Radia Interface Specification

Generated by Doxygen 1.8.15

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Chapter 2

File Documentation

2.1 radentry.h File Reference

Macros

- #define EXP
- #define CALL
- #define OK 0

Functions

- EXP const char ∗CALL RadErrGet (int er)
- EXP int CALL RadErrGetSize (int ∗siz, int er)
- EXP int CALL RadErrGetText (char ∗t, int er)
- EXP const char ∗CALL RadWarGet (int er)
- EXP int CALL RadWarGetSize (int ∗siz, int er)
- EXP int CALL RadWarGetText (char ∗t, int er)
- EXP int CALL RadObjRecMag (int ∗n, double ∗P, double ∗L, double ∗M)
- EXP int CALL RadObjThckPgn (int ∗n, double xc, double lx, double ∗FlatVert, int nv, char a, double ∗M)
- EXP int CALL RadObjPolyhdr (int ∗n, double ∗FlatVert, int nv, int ∗FlatFaces, int ∗FacesLen, int nf, double ∗M, double ∗M_LinCoef, double ∗J, double ∗J_LinCoef)
- EXP int CALL RadObjArcPgnMag (int ∗n, double ∗P, char a, double ∗FlatVert, int nv, double ∗Phi, int nseg, char sym_no, double ∗M)
- EXP int CALL RadObjMltExtPgn (int ∗n, double ∗FlatVert, int ∗SlicesLen, double ∗Attitudes, int ns, double ∗M)
- EXP int CALL RadObjMltExtRtg (int ∗n, double ∗FlatCenPts, double ∗FlatRtgSizes, int ns, double ∗M)
- EXP int CALL RadObjMltExtTri (int ∗n, double xc, double lx, double ∗FlatVert, double ∗FlatSubd, int nv, char a, double ∗M, char ∗opt)
- EXP int CALL RadObjCylMag (int ∗n, double ∗P, double r, double h, int nseg, char a, double ∗M)
- EXP int CALL RadObjRecCur (int ∗n, double ∗P, double ∗L, double ∗J)
- EXP int CALL RadObjArcCur (int ∗n, double ∗P, double ∗R, double ∗Phi, double h, int nseg, char man_auto, char a, double j)
- EXP int CALL RadObjRaceTrk (int ∗n, double ∗P, double ∗R, double ∗L, double h, int nseg, char man_auto, char a, double j)
- EXP int CALL RadObjFlmCur (int ∗n, double ∗FlatPts, int np, double i)
- EXP int CALL RadObjScaleCur (int n, double scaleCoef)
- EXP int CALL RadObjBckg (int ∗n, double ∗B)

- EXP int CALL RadObjCnt (int ∗n, int ∗Objs, int nobj)
- EXP int CALL RadObjAddToCnt (int cnt, int ∗Objs, int nobj)
- EXP int CALL RadObjCntSize (int ∗n, int cnt)
- EXP int CALL RadObjCntStuf (int ∗Objs, int cnt)
- EXP int CALL RadObjDpl (int ∗n, int obj, char ∗opt)
- EXP int CALL RadObjM (double ∗M, int ∗arMesh, int obj)
- EXP int CALL RadObjCenFld (double ∗B, int ∗arMesh, int obj, char type)
- EXP int CALL RadObjSetM (int obj, double ∗M)
- EXP int CALL RadObjCutMag (int ∗Objs, int ∗nobj, int obj, double ∗P, double ∗N, char ∗opt)
- EXP int CALL RadObjDivMagPln (int ∗n, int obj, double ∗SbdPar, int nSbdPar, double ∗FlatNorm, char ∗opt)
- EXP int CALL RadObjDivMagCyl (int ∗n, int obj, double ∗SbdPar, int nSbdPar, double ∗FlatCylPar, double rat, char ∗opt)
- EXP int CALL RadObjGeoVol (double ∗v, int obj)
- EXP int CALL RadObjGeoLim (double ∗L, int obj)
- EXP int CALL RadObjDegFre (int ∗num, int obj)
- EXP int CALL RadObjDrwQD3D (int obj, char ∗opt)
- EXP int CALL RadObjDrwOpenGL (int obj, char ∗opt)
- EXP int CALL RadObjDrwAtr (int obj, double ∗RGB, double thcn)
- EXP int CALL RadObjFullMag (int ∗n, double ∗P, double ∗L, double ∗M, double ∗K, int nK, int grp, int mat, double ∗RGB)
- EXP int CALL RadTrfPlSym (int ∗trf, double ∗P, double ∗N)
- EXP int CALL RadTrfRot (int ∗trf, double ∗P, double ∗V, double phi)
- EXP int CALL RadTrfTrsl (int ∗trf, double ∗V)
- EXP int CALL RadTrfInv (int ∗trf)
- EXP int CALL RadTrfCmbL (int ∗fintrf, int origtrf, int trf)
- EXP int CALL RadTrfCmbR (int ∗fintrf, int origtrf, int trf)
- EXP int CALL RadTrfMlt (int ∗objout, int obj, int trf, int mlt)
- EXP int CALL RadTrfOrnt (int ∗objout, int obj, int trf)
- EXP int CALL RadTrfZerPara (int ∗objout, int obj, double ∗P, double ∗N)
- EXP int CALL RadTrfZerPerp (int ∗objout, int obj, double ∗P, double ∗N)
- EXP int CALL RadMatApl (int ∗objout, int obj, int mat)
- EXP int CALL RadMatLin (int ∗mat, double ∗Ksi, double ∗Mr, int nMr)
- EXP int CALL RadMatStd (int ∗mat, char ∗id, double m)
- EXP int CALL RadMatMvsH (double ∗M, int ∗nM, int obj, char ∗id, double ∗H)
- EXP int CALL RadMatSatIsoFrm (int ∗mat, double ∗KsiMs1, double ∗KsiMs2, double ∗KsiMs3)
- EXP int CALL RadMatSatIsoTab (int ∗mat, double ∗MatData, int np)
- EXP int CALL RadMatSatLamFrm (int ∗mat, double ∗KsiMs1, double ∗KsiMs2, double ∗KsiMs3, double p, double ∗N)
- EXP int CALL RadMatSatLamTab (int ∗n, double ∗MatData, int np, double p, double ∗N)
- EXP int CALL RadMatSatAniso (int ∗mat, double ∗DataPar, int nDataPar, double ∗DataPer, int nDataPer)
- EXP int CALL RadRlxPre (int ∗n, int obj, int srcobj)
- EXP int CALL RadRlxMan (double ∗D, int ∗n, int intrc, int meth, int iter, double rlxpar)
- EXP int CALL RadRlxAuto (double ∗D, int ∗n, int intrc, double prec, int iter, int meth, const char ∗opt)
- EXP int CALL RadRlxUpdSrc (int intrc)
- EXP int CALL RadSolve (double ∗D, int ∗n, int obj, double prec, int iter, int meth)
- EXP int CALL RadFld (double ∗B, int ∗nB, int obj, char ∗id, double ∗Coords, int np)
- EXP int CALL RadFldCmpCrt (int ∗n, double prcB, double prcA, double prcBInt, double prcFrc, double prc↩ TrjCrd, double prcTrjAng)
- EXP int CALL RadFldCmpPrc (int ∗n, char ∗opt)
- EXP int CALL RadFldUnits (char ∗OutStr)
- EXP int CALL RadFldUnitsSize (int ∗size)
- EXP int CALL RadFldLenRndSw (int ∗n, char ∗OnOrOff)
- EXP int CALL RadFldLenTol (int ∗n, double AbsVal, double RelVal, double ZeroVal)
- EXP int CALL RadFldEnr (double ∗d, int objdst, int objsrc, int ∗SbdPar)
- EXP int CALL RadFldEnrFrc (double ∗f, int ∗nf, int objdst, int objsrc, char ∗id, int ∗SbdPar)

