto 190
      if(idat.eq.stru) goto 320
      if(idat.eq.prin) goto 550
      if(idat.eq.disp) goto 170
      if(idat.eq.tune) goto 600
      if(idat.eq.sync) goto 710
      if(idat.eq.iter) goto 940
      if(idat.eq.fluc) goto 790
      if(idat.eq.mult) goto 740
      if(idat.eq.chro) goto 560
      if(idat.eq.trac) goto 510
      if(idat.eq.diff) goto 520
      if(idat.eq.line) goto 660
      if(idat.eq.limi) goto 950
      if(idat.eq.orbi) goto 980
      if(idat.eq.init) goto 500
      if(idat.eq.comb) goto 1030
      if(idat.eq.subr) goto 1110
      if(idat.eq.reso) goto 1120
      if(idat.eq.sear) goto 1200
      if(idat.eq.orga) goto 880
      if(idat.eq.post) goto 1280
      if(idat.eq.ripp) goto 1290
      if(idat.eq.deco) goto 1320
      if(idat.eq.comm) goto 1390
      if(idat.eq.norm) goto 1400
      if(idat.eq.corr) goto 1410
      if(idat.eq.beam) goto 1600
      if(idat.eq.trom) goto 1700
!GRD
      if(idat.eq.coll) goto 1285
!GRD
      if(idat.eq.fluk) goto 1800
```

```
      if(idat.eq.dist) goto 1900
      if(idat.eq.next) goto 110
      if(idat.eq.ende) goto 771
      call prror(15)

!-----------------------------------------------------------------------
!  DATENBLOCK SINGLE ELEMENTS
!  ELLEMENTLISTE
!-----------------------------------------------------------------------
  120 i=1

c     fort.2 or fort.3, the current line is stored in ch
c     Nota Bene: in case the declaration of SINGLE ELEMENTs is finished,
c                these lines instruct the code what to do next
  130 if(imod.eq.1) then
c        [...]
      endif

c     actually parse the line:
      call intepr(1,1,ch,ch1)
c     store the most important values:
      read(ch1,*) idat,kz(i),ed(i),ek(i),el(i),bbbx(i),bbby(i),bbbs(i)
c     for special values of kz(i), further dedicated instructions are performed
c     Nota Bene: it's actually a series of if-endif pieces of code;
      if(kz(i).eq.25) then
c        [...]
      endif

c     thick or thin lens model?
      if(abs(el(i)).gt.pieni.and.kz(i).ne.0) ithick=1

c     store the name of current SINGLE ELEMENT
      bez(i)=idat

c     go to next line
      goto 130

!-----------------------------------------------------------------------
!  DATENBLOCK DISPLACEMENT OF ELEMENTS
!-----------------------------------------------------------------------
  170 read(3,10020,end=1530,iostat=ierro) ch
c     Nota Bene: GOTO 110 statements are under IF conditionals;

c     variable initialisation of xpl0,xrms0,zpl0,zrms0

c     actually parse the line:
      call intepr(1,1,ch,ch1)
c     acquire the most important values:
      read(ch1,*) idat,xpl0,xrms0,zpl0,zrms0

c     values are stored in xpl(j), xrms(j), zpl(j), zrms(j);
c     in case of AC dipole (type=+/-16), these four numbers
c       are actually stored in nturn1(j), nturn2(j), nturn3(j), nturn4(j),
c       because they have a different meaning;
c       j is the index of the SINGLE ELEMENT;

c     go to next line
      goto 170

!-----------------------------------------------------------------------
!  BLOCK DEFINITIONS
!-----------------------------------------------------------------------
c     treat the first line of the BLOCK declaration (mper, msym(i))
  190 if(imod.eq.1) then
c        [...]
      endif
```