- EXP int CALL RadFldEnrTrq (double ∗f, int ∗nf, int objdst, int objsrc, char ∗id, double ∗P, int ∗SbdPar)
- EXP int CALL RadFldFrc (double ∗f, int ∗nf, int obj, int shape)
- EXP int CALL RadFldFocPot (double ∗d, int obj, double ∗P1, double ∗P2, int np)
- EXP int CALL RadFldInt (double ∗f, int ∗nf, int obj, char ∗InfOrFin, char ∗id, double ∗P1, double ∗P2)
- EXP int CALL RadFldLst (double ∗B, int ∗nB, int obj, char ∗id, double ∗P1, double ∗P2, int np, char ∗Arg↩ OrNoArg, double start)
- EXP int CALL RadFldPtcTrj (double ∗f, int ∗nf, int obj, double E, double ∗InitCond, double ∗LongLim, int np)
- EXP int CALL RadFldFocKickPer (double ∗M1, double ∗M2, double ∗IntBtrE2, double ∗Arg1, double ∗Arg2, int ∗size, int obj, double ∗P1, double ∗Ns, double per, int nper, int nps, double ∗Ntr, double r1, int np1, double d1, double r2, int np2, double d2, int nh, char ∗com)
- EXP int CALL RadFldFocKickPerFormStr (char ∗str, double ∗M1, double ∗M2, double ∗IntBtrE2, double ∗Arg1, double ∗Arg2, int np1, int np2, double per, int nper, char ∗com)
- EXP int CALL RadFldShimSig (double ∗f, int ∗nf, int obj, char ∗id, double ∗V, double ∗P1, double ∗P2, int np, double ∗Vi)
- EXP int CALL RadFldFrcShpRtg (int ∗n, double ∗P, double ∗W)
- EXP int CALL RadUtiDel (int ∗n, int obj)
- EXP int CALL RadUtiDelAll (int ∗n)
- EXP int CALL RadUtiDmp (char ∗OutStr, int ∗pSize, int ∗arObj, int nObj, char ∗AscOrBin)
- EXP int CALL RadUtiDmpPrs (int ∗arElem, int ∗nElem, unsigned char ∗sBytes, int nBytes)
- EXP int CALL RadUtiIntrptTim (double ∗d, double t)
- EXP int CALL RadUtiDataGet (char ∗pcData, const char typeData[3], long key=0)
- EXP int CALL RadUtiVer (double ∗d)
- EXP int CALL RadUtiYeldFuncSet (int(∗pExtFunc)())

2.1.1 Macro Definition Documentation

2.1.1.1 CALL

```
#define CALL
```

2.1.1.2 EXP

```
#define EXP
```

2.1.1.3 OK

```
#define OK 0
```

2.1.2 Function Documentation

2.1.2.1 RadErrGet()

```
EXP const char∗ CALL RadErrGet (
              int er )
```

Returns the error message associated to error number. This function cannot be called from Visual Basic For Applications.

Parameters

er	[in] error number

Returns

the chain of characters representing the error string

Author

P. Elleaume

Version

1.0

See also

GetErrorSize and GetErrorText

2.1.2.2 RadErrGetSize()

```
EXP int CALL RadErrGetSize (
              int * siz,
              int er )
```

Returns the length of the error message string not counting "\0".

Parameters

siz	[out] length of the error message
er	[in] error number

Returns

integer error code (0 : No Error, $>$0 : Error Number, $<$0 : Warning Number)

Author

P. Elleaume

Version

1.0

2.1.2.3 RadErrGetText()

```
EXP int CALL RadErrGetText (
             char * t,
             int er )
```

Returns the text of error message associated to error number.

Parameters

t	[out] error message string
er	[in] error number

Returns

integer error code (0 : No Error, >0 : Error Number, <0 : Warning Number)

Author

P. Elleaume

Version

1.0

2.1.2.4 RadFld()

```
EXP int CALL RadFld (
             double * B,
             int * nB,
             int obj,
             char * id,
             double * Coords,
             int np )
```

Computes magnetic field created by the object obj at one or many points.

Parameters

B	[out] flat array of all computed values of the magnetic field components (should be allocated by calling function)
nB	[out] total number of calculated magnetic field component values
obj	[in] reference number of the magnetic field source object
id	[in] string identifying magnetic field components to be computed
Coords	[in] flat array of coordinates of all points where the field should be computed (x1,y1,z1,x2,y2,z2,...)
np	[in] number of points where the magnetic field should be calculated (the length of the array Coords is equal to 3∗np)

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.5 RadFldCmpCrt()

```
EXP int CALL RadFldCmpCrt (
             int * n,
             double prcB,
             double prcA,
             double prcBInt,
             double prcFrc,
             double prcTrjCrd,
             double prcTrjAng )
```

Sets general absolute accuracy levels for computation of magnetic field induction (prcB), vector potential (prcA), induction integrals along straight line (prcBint), field force (prcFrc), relativistic particle trajectory coordinates (prc←TrjCrd) and angles (prcTrjAng).

Parameters

n	[out] dummy
prcB	[in] absolute accuracy level for magnetic field induction [T]
prcA	[in] absolute accuracy level for vector potential
prcBInt	[in] absolute accuracy level for magnetic field induction integrals along straight line [T∗mm]
prcFrc	[in] absolute accuracy level for the force [N]
prcTrjCrd	[in] absolute accuracy level for particle trajectory coordinate
prcTrjAng	[in] absolute accuracy level for particle trajectory angle

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.6 RadFldCmpPrc()

```
EXP int CALL RadFldCmpPrc (
             int * n,
             char * opt )
```

Sets general absolute accuracy levels for computation of magnetic field induction (PrcB), vector potential (PrcA), induction integral along straight line (PrcBInt), field force (PrcForce), torque (PrcTorque), energy (PrcEnergy); relativistic charged particle trajectory coordinates (PrcCoord) and angles (PrcAngle). The function works according to the mechanism of string options. The name(s) of the option(s) should be: PrcB, PrcA, PrcBInt, PrcForce, PrcTorque, PrcEnergy, PrcCoord, PrcAngle.

Parameters

n	[out] dummy
opt	[in] pointer to an option string, which can be "PrcB->..." or "PrcA->..." or "PrcBInt->..." or "PrcForce->..." or "PrcTorque->..." or "PrcEnergy->..." or "PrcCoord->..." or "PrcAngle->...", where "..." should be replaced by the appropriate absolute accuracy level.

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.7 RadFldEnr()

```
EXP int CALL RadFldEnr (
              double * d,
              int objdst,
              int objsrc,
              int * SbdPar )
```

Computes potential energy (in Joule) of the object objdst in the field created by the object objsrc. If SbdPar = 0, the function performes the computation based on absolute accuracy value for the energy (by default 10 Joule; can be modified by the function radFldCmpPrc). Otherwise, the computation is performed based on the destination object subdivision numbers (kx=SbdPar[0],ky=SbdPar[1],kz=SbdPar[2]).

Parameters

d	[out] computed energy value
objdst	[in] reference number of the object, the energy of which should be computed
objsrc	[in] reference number of the source object creating the magnetic field
SbdPar	[in] array of 3 integer numbers specifying subdivision parameters for the energy computation (kx=SbdPar[0],ky=SbdPar[1],kz=SbdPar[2]). If SbdPar = 0, the function performes the computation based on absolute accuracy value for the energy (by default 10 Joule, can be modified by the function radFldCmpPrc).

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.8 RadFldEnrFrc()

```
EXP int CALL RadFldEnrFrc (
             double * f,
             int * nf,
             int objdst,
             int objsrc,
             char * id,
             int * SbdPar )
```

Computes force (in Newton) acting on the object objdst in the field produced by the object objsrc. If SbdPar = 0, the function performes the computation based on absolute accuracy value for the force (by default 10 Newton; can be modified by the function radFldCmpPrc). Otherwise, the computation is performed based on the destination object subdivision numbers {kx=SbdPar[0],ky=SbdPar[1],kz=SbdPar[2]}.

Parameters

f	[out] computed force component(s)
nf	[out] number of force components computed
objdst	[in] reference number of the object, on which the force is acting
objsrc	[in] reference number of the source object creating the magnetic field
id	[in] string identifying the force components to be computed (fx|fy|fz)
SbdPar	[in] array of 3 integer numbers specifying subdivision parameters for the energy/force computation (kx=SbdPar[0],ky=SbdPar[1],kz=SbdPar[2]). If SbdPar = 0, the function performes the computation based on absolute accuracy value for the force (by default 10 Newton, can be modified by the function radFldCmpPrc).

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.9 RadFldEnrTrq()

```
EXP int CALL RadFldEnrTrq (
             double * f,
             int * nf,
             int objdst,
             int objsrc,
             char * id,
             double * P,
             int * SbdPar )
```

Computes torque (in Newton∗mm) with respect to point P, acting on the object objdst in the field produced by the object objsrc. If SbdPar = 0, the function performes the computation based on absolute accuracy value for the torque (by default 10 Newton∗mm; can be modified by the function radFldCmpPrc). Otherwise, the computation is performed based on the destination object subdivision numbers {kx=SbdPar[0],ky=SbdPar[1],kz=SbdPar[2]}.

Parameters

f	[out] computed torque component(s)
nf	[out] number of torque components computed
objdst	[in] reference number of the object on which the force is acting
objsrc	[in] reference number of the source object creating the magnetic field
id	[in] string identifying the force components to be computed (tx|ty|tz)
P	[in] array of 3 real numbers specifying cartesian coordinates of the point with respect to which the torque should be computed
SbdPar	[in] array of 3 integer numbers specifying subdivision parameters for the energy/force computation (kx=SbdPar[0],ky=SbdPar[1],kz=SbdPar[2]). If SbdPar = 0, the function performes the computation based on absolute accuracy value for the torque (by default 10 Newton∗mm, can be modified by the function radFldCmpPrc).

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Author

O.C.

2.1.2.10 RadFldFocKickPer()

```
EXP int CALL RadFldFocKickPer (
            double * M1,
            double * M2,
            double * IntBtrE2,
            double * Arg1,
            double * Arg2,
            int * size,
            int obj,
            double * P1,
            double * Ns,
            double per,
            int nper,
            int nps,
            double * Ntr,
            double r1,
            int np1,
            double d1,
            double r2,
            int np2,
            double d2,
            int nh,
            char * com )
```

Computes matrices of 2nd order kicks for trajectory of relativistic charged particle in periodic magnetic field produced by the object obj. The computed kick matrices can be used in charged particle tracking codes. The longitudinal integration along one period starts at point P1 and is done along direction pointed by vector Ns; one direction of the transverse grid is pointed by vector Ntr, the other transverse direction is given by vector product of Ntr and Ns.