```
      i=0
  220 do 230 m=1,40
  230 ilm0(m)=idum

c     fort.2 or fort.3, the current line is stored in ch
c     Nota Bene: in case the declaration of BLOCKs is finished,
c                these lines instruct the code what to do next
      if(imod.eq.1) then
c        [...]
      endif

c     actually parse the line:
      call intepr(2,1,ch,ch1)
      read(ch1,*) idat,(ilm0(m),m=1,40)

      if(idat.eq.idum) goto 270
c     new BLOCK
      i=i+1
      if(i.gt.nblo-1) call prror(18)
      bezb(i)=idat
      k0=0
      mblo=i

c     acquire useful information:
  270 ka=k0+1
      ke=k0+40
      do 300 l=ka,ke
        if(l.gt.nelb) call prror(26)
        ilm(l)=ilm0(l-k0)
        if(ilm(l).eq.idum) goto 310
        mel(i)=l
        beze(i,l)=ilm(l)
        do 280 j=1,il
          if(bez0(j).eq.ilm(l)) goto 290
  280   continue
        erbez=ilm(l)
        call prror(19)
  290   mtyp(i,l)=j
        if(kz(j).ne.8) elbe(i)=elbe(i)+el(j)
  300 continue
  310 k0=l-1
      goto 220

!-----------------------------------------------------------------------
!  STRUCTURE INPUT
!-----------------------------------------------------------------------
  320 i=0
  330 do 340 k=1,40
  340 ilm0(k)=idum

c     fort.2 or fort.3, the current line is stored in ch
c     Nota Bene: in case the declaration of BLOCKs is finished,
c                these lines instruct the code what to do next
      if(imod.eq.1) then
c        [...]
      endif

      i2=1
c     read possible repetition parts
      do 420 ii=1,80
  420 continue

c     actual interpretation of the line
  430 call intepr(3,i2,ch,ch1)
      read(ch1,*) (ilm0(k),k=1,40)

c     acquire useful information:
```

```
      do 490 k=1,40
        if(ilm0(k).eq.idum) goto 490
        if(ilm0(k).eq.go) goto 480
        i=i+1
        do 440 j=1,mblo
          if(bezb(j).eq.ilm0(k)) goto 470
  440   continue
        do 450 l=1,il
          if(bez0(l).eq.ilm0(k)) goto 460
  450   continue
        erbez=ilm0(k)
        call prror(20)
  460   continue
        ic(i)=l+nblo
        if(bez0(l).eq.cavi) icy=icy+1
        goto 490
  470   ic(i)=j
        goto 490
  480   kanf=i+1
  490 continue
      mbloz=i
      if(mbloz.gt.nblz-2) call prror(21)
      goto 330

!-----------------------------------------------------------------------
!  INITIAL COORDINATES
!-----------------------------------------------------------------------
  500 read(3,10020,end=1530,iostat=ierro) ch
c     the data lines are stored as described in the manual
c       (see keyword INIT); in particular:
c     - exz(1,j=1,6): infos about particle #1 (data line 2-7);
c     - exz(2,j=1,6): infos about particle #2 (data line 8-13);
c     - e0, ej(1), ej(2): last three data lines (respectively);

      iclr=0
      nbidu=1
      goto 110

!-----------------------------------------------------------------------
!  TRACKING PARAMETERS
!-----------------------------------------------------------------------
  510 read(3,10020,end=1530,iostat=ierro) ch
c     the data lines are stored as described in the manual
c       (see keyword TRAC)

      iclr=0
      nbidu=1
      goto 110

!-----------------------------------------------------------------------
!  DIFFERENTIAL ALGEBRA
!-----------------------------------------------------------------------
  520 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
c     Nota Bene: GOTO 110 statements are under IF conditionals;

!-----------------------------------------------------------------------
!  PRINTOUT INPUT PARAMETERS
!-----------------------------------------------------------------------
  550 iout=1
      goto 110

!-----------------------------------------------------------------------
!  CHROMATCITY ADJUSTMENT
!-----------------------------------------------------------------------
  560 ichrom=1
c     missing comments
```

```
      goto 110

!---------------------------------------------------------------------
!  TUNE ADJUSTMENT
!---------------------------------------------------------------------
  600 iqmod=1
c     missing comments
c     Nota Bene: GOTO 110 statements are under IF conditionals;