Parameters

M1	[out] flat array of real numbers representing the matrix of kick values in the first transverse direction given by the Ntr vector
M2	[out] flat array of real numbers representing the matrix of kick values in the second transverse direction given by vector product of Ntr and Ns
IntBtrE2	[out] flat array of real numbers representing the matrix of longitudinally-integrated squared transverse magnetic field
Arg1	[out] array of position values in the first transverse direction where kick matrix was computed
Arg2	[out] array of position values in the second transverse direction where kick matrix was computed
size	[out] length of formatted string containing the computed results; such string can be prepared by the function RadFldFocKickPerFormStr
obj	[in] reference number of the object creating the magnetic field
P1	[in] array of 3 real numbers specifying cartesian coordinates of the integration start point
Ns	[in] array of 3 real numbers specifying cartesian coordinates of a vector defining the longitudinal direction for the periodic magnetic field
per	[in] period length
nper	[in] number of full periods
nps	[in] number of longitudinal points
Ntr	[in] array of 3 real numbers specifying cartesian coordinates of a vector defining first transverse direction
r1	[in] range of the transverse grid along direction given by Ntr vector
np1	[in] number of points in the transverse grid along direction given by Ntr vector
d1	[in] step to calculate derivative along direction given by Ntr vector; d1=0 means that the step for derivative calculation is equal to the corresponding step of the transverse grid where kick matrix is computed
r2	[in] range of the transverse grid along direction given by vector product of Ntr and Ns
np2	[in] number of points in the transverse grid along direction given by vector product of Ntr and Ns
d2	[in] step to calculate derivative along direction given by vector product of Ntr and Ns; d2=0 means that the step for derivative calculation is equal to the corresponding step of the transverse grid where kick matrix is computed
nh	[in] maximum number of magnetic field harmonics to take into account
com	[in] arbitrary string comment

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Authors

: P.E., O.C.

2.1.2.11 RadFldFocKickPerFormStr()

```
EXP int CALL RadFldFocKickPerFormStr (
             char * str,
             double * M1,
             double * M2,
             double * IntBtrE2,
             double * Arg1,
             double * Arg2,
             int np1,
             int np2,
             double per,
             int nper,
             char * com )
```

Prepares a formatted string containing second-order kick matrices for trajectory of relativistic charged particle in periodic magnetic field.

Parameters

str	[out] C-string containing second-order kick matrices values in the format compatible with particle tracking computer codes (e.g. BETA, TRACY)
M1	[in] flat array of real numbers representing the matrix of kick values in one transverse direction
M2	[in] flat array of real numbers representing the matrix of kick values in another transverse direction
IntBtrE2	[in] flat array of real numbers representing the matrix of longitudinally-integrated squared transverse magnetic field
Arg1	[in] array of position values in the first transverse direction where kick matrix was computed
Arg2	[in] array of position values in the second transverse direction where kick matrix was computed
np1	[in] number of points in the transverse grid along the first transverse direction
np2	[in] number of points in the transverse grid along the second transverse direction
per	[in] period length
nper	[in] number of full periods
com	[in] arbitrary string comment that will be included into the formatted string str

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Authors

: O.C.

2.1.2.12 RadFldFocPot()

```
EXP int CALL RadFldFocPot (
             double * d,
             int obj,
             double * P1,
             double * P2,
             int np )
```

Computes "focusing potential" for trajectory of relativistic charged particle in magnetic field produced by the object obj. The integration is made from P1 to P2 with np equidistant points.

Parameters

d	[out] computed focusing potential value
obj	[in] reference number of the magnetic field source object
P1	[in] array of 3 real numbers specifying cartesian coordinates of an edge point of the integration segment
P2	[in] array of 3 real numbers specifying cartesian coordinates of an edge point of the integration segment
np	[in] number of points for the integration

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.13 RadFldFrc()

```
EXP int CALL RadFldFrc (
              double * f,
              int * nf,
              int obj,
              int shape )
```

Computes force of the field produced by the object obj into a shape defined by shape. shape can be the result of RadObjRecMag (parallelepiped) or RadFldFrcShpRtg (rectangular surface). This function uses the algorithm based on Maxwell tensor, which may not always provide high efficiency. We suggest to use the function RadFldEnrFrc instead of this function.

Parameters

f	[out] computed force component(s)
nf	[out] number of force components computed
obj	[in] reference number of the magnetic field source object
shape	[in] reference number of the shape object

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.14 RadFldFrcShpRtg()

```
EXP int CALL RadFldFrcShpRtg (
              int * n,
              double * P,
              double * W )
```

Creates a rectangle with central point P and dimensions W (to be used for force computation via Maxwell tensor).

Parameters

n	[out] reference number of the object created
P	[in] array of 3 real numbers specifying cartesian coordinates of the center point
W	[in] array of 2 real numbers specifying dimensions of the rectangle

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Author

O.C.

2.1.2.15 RadFldInt()

```
EXP int CALL RadFldInt (
              double * f,
              int * nf,
              int obj,
              char * InfOrFin,
              char * id,
              double * P1,
              double * P2 )
```

Computes magnetic field integral produced by the object obj along a straight line specified by points P1 and P2; depending on the InfOrFin variable value, the integral is infinite ("inf") or finite ("fin"), from P1 to P2; the field integral component is specified by the id input variable. The unit is T$*$mm.

Parameters

f	[out] computed field integral component(s)
nf	[out] number of field integral components computed
obj	[in] reference number of the object creating the magnetic field
InfOrFin	[in] string specifying the type of field integral: finite ("fin") or infinite ("inf")
id	[in] string identifying the field integral components to be computed (ibx|iby|ibz)
P1	[in] array of 3 real numbers specifying cartesian coordinates of a point on the integration line
P2	[in] array of 3 real numbers specifying cartesian coordinates of another point on the integration line

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.16 RadFldLenRndSw()

```
EXP int CALL RadFldLenRndSw (
             int * n,
             char * OnOrOff )
```

Switches on or off the randomization of all the length values. The randomization magnitude can be set by the function radFldLenTol.

Parameters

n	[out] dummy
OnOrOff	[in] string containing either "on" or "off"

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.17 RadFldLenTol()

```
EXP int CALL RadFldLenTol (
             int * n,
             double AbsVal,
             double RelVal,
             double ZeroVal )
```

Sets absolute and relative randomization magnitudes for all the length values, including coordinates and dimensions of the objects producing magnetic field, and coordinates of points where the field is computed. Optimal values of the variables can be: RelVal=10^(-11), AbsVal=L*RelVal, ZeroVal=AbsVal, where L is the distance scale value (in mm) for the problem to be solved. Too small randomization magnitudes can result in run-time code errors.