!---------------------------------------------------------------------
!  LINEAR OPTICS CALCULATION
!---------------------------------------------------------------------
 660  continue
c     missing comments
c     Nota Bene: GOTO 110 statements are under IF conditionals;

!---------------------------------------------------------------------
!  SYNCHROTRON OSCILLATIONS
!---------------------------------------------------------------------
  710 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
      goto 110

!---------------------------------------------------------------------
!  MULTIPOLE COEFFICIENTS  FOR KZ = 11
!---------------------------------------------------------------------
  740 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
c     Nota Bene: GOTO 110 statements are under IF conditionals;

!---------------------------------------------------------------------
!  FLUCTUATION RANDOM STARTING NUMBER
!---------------------------------------------------------------------
  790 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
      goto 110
  870 call prror(80)

!---------------------------------------------------------------------
!  ORGANISATION OF RANDOM NUMBERS
!---------------------------------------------------------------------
  880 write(*,10130)
c     missing comments
      goto 110

!---------------------------------------------------------------------
!  ITERATION ERRORS FOR CLOSED ORBIT ,TUNE ADJUSTMENT AND CHROMATICITY
!---------------------------------------------------------------------
  940 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments

      iclr=0
      goto 110

!---------------------------------------------------------------------
!  APERTURE LIMITATIONS
!---------------------------------------------------------------------
  950 write(*,10320)

c     each line of this declaration part is parsed, and data are stored:
c       - xaper, yaper in apx(j), apz(j);
c       - yaper**2, xaper**2, xaper**2*yaper**2 in ape(1,j), ape(2,j), ape(3,j);
c       respectively (see SixTrack Manual, keyword LIMI, for the meaning
c       of xaper,yaper);
c       j is the index of the SINGLE ELEMENT;
      goto 110
```

```
!-----------------------------------------------------------------------
!  ORBIT CORRECTION
!-----------------------------------------------------------------------
  980 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
c     Nota Bene: GOTO 110 statements are under IF conditionals;

!-----------------------------------------------------------------------
!  COMBINATION OF ELEMENTS
!-----------------------------------------------------------------------
 1030 ii=0
c     missing comments
      goto 110

!-----------------------------------------------------------------------
!  SUBRESONANCE CALCULATION
!-----------------------------------------------------------------------
 1110 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
      isub=1
      goto 110

!-----------------------------------------------------------------------
!  RESONANCE-COMPENSATION
!-----------------------------------------------------------------------
 1120 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
      irmod2=1
      goto 110

!-----------------------------------------------------------------------
!  SEARCH FOR OPTIMUM PLACES TO COMPENSATE RESONANCES
!-----------------------------------------------------------------------
 1200 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
c     Nota Bene: GOTO 110 statements are under IF conditionals;

!-----------------------------------------------------------------------
!  POSTPROCESSING
!-----------------------------------------------------------------------
 1280 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
      ipos=1
      goto 110
!-----------------------------------------------------------------------
!  POWER SUPPLY RIPPLE
!-----------------------------------------------------------------------
 1290 irip=1
 1300 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
c     Nota Bene: GOTO 110 statements are under IF conditionals;

!-----------------------------------------------------------------------
!  DECOUPLING ROUTINE
!-----------------------------------------------------------------------
 1320 iskew=1
c     missing comments
      goto 110

!GRD-----------------------------------------------------------------------
!  COLLIMATION INPUT BLOCK
!GRD-----------------------------------------------------------------------
 1285 read(3,10020,end=1530,iostat=ierro) ch

c     it dumps an error message, it skips the concerned lines, and
c       go back to 110
```

```
!-----------------------------------------------------------------------
!  COMMENT LINE
!-----------------------------------------------------------------------
 1390 read(3,10020,end=1530,iostat=ierro) commen
c     no need of comments
      goto 110

!-----------------------------------------------------------------------
!  NORMAL FORMS
!-----------------------------------------------------------------------
 1400 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
c     Nota Bene: GOTO 110 statements are under IF conditionals;

!-----------------------------------------------------------------------
!  TUNESHIFT CORRECTIONS
!-----------------------------------------------------------------------
 1410 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
c     Nota Bene: GOTO 110 statements are under IF conditionals;

!-----------------------------------------------------------------------
!  Beam-Beam Element
!-----------------------------------------------------------------------
 1600 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
c     Nota Bene: GOTO 110 statements are under IF conditionals;

!-----------------------------------------------------------------------
!  TROMBONE ELEMENT KZ=22
!-----------------------------------------------------------------------
 1700 read(3,10020,end=1530,iostat=ierro) ch
c     missing comments
c     Nota Bene: GOTO 110 statements are under IF conditionals;

!-----------------------------------------------------------------------
!  FLUKA COUPLING
!-----------------------------------------------------------------------
 1800 read(3,10020,end=1530,iostat=ierro) ch
c     activate the coupling to Fluka, and its debug (in case);
c     boolean flags fluka_enable and fluka_debug are used;
c     Nota Bene: GOTO 110 statements are under IF conditionals;

!-----------------------------------------------------------------------
!  Initial DISTribution generation
!-----------------------------------------------------------------------
 1900 read(3,10020,end=1530,iostat=ierro) ch
c     store the filename with the distribution in variable beam_filename;
c     Nota Bene: GOTO 110 statements are under IF conditionals;

!-----------------------------------------------------------------------
c     perform some further initialisation:
c     - for beam-beam effects;
  771 if(napx.ge.1) then
c       [...]
      endif

c     - for elements with kz(i)=15 (??);
      do j=1,il
         if(kz(j).eq.15) then
         endif
      enddo
      if(iout.eq.0) return

c     if requested, dump input data:
      write(*,10050)
c     - SINGLE ELEMENTs data:
```

```
      write(*,10060)
      il1=il
      if(ncy2.eq.0) il1=il-1
      do 1435 k=1,il1
      if(abs(kz(k)).eq.12) then
        write(*,10070) k,bez(k),kz(k),ed(k),ek(k),phasc(k),xpl(k),      &
     &xrms(k),zpl(k),zrms(k)
        kz(k)=abs(kz(k))
        phasc(k)=phasc(k)*rad
      else
        write(*,10070) k,bez(k),kz(k),ed(k),ek(k),el(k),xpl(k),xrms(k), &
     &zpl(k),zrms(k)
      endif
 1435 continue
      write(*,10130)
c     - BLOCs data:
      write(*,10080)
      write(*,10090) mper,(msym(k),k=1,mper)
      write(*,10250) mblo,mbloz
      write(*,10100)
      do 1450 l=1,mblo
      kk=mel(l)
      ll=kk/6
      if(ll.ne.0) then
        do 1440 l1=1,ll
          l2=(l1-1)*6+1
          l3=l2+5
          if(l2.eq.1) then
            write(*,10260) l,bezb(l),kk,(beze(l,k),k=1,6)
          else
            write(*,10270) (beze(l,k),k=l2,l3)
          endif
 1440   continue
        if(mod(kk,6).ne.0) then
          l4=ll*6+1
          write(*,10270) (beze(l,k),k=l4,kk)
        endif
      else
        write(*,10260) l,bezb(l),kk,(beze(l,k),k=1,kk)
      endif
 1450 continue
c     - STRUCTURE:
      write(*,10120)
      mblozz=mbloz/5+1
      do 1480 k=1,mblozz
      k10=(k-1)*5
      if((mbloz-k10).eq.0) goto 1480
      do 1470 l=1,5
        if((k10+l).gt.mbloz) ic0(l)=' '
        if((k10+l).gt.mbloz) goto 1470
        icc=ic(k10+l)
        if(icc.gt.nblo) goto 1460
        ic0(l)=bezb(icc)
        goto 1470
 1460   ic0(l)=bez0(icc-nblo)
 1470 continue
      k11=k10+1
      write(*,10280) k11,(ic0(l),l=1,5)
 1480 continue
      write(*,10130)
 1490 if(idp.eq.0) goto 1500
c     - additional print-out;
c       missing comments
!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
      return
```