Parameters

n	[out] dummy
AbsVal	[in] absolute position/length randomization magnitude [mm]
RelVal	[in] relative position/length randomization magnitude
ZeroVal	[in] absolute zero tolerance [mm]

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Author

O.C.

2.1.2.18 RadFldLst()

```
EXP int CALL RadFldLst (
             double * B,
             int * nB,
             int obj,
             char * id,
             double * P1,
             double * P2,
             int np,
             char * ArgOrNoArg,
             double start )
```

Computes magnetic field created by object obj in np equidistant points along a line segment from P1 to P2; the field component is specified by the id input variable.

Parameters

B	[out] computed field value(s)
nB	[out] number of field values computed
obj	[in] reference number of the object creating the magnetic field
id	[in] string identifying magnetic field components to be computed
P1	[in] array of 3 real numbers specifying cartesian coordinates of an edge point of the line segment
P2	[in] array of 3 real numbers specifying cartesian coordinates of another edge point of the line segment
np	[in] number of points where the magnetic field should be calculated
ArgOrNoArg	[in] string specifying whether or not to output a longitudinal position for each point where the field is computed ("arg|noarg")
start	[in] start value for the longitudinal position

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Author

O.C.

2.1.2.19 RadFldPtcTrj()

```
EXP int CALL RadFldPtcTrj (
             double * f,
             int * nf,
             int obj,
             double E,
             double * InitCond,
             double * LongLim,
             int np )
```

Computes transverse coordinates and their derivatives (angles) of a relativistic charged trajectory in the magnetic field produced by object obj, using a Runge-Kutta integration. The charge of the particle is that of electron. All positions are in millimeters and angles in radians.

Parameters

f	[out] computed tragectory parameters
nf	[out] number of tragectory parameters computed
obj	[in] reference number of the magnetic field source object
E	[in] particle energy [GeV]
InitCond	[in] array of 4 real numbers specifying initial transverse coordinates and angles of the trajectory (x0,dxdy0,z0,dzdy0, y is longitudinal coordinate)
LongLim	[in] array of 2 real numbers specifying limits of the longitudinal position (y1,y2)
np	[in] number of points where the trajectory should be calculated

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Author

O.C.

2.1.2.20 RadFldShimSig()

```
EXP int CALL RadFldShimSig (
             double * f,
             int * nf,
             int obj,
             char * id,
             double * V,
             double * P1,
             double * P2,
             int np,
             double * Vi )
```

Computes a virtual "shim signature", i.e. variation of a given magnetic field component introduced by given displacement of magnetic field source object.

Parameters

f	[out] computed array of field component values
nf	[out] number of values in the field component array computed
obj	[in] reference number of the magnetic field source object
id	[in] string identifying magnetic field component to be computed (can be e.g. "bx", "bz", "ix", "iz",...)
V	[in] array of 3 real numbers specifying cartesian coordinates of the field source object displacement
P1	[in] array of 3 real numbers specifying cartesian coordinates of an edge point of the line segment along which the "shim signature" should be computed
P2	[in] array of 3 real numbers specifying cartesian coordinates of another edge point of the line segment
np	[in] number of points where the magnetic field should be computed
Vi	[in] array of 3 real numbers specifying cartesian coordinates of a vector defining the integration line (is taken into account only if id string specifies field integral component)

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Authors

: O.C.

2.1.2.21 RadFldUnits()

```
EXP int CALL RadFldUnits (
             char * OutStr )
```

Shows the physical units currently in use.

Parameters

OutStr	[out] string of physical units currently in use

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Author

O.C.

2.1.2.22 RadFldUnitsSize()

```
EXP int CALL RadFldUnitsSize (
             int * size )
```

Specifies the length of string about physical units currently in use.

Parameters

size	[out] length of string about physical units currently in use

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.23 RadMatApl()

```
EXP int CALL RadMatApl (
            int * objout,
            int obj,
            int mat )
```

Applies material mat to object obj.

Parameters

objout	[out] reference number of the final object with material applied
obj	[in] reference number of the original object
mat	[in] reference number of the material to be applied

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.24 RadMatLin()

```
EXP int CALL RadMatLin (
            int * mat,
            double * Ksi,
            double * Mr,
            int nMr )
```

Creates a linear anisotropic magnetic material.

Parameters

mat	[out] reference number of the material created
Ksi	[in] array of 2 magnetic susceptibility values for the directions parallel and perpendicular to the easy magnetization axis
Mr	[in] array of 3 cartesian coordinates of the remanent magnetization vector
nMr	[in] number of components of the remanent magnetization vector. If nMr = 1, Mr[0] specifies absolute value of the remanent magnetization; the direction of the easy magnetisation axis is set up by the magnetization vector in the object to which the material is applied (the magnetization vector is specified at the object creation).

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.25 RadMatMvsH()

```
EXP int CALL RadMatMvsH (
             double * M,
             int * nM,
             int obj,
             char * id,
             double * H )
```

Computes magnetization from magnetic field strength vector.

Parameters

M	[out] array of magnetization components calculated
nM	[out] number of magnetization components calculated (length of the array M)
obj	[in] reference number of a material or of an object with material applied
id	[in] string specifying the magnetization components to be calculated (e.g. "mz")
H	[in] magnetic field strength vector in Tesla (mu0∗H)

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.26 RadMatSatAniso()

```
EXP int CALL RadMatSatAniso (
             int * mat,
             double * DataPar,
             int nDataPar,
             double * DataPer,
             int nDataPer )
```

Creates a nonlinear anisotropic magnetic material. The magnetization vector component parallel to the easy axis is computed either by the formula: ms1∗tanh(ksi1∗(hpa-hc1)/ms1) + ms2∗tanh(ksi2∗(hpa-hc2)/ms2) + ms3∗tanh(ksi3∗(hpa-hc3)/ms3) + ksi0∗(hpa-hc0), where hpa is the field strength vector component parallel to the easy axis, or by ksi0∗hpa. The magnetization vector component perpendicular to the easy axis is computed either by the formula: ms1∗tanh(ksi1∗hpe/ms1) + ms2∗tanh(ksi2∗hpe/ms2) + ms3∗tanh(ksi3∗hpe/ms3) + ksi0∗hpe, where hpe is the field strength vector component perpendicular to the easy axis, or by ksi0∗hpe. At least one of the magnetization components should non-linearly depend on the field strength. The direction of the easy magnetisation axis is set up by the magnetization vector in the object to which the material is later applied.

Parameters

mat	[out] reference number of the material created
DataPar	[in] flat array of constants defining the magnetic material behavior in the direction parallel to the easy magnetization axis. It can be {ksi1,ms1,hc1,ksi2,ms2,hc2,ksi3,ms3,hc3,ksi0,hc0} or {ksi0}.
nDataPar	[in] length of the array DataPar. Can be equal to 11 or 1, for the DataPar to be interpreted as {ksi1,ms1,hc1,ksi2,ms2,hc2,ksi3,ms3,hc3,ksi0,hc0} or {ksi0}.
DataPer	[in] flat array of constants defining the magnetic material behavior in the direction perpendicular to the easy magnetization axis. It can be {ksi1,ms1,ksi2,ms2,ksi3,ms3,ksi0} or {ksi0}.
nDataPer	[in] length of the array DataPer. Can be equal to 7 or 1, for the DataPer to be interpreted as {ksi1,ms1,ksi2,ms2,ksi3,ms3,ksi0} or {ksi0}.

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.27 RadMatSatIsoFrm()

```
EXP int CALL RadMatSatIsoFrm (
             int * mat,
             double * KsiMs1,
             double * KsiMs2,
             double * KsiMs3 )
```

Creates a nonlinear isotropic magnetic material with the magnetization magnitude equal M = ms1∗tanh(ksi1∗H/ms1) + ms2∗tanh(ksi2∗H/ms2) + ms3∗tanh(ksi3∗H/ms3), where H is the magnitude of the magnetic field strength vector (in Tesla).

Parameters

mat	[out] reference number of the material created
KsiMs1	[in] array of 2 numbers specifying the parameters ms1 and ksi1 (see the formula above)
KsiMs2	[in] array of 2 numbers specifying the parameters ms2 and ksi2 (see the formula above)
KsiMs3	[in] array of 2 numbers specifying the parameters ms3 and ksi3 (see the formula above)

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.28 RadMatSatIsoTab()

```
EXP int CALL RadMatSatIsoTab (
             int * mat,
             double * MatData,
             int np )
```

Creates a nonlinear isotropic magnetic material with the M versus H curve defined by the list of pairs corresponding values of H and M (H1,M1,H2,M2,...).

Parameters

mat	[out] reference number of the material created
MatData	[in] flat array of material data points (H1,M1,H2,M2,H3,M3,...)
np	[in] number of material data points

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.29 RadMatSatLamFrm()

```
EXP int CALL RadMatSatLamFrm (
             int * mat,
             double * KsiMs1,
             double * KsiMs2,
             double * KsiMs3,
             double p,
             double * N )
```

Creates laminated nonlinear anisotropic magnetic material with packing factor p and the lamination planes perpendicular to the vector N. The magnetization magnitude vs magnetic field strength for the corresponding isotropic material is defined by the formula M = ms1∗tanh(ksi1∗H/ms1) + ms2∗tanh(ksi2∗H/ms2) + ms3∗tanh(ksi3∗H/ms3), where H is the magnitude of the magnetic field strength vector (in Tesla); ksi1, ms1, ksi2, ms2, ksi3, ms3 constants are given by parameters KsiMs1, KsiMs2, KsiMs3.

Parameters

mat	[out] reference number of the material created
KsiMs1	[in] array of 2 numbers specifying the parameters ms1 and ksi1 (see the formula above)
KsiMs2	[in] array of 2 numbers specifying the parameters ms2 and ksi2 (see the formula above)
KsiMs3	[in] array of 2 numbers specifying the parameters ms3 and ksi3 (see the formula above)
p	[in] lamination stacking factor
N	[in] array of 3 numbers specifying cartesian coordinates of a vector normal to the lamination planes; if the pointer N is 0, the lamination planes are assumed to be perpendicular to the magnetization vector in the object to which the material is applied (the magnetization vector should be specified at the object creation)

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.30 RadMatSatLamTab()

```
EXP int CALL RadMatSatLamTab (
                int * n,
                double * MatData,
                int np,
                double p,
                double * N )
```

Creates laminated nonlinear anisotropic magnetic material with packing factor p and the lamination planes perpendicular to the vector N. The magnetization magnitude vs magnetic field strength for the corresponding isotropic material is defined by pairs of values H, M in Tesla.

Parameters

mat	[out] reference number of the material created
MatData	[in] flat array of material data points (H1,M1,H2,M2,H3,M3,...)
np	[in] number of material data points
p	[in] lamination stacking factor
N	[in] array of 3 numbers specifying cartesian coordinates of a vector normal to the lamination planes; if the pointer N is 0, the lamination planes are assumed to be perpendicular to the magnetization vector in the object to which the material is applied (the magnetization vector should be specified at the object creation)

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.31 RadMatStd()

```
EXP int CALL RadMatStd (
             int * mat,
             char * id,
             double m )
```

Creates a pre-defined magnetic material. The material is identified by its name/formula (e.g. "NdFeB").

Parameters

mat	[out] reference number of the material created
id	[in] null-terminated string identifying the material
m	[in] amplitude of the remanent magnetization (for permanent-magnet type materials)

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.32 RadObjAddToCnt()

```
EXP int CALL RadObjAddToCnt (
             int cnt,
             int * Objs,
             int nobj )
```

Adds objects to the container object cnt.

Parameters

cnt	[in] reference number of the container object
Objs	[in] array of reference numbers of the objects (n1, n2, n3,...) to put to the container
nobj	[in] number of objects to put to the container (the length of the array Elems)

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.33 RadObjArcCur()

```
EXP int CALL RadObjArcCur (
            int * n,
            double * P,
            double * R,
            double * Phi,
            double h,
            int nseg,
            char man_auto,
            char a,
            double j )
```

Creates a current carrying finite-length arc of rectangular cross-section. The arc rotation axis is directed along Z.