```
c     FORMAT statements

      end
```

This topic: LHCAtHome > SixTrackSubdaten
Topic revision: r1 - 27-Nov-2012 - DavidSinuela

SixTrackSubenvar

This subroutine filles the matrix a(i,l,ll) which is used for closed orbit and optics calculations.
Notice that this subroutine does not include any longitudinal coordinates.

The basic structure of the subroutine is as follows

```
! Set dpd (delta + one) and the square root of this term
! Used for computation of the elements of a(i,l,ll)
dpd = one+dpp
dpsq = sqrt(dpd)

! do-loop for every element in the single element list
do 200 i=1,il
    ! set the a(i,l,ll) elements to zero
    do ll=1,6
        do l=1,2
            a(i,l,ll) = zero
        enddo
    enddo

    ! check if length < 1e-38
    ! this is treated as a non-linear insertion
    if(abs(el(i)).le.pieni) goto 190

    kz1 = kz(i) + 1

    ! now follows a number of goto-statements depending on
    ! the type of the current element
    ! the included elements are
    ! Drift
    ! Exact drift (this is currently being implemented, June 2013)
    ! Rectangular dipole magnet (horizontal, vertical)
    ! Sektor dipole magnet (horizontal, vertical)
    ! Quadrupole (focusing, defocusing)
    ! Combined function magnet (horizontal, vertical, focusing, defocusing)
    ! Edge focusing
    ! Non-linear insertion

    ! For the non-linear insertion the sm(i) is set
    sm(i) = ed(i)

    ! The last thing that is done is a call to the subroutine block
    call block
```

This topic: LHCAtHome > SixTrackSubenvar
Topic revision: r1 - 17-Jun-2013 - MattiasFjellstrom

SixTrackSubprogram

An overview description of the program flow in program maincr (not exhaustive)

(1) Declaration of varibles and commons

(2) Opening all datafiles (`fort.2`, ..., `fort.98`)

(3) Prints start-up message with current date and time

(4) First a number of variables related to the structure of elements, blocks and all tracked particles are set to zero inside the maincr code. Then a call is made to `subroutine comnul`, which sets all common variables to zero.

(5) Call to `subroutine daten`, which reads all input files.

(6) Sets up variables related to plotting.

(7) Looong do-loop (**todo: describe everything that happens inside of this loop**)

(8) If `ibidu` is set to 1 in the TRAC input block, then a dump of the whole accelerator description is done to unit 32. If ibidu is set to 2, then the whole accelerator description is read in from unit 32.

(9) Check if `idfor` is equal to 2, in which case initial coordinates for tracking is read in from unit 13. Otherwise the initial coordinates are taken from fort.3. A check is also done to see if the closed orbit should be added to the initial coordinates or not.

(10) ...

(11) Tracking. A call is made to either of the subroutines `trauthin` och trauthck in track.f

(12) Print outs depending on if particles were lost or not.

(13) Close all data files.

This topic: LHCAtHome > SixTrackSubprogram
Topic revision: r1 - 17-Apr-2013 - MattiasFjellstrom

SixTrackSubthck4d

thck4d

This subroutine is responsible for tracking in 4d with thick lenses. It is called when the `SYNC` input option is not present in the `fort.3` file, i.e. whenever no synchrotron motion is requested.

Structure:

```
      subroutine thck4d(nthinerr)
      * there is some declaration of variables and commons;
      * there is some initialization of variables;
      *     in particular, idz1 and idz2, which trigger
      *     the addition of the dispersion contribution to the
      *     transverse coordinates;
      * cycle on revolutions in the forward direction;
      do 490 n=1,numl

         * cycle on BLOCs/non-linear SINGLE ELEMENTs (i.e. OUTSIDE BLOCs)
         do 480 i=1,iu

            * check if the current thing is a BLOC (ktrack(i)==1)
            *       or a non-linear SINGLE ELEMENT (i.e. outside any BLOC):
            if(ktrack(i).eq.1) then
              * BLOC
              ix=ic(i)
            else
              * non-linear, SINGLE ELEMENT outside any BLOC
              ix=ic(i)-nblo
            endif

            * call to FLUKA
            * NB: the FLUKA element MUST be a SINGLE ELEMENT
            *     OUTSIDE ANY BLOCK!!
            if(ktrack(i).ne.1 .and. fluka_element.eq.ix) goto 755

            * decide what to do according to the value of ktrack(i)
            * NB: ktrack(i)==1 (thus a block) sends the flow to line 20
            *     which is just below this point, and implements the
            *     algebra of bl1v matrix;
            goto(20,470,740,470,470,470,470,470,470,470,40,60,80,100,   &
     &120,140,160,180,200,220,270,290,310,330,350,370,390,410,          &
     &430,450,470,240,500,520,540,560,580,600,620,640,680,700,720,      &
     &470,748,470,470,470,470,470,745,746,751,752,753,754),ktrack(i)
            [...]
            * after each labeled block of instructions there's the following command,
            *       bringing the flow to the part about the aperture check:
            goto 470

  755       continue
            * actual call to FLUKA + related lines
            [...]
            goto 470
  470       continue
            * aperture check:
            kapez=abs(kape(ix))
            if(kapez.eq.2) then
               ! Rectanble
               call lostpar3(i,ix,nthinerr)
               if(nthinerr.ne.0) return
            elseif(kapez.eq.3) then
```

```
               ! Ellipse
               call lostpar4(i,ix,nthinerr)
               if(nthinerr.ne.0) return
            else
               call lostpar2(i,ix,nthinerr)
               if(nthinerr.ne.0) return
            endif
480      continue
490   continue
      return
      end
```

This topic: LHCAtHome > SixTrackSubthck4d
Topic revision: r1 - 27-Nov-2012 - RiccardoDeMaria

SixTrackSubtrauthck

trauthck

This subroutine is responsible for deciding if the tracking is to be performed in 4D or 6D. It is called when the lattice contains thick elements.

Structure:

```
subroutine trauthck(nthinerr)
*declarations of common variables, common blocks and parameters

do 5 i=1,npart
  nlostp(i) = i
5 continue

do 10 i=1,nblz
  ktrack(i) = 0
  strack(i) = zero
  strackc(i) = zero
  stracks(i) = zero
10 continue

* beam-beam element
if(nbeam.ge.1) then
  * left to do: write what this if-statement does
endif

* continues with a do-loop containing the bulk of the subroutine
do 290 i=1,iu
  * left to do: explain what happens in the main do-loop
290 continue

do 300 j=1,napx
  dpsv1(j) = (dpsv(j)*c1e3)/(one+dpsv(j))
300 continue

* Then comes the part where either 4D or 6D tracking is called

if (nwri.eq.0) nwri = num1+numlr+1

* idp is a switch for synchrotron motion (0: off, 1: on)
* ition is the transition energy switch defined in the SYNC input block
if (idp.eq.0.or.ition.eq.0) then
  call thck4d(nthinerr)
else
  hsy(3) = (c1m3*hsy(3))*dble(ition)
  do 310 jj=1,nele
    if (kz(jj).eq.12) hsyc(jj)=(c1m3*hsyc(jj))*dble(itionc(jj))
  310 continue

  * phas: synchrotron phase
  * if |phas| > 0: 6D-tracking with acceleration
  * if |phas| = 0: 6D-tracking without acceleration
  if (abs(phas).ge.pieni) then
    call thck6dua(nthinerr)
  else
    call thck6d(nthinerr)
  endif
endif
```