Parameters

n	[out] reference number of the object created
P	[in] array of 3 cartesian coordinates of the arc center point
R	[in] array of 2 numbers - inner and outer radii
Phi	[in] array of 2 numbers - initial and final azimuth angles
h	[in] height
nseg	[in] number of segments
man_auto	[in] character which can be either 'm' or 'a'; the magnetic field from the arc is then computed based on the number of segments nseg ('m', i.e. "manual"), or on the general absolute precision level specified by the functions RadFldCmpCrt or RadFldCmpPrc ('a', i.e. "automatic")
a	[in] character defining the orientation of the rotation axis of the arc; it can be either 'x', 'y' or 'z'
j	[in] azimuthal current density

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.34 RadObjArcPgnMag()

```
EXP int CALL RadObjArcPgnMag (
            int * n,
            double * P,
            char a,
            double * FlatVert,
            int nv,
            double * Phi,
            int nseg,
            char sym_no,
            double * M )
```

Creates a uniformly magnetized finite-length arc of polygonal cross-section.

Parameters

n	[out] reference number of the object created
P	[in] array of 2 cartesian coordinates defining the position of the rotation axis in the plane perpendicular to this axis
a	[in] character defining the orientation of the rotation axis in 3D space; it can be either to 'x', 'y' or 'z'
FlatVert	[in] flat array of radial and axial coordinates (r1, z1, r2, z2,...) of vertex points of the cross-section polygon
nv	[in] number of vertex points of the cross-section polygon (the length of the FlatVert array is 2∗nv)
Phi	[in] array of 2 numbers - initial and final azimuth angles
nseg	[in] number of segments in the arc
sym_no	[in] character which can be either 's' or 'n'; depending on the value of this switch, the magnetization vectors in nseg sector polyhedrons either will be linked by rotational symmetry ('s'), or will behave as independent magnetization vectors at any subsequent relaxation
M	[in] array of 3 cartesian coordinates of the magnetization vector inside the block

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.35 RadObjBckg()

```
EXP int CALL RadObjBckg (
            int * n,
            double * B )
```

Creates a source of uniform background magnetic field.

Parameters

n	[out] reference number of the object created
B	[in] array of 3 cartesian coordinates of the magnetic field vector

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.36 RadObjCenFld()

```
EXP int CALL RadObjCenFld (
            double * B,
            int * arMesh,
            int obj,
            char type )
```

Provides coordinates of geometrical center point and magnetic field at that point.

Parameters

B	[out] flat array of resulting center point coordinates and field values. If this pointer is 0 at input, the actual output data array can be obtained by calling RadUtiDataGet function. This aray will contain coordinates of geometrical center point(s) and magnetic field components. If obj is a container, the array will include the container members' center points and their magnetic field components.
arMesh	[out] flat array defining the structure of resulting field array, i.e. number of dimensions (arMesh[0]) and number of values per dimension. The actual output data array can be obtained by calling RadUtiDataGet function. This aray will contain coordinates of geometrical center point(s) and magnetic field components. If obj is a container, the array will include the container members' center points and their magnetic field components.
obj	[in] reference number of a magnetic field source object
type	[in] character identifying type of a magnetic field characteristic to return (can be 'A' or 'B' or 'H' or 'J' or 'M')

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.37 RadObjCnt()

```
EXP int CALL RadObjCnt (
              int * n,
              int * Objs,
              int nobj )
```

Creates a container of magnetic field sources.

Parameters

n	[out] reference number of the object created
Objs	[in] array of reference numbers of the objects (n1, n2, n3,...) to put to the container
nobj	[in] number of objects (the length of the array Elems)

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Author

O.C.

2.1.2.38 RadObjCntSize()

```
EXP int CALL RadObjCntSize (
              int * n,
              int cnt )
```

Calculates the number of objects in the container cnt.

Parameters

n	[out] calculated number of objects
cnt	[in] reference number of the container object
deep	[in] switch specifying whether all atomic elements of eventual member containers have to be counted (1) or not (0, default)

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Author

O.C.

2.1.2.39 RadObjCntStuf()

```
EXP int CALL RadObjCntStuf (
              int * Objs,
              int cnt )
```

Fills-in an array with the reference numbers of objects present in the container cnt. The array should be allocated in the calling application. The necessary size of the array can be determined using the function RadObjCntSize.

Parameters

Objs	[out] array of reference numbers of the objects present in the container
cnt	[in] reference number of the container object

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.40 RadObjCutMag()

```
EXP int CALL RadObjCutMag (
              int * Objs,
              int * nobj,
              int obj,
              double * P,
              double * N,
              char * opt )
```

Cuts the object obj by a plane passing through a given point normally to a given vector.

Parameters

Objs	[out] array of reference numbers of the objects produced by the cutting
nobj	[out] amount of the objects produced by the cutting
obj	[in] reference number of the object to cut
P	[in] array of 3 cartesian coordinates of a point the cutting plane passes through
N	[in] array of 3 cartesian coordinates of the vector normal to the cutting plane
opt	[in] pointer to an option string, which can be "Frame->Lab", "Frame->Loc" or 0. This specifies whether the cuting plane is defined in the laboratory frame ("Frame->Lab" or 0) or in the local frame of the object obj.

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.41 RadObjCylMag()

```
EXP int CALL RadObjCylMag (
              int * n,
              double * P,
              double r,
              double h,
              int nseg,
              char a,
              double * M )
```

Creates a finite-length arc magnet of rectangular cross-section.

Parameters

n	[out] reference number of the object created
P	[in] array of 3 cartesian coordinates of the arc center point
R	[in] array of 2 numbers - inner and outer radii
Phi	[in] array of 2 numbers - initial and final azimuth angles
h	[in] height
nseg	[in] number of segments
a	[in] character defining the orientation of the rotation axis of the arc; it can be either to 'x', 'y' or 'z'
M	[in] array of 3 cartesian coordinates of the magnetization vector inside the whole block

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Author

O.C.Creates a cylindrical magnet.

Parameters

n	[out] reference number of the object created
P	[in] array of 3 cartesian coordinates of the cylinder center point
r	[in] cylinder radius
h	[in] cylinder height
nseg	[in] number of segments
a	[in] character defining the orientation of the cylinder axis; it can be either to 'x', 'y' or 'z'
M	[in] array of 3 cartesian coordinates of the magnetization vector inside the whole block

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.42 RadObjDegFre()

```
EXP int CALL RadObjDegFre (
             int * num,
             int obj )
```

Gives number of degrees of freedom for the relaxation of an object.