```
return
end
```

This topic: LHCAtHome > SixTrackSubtrauthck
Topic revision: r2 - 08-Apr-2013 - MattiasFjellstrom

SixTrackSubwritelin

The input arguments are

- `nr` the number of the element or block in the accelerator lattice where the linear parameters are to be printed
- `typ` names of the elements or blocks where the linear parameters are to be printed
- `tl` the accumulated total length from START to the element
- `p1` the phase advance phi (multiples of two pi)
- `t` matrix containing information about the linear parameters alpha, beta and gamma (for both horizontal and vertical planes), it also has information about the dispersion and the closed orbit
- `ixwl` related to special correction calculations for sextupoles and octupoles (normal and skew)

Basic structure:

```
subroutine writelin(nr,typ,tl,p1,t,ixwl)
! Declarations of common variables
...

! Check if the element is the START location
! the parameter istart is never ever used anywhere else
istart = 0
if(typ.eq.'START') istart = 1

! the linear parameters are printed at nlin elements
! if it is set to 0, the parameters are printed at each element
! iwrite is a switch to perform the printing
iwrite = 0
if(nlin.eq.0) then
  iwrite = 1
else
  ! check if linear parameters are to be printed at the current element

if(iwrite.eq.1) then
  ! perform the calculation of the linear parameters
  ...


if(ncorru.eq.0) then
  ! if there are no correctors present (ORBI input block)
  ! perform the printing

else
  ! calculate corrections due to
  ! sextupoles and octupoles
```

This topic: LHCAtHome > SixTrackSubwritelin
Topic revision: r1 - 05-Apr-2013 - MattiasFjellstrom

SixTrackMinutes

SixTrack Minutes

25-Jan-2013

Present: Eric, Laurent, Riccardo, Igor, Nils, Pete.

Maxim and Kai added to boinc:user Riccardo and Laurent added to boinc:admins

In the AFS buffer e.g. /afs/cern.ch/user/b/boinc/scratch0/boinc two studies having the same workspace and jobname from two different users can collide. In the unlikely event an error is issued to warn the users therefore the collision will not be silent.

User should use the new workspace space (up to 100GB) that can be obtained by the CERN accounts . Boinc04 can be used for testing and developing for:

- checkpoint and restart improvements (to allow volunteers to receives shorter jobs),
- retrieve more output files (maybe using the zip libraries linked in the boinc client),
- know what are the limits of the system in terms AFS robustness.

Riccardo and Laurent will be added to user able to acces the boinc server installation in /data/boinc/project/sixtrack.

Boinc server use a local mysql server, we could use in the future the DB ondemand service from IT if some technical issues are solved (DB engine and server port).

Virtual Machine clients is under slow but steady development but is more involved for the user. Test4Theory is using this but users have some troubles.

Version 444.6 version is updated in boinc04 aka boinctest.

Outstanding issues: review the checkpoint and restart counter.

-- RiccardoDeMaria - 25-Jan-2013

This topic: LHCAtHome > SixTrackMinutes
Topic revision: r4 - 28-Jan-2013 - LaurentDeniau

SixTrackRoadMap

RoadMap

This is a tentative roadmap for the next development. People and time estimates are indicative.

Physics:

- Exact Hamiltonian for
    - ♦ Drift and thin bends [Mattias, April 2013]
    - ♦ Thick bends [1 month]
- Per-particle mass and charge state [Mattias, June 2013]
- Track total time and total path length [2 weeks]
- Nonlinear fringe fields [Barbara, June 2013]
- Generic Taylor maps [Dave]
- Extend turn (and time) dependent functions for strengths [Alessio, David]
- Extend rf multipoles (any order including tilt and misalignment) [Mattias, August 2013]
- Extend aperture model (racetrack and generic polygon) [1 week]

Numerics

- maintain numerical reproducibility [Eric]
- evaluate 10^7 turns numerical stability [Eric, Massimo]

Documentation

- physics manual [Mattias, Riccardo March 2013]
- improve sixdesk manual [Riccardo]

Massive tracking

- submit to BOINC and LSF portion of tracking jobs.
    - ♦ check if the state save is platform independent [1 week]
    - ♦ or make it so (hdf5) [3 weeks]
    - ♦ update the boinc api to send the complete state (either zip file or hdf5) [1 week] *update the boinc server and sixdesk logic [2 weeks]
- increase the check frequency of checkpoint and restart [1 day]
- SixDesk:
    - ♦ reduce external dependencies [3 days]
    - ♦ reduce disk I/O

Release management:

- consolidate build system (Makefile or CMake) for all platform with and without boinc [Laurent, Riccardo, February 2013]
- introduce tests in the build system for all platforms [Laurent, Riccardo February 2013]
- develop a nightly build system (CTest?) [Laurent, Riccardo 1 week]

This topic: LHCAtHome > SixTrackRoadMap
Topic revision: r3 - 01-Feb-2013 - RiccardoDeMaria

SixTrackSource

SixTrack Source Code organization

The sixtrack distribution contains source for several executable:

- `sixtrack`: tracking program linked with `track.o sixve.o sixvefox.o dabnews.o lielib.o` [`beamgas.o boinc crlibm myhdf5lib`]
- `sixtrack_da`: Differantial Alagebra (DA) version of sixtrack linked with `sixda.o sixsc.o sixscfox.o dabnew.o lielib.o`
- dafo: preprocessor for generating DA formulas
- astuce: fortran preprocessor

and several object files: boinc_api_fortran, `myboinc`

Sixtrack is compiled by `make_six`

```
./make_six [-]option_name ...
```

and a number of options that:

- modifies ast files by turning on or off certain flags and decks;
- calls `astuce` which generates fortran files starting from source files with the same name;
- calls the compilers with appropriate options.

`make_six` options:

- junk: for Eric specific debuging
- tilt
- tracking
- fast
- crlibm
- api
- cernlib
- naglib
- da
- collimat
- cpss
- boinc
- cr
- nagfor
- g77
- g95
- gfortran
- bpm
- beamgas
- bnlelens
- bignblz
- debug
- hdf5

- ifort
- fio
- SSE
- lf95

Astuce

Fortran files are generated by astuce from .s files trough .ast files with flags.

An ast file has the structure:

```
output_file.f
df <flag1>,...    set flags
e  <deck>         specify deck
....
ex terminate
```

For instance:

```
sixtrack.s
trackn.f
df vvector,crlibm,tilt,fast,collimat,cr,boinc,bpm,beamgas,bnlelens,bignblz,debug,hdf5,fio,ifort
e tra_thck
e tra_thin
e nwrtcoll
e nwrtbnl
ex
```

The syntax is the following:

```
+cd code block (implicitely terminated)
+if flag start if clause
+ei end if clause
+ca add block
+dk define deck (implicitely terminated)
```

A flag can be composed (e.g. collimat.and..not.bnlelen)

Ast Files

Specifc for `sixtrack` executable:

- sixve.ast
    - ♦ maincr: main program
- track.ast: main tracking loop
    - ♦ tra_thck
    - ♦ tra_thin
    - ♦ nwrtcoll
    - ♦ nwrtbnl
- sixvefox.ast
- dabnews.ast

Specific for `sixtrack_da` executable:

- sixda.ast

        - ♦ mainda: main program
    - sixsc.ast
    - sixscfox.ast
    - dabnew.ast

For both sixtrack:

    - lielib.ast
    - beamgas.ast

Source files

    - dabnew.s
    - lielib.s
    - sixtrack.s

Flags

    - automatc
    - beamgas
    - big
    - bignblz
    - bnlelens
    - boinc
    - bpm
    - collimat
    - cr
    - crlibm
    - ctrack
    - debug
    - fast
    - fio
    - hdf5
    - hhp
    - iibm
    - nagfor
    - rvet
    - small
    - tilt
    - time
    - vvector

-- RiccardoDeMaria - 26-Sep-2012

This topic: LHCAtHome > SixTrackSource
Topic revision: r6 - 22-Oct-2012 - RiccardoDeMaria