Parameters

num	[out] number of degrees of freedom
obj	[in] reference number of a 3D object

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.43 RadObjDivMagCyl()

```
EXP int CALL RadObjDivMagCyl (
             int * n,
             int obj,
             double * SbdPar,
             int nSbdPar,
             double * FlatCylPar,
             double rat,
             char * opt )
```

Subdivides (segments) the object obj by a set of coaxial elliptic cylinders.

Parameters

n	[out] reference number of the object created (as a rule, this is a container object)
obj	[in] reference number of the object to subdivide

Parameters

SbdPar	[in] array of 3 (k1,k2,k3) or 6 (k1,q1,k2,q2,k3,q3) subdivision parameters. The meaning of k1, k2 and k3 depends on the value of the option kxkykz: if kxkykz->Numb (default), then k1, k2 and k3 are subdivision numbers; if kxkykz->Size, they are average sizes of the sub-objects to be produced; q1, q2 and q3 are ratios of the last-to-first sub-object sizes. The parameters (k1,q1),(k2,q2) and (k3,q3) correspond to radial, azimuthal, and axial directions respectively.
nSbdPar	[in] number of subdivision parameters (length of the SbdPar array)
FlatCylPar	[in] array of 9 numbers (ax,ay,az,vx,vy,vz,px,py,pz) specifying positions of subdividing coaxial elliptic cylinders in space. The cylinders axis is defined by the point (ax,ay,az) and vector (vx,vy,vz). One of two axes of the cylinder base ellipses is exactly the perpendicular from the point (px,py,pz) to the cylinder axis.
rat	[in] the ratio of the ellipse axes lengths in the bases of subdividing coaxial elliptic cylinders
opt	[in] pointer to options string, which can be "kxkykz->Numb" (default) or "kxkykz->Size" for the segmentation parameters to be interpreted as numbers of peices or their average dimensions; "Frame->Lab", "Frame->LabTot" or "Frame->Loc" for the subdivision to be performed in the laboratory frame or in local frame of the 3D object. The action of "Frame->Lab" and "Frame->LabTot" differs only for containers: "Frame->Lab" means that each of the objects in the container is subdivided separately; "Frame->LabTot" means that all objects in the container are subdivided as one object, by the same planes. opt can contain composition of these sub-strings separated by ";".

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.44 RadObjDivMagPln()

```
EXP int CALL RadObjDivMagPln (
              int * n,
              int obj,
              double * SbdPar,
              int nSbdPar,
              double * FlatNorm,
              char * opt )
```

Subdivides (segments) the object obj by 3 sets of parallel planes.

Parameters

n	[out] reference number of the object created (as a rule, this is a container object)
obj	[in] reference number of the object to subdivide
SbdPar	[in] array of 3 (k1,k2,k3) or 6 (k1,q1,k2,q2,k3,q3) subdivision parameters. The meaning of k1, k2 and k3 depends on the value of the option kxkykz: if kxkykz->Numb (default), then k1, k2 and k3 are subdivision numbers; if kxkykz->Size, they are average sizes of the sub-objects to be produced; q1, q2 and q3 are ratios of the last-to-first sub-object sizes.
nSbdPar	[in] number of subdivision parameters (length of the SbdPar array)

Parameters

FlatNorm	[in] array of 9 numbers specifying cartesian coordinates of 3 vectors normal to the subdivision planes
opt	[in] pointer to options string, which can be "kxkykz->Numb" (default) or "kxkykz->Size" for the segmentation parameters to be interpreted as numbers of peices or their average dimensions; "Frame->Lab", "Frame->LabTot" or "Frame->Loc" for the subdivision to be performed in the laboratory frame or in local frame of the 3D object. The action of "Frame->Lab" and "Frame->LabTot" differs only for containers: "Frame->Lab" means that each of the objects in the container is subdivided separately; "Frame->LabTot" means that all objects in the container are subdivided as one object, by the same planes. opt can contain composition of these sub-strings separated by ";".

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.45 RadObjDpl()

```
EXP int CALL RadObjDpl (
            int * n,
            int obj,
            char * opt )
```

Duplicates the object obj.

Parameters

n	[out] reference number of the object created
obj	[in] reference number of the object to duplicate
opt	[in] pointer to an option string, which can be "FreeSym->False", "FreeSym->True" or 0. This specifies whether the symmetries (transformations with multiplicity more than one) previously applied to the object obj should be simply copied at the duplication ("FreeSym->False" or 0), or a container of new independent objects should be created in place of any symmetry previously applied to the object obj. In both cases the final object created by the duplication has exactly the same geometry as the initial object obj.

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.46 RadObjDrwAtr()

```
EXP int CALL RadObjDrwAtr (
             int obj,
             double * RGB,
             double thcn )
```

Applies drawing attributes - RGB color (r,g,b) and line thickness thcn - to object obj.

Parameters

obj	[in] reference number of the object to which drawing attributes should be applied
RGB	[in] array of 3 numbers from 0 to 1 specifying intensities of red, green and blue colors
thcn	[in] line thickness parameter

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.47 RadObjDrwOpenGL()

```
EXP int CALL RadObjDrwOpenGL (
             int obj,
             char * opt )
```

Starts an application for viewing of 3D geometry of the object obj. The viewer is based on the GLUT / OpenGL graphics library.

Parameters

obj	[in] reference number of the object to be viewed
opt	[in] pointer to options string, which can be "Axes->True" (default) or "Axes->False" for showing or not the axes of the Cartesian laboratory frame; "Faces->True" (default) or "Faces->False" for showing or not visible faces of 3D objects; "EdgeLines->True" (default) or "EdgeLines->False" for highlighting or not the edge lines of 3D objects. opt can contain composition of these option sub-strings separated by ";".

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.48 RadObjDrwQD3D()

```
EXP int CALL RadObjDrwQD3D (
              int obj,
              char * opt )
```

Starts an application for viewing of 3D geometry of the object obj. The viewer is based on the QuickDraw 3D graphics library.

Parameters

obj	[in] reference number of the object to be viewed
opt	[in] pointer to options string, which can be "Axes->True" (default) or "Axes->False" for showing or not the axes of the Cartesian laboratory frame; "Faces->True" (default) or "Faces->False" for showing or not visible faces of 3D objects; "EdgeLines->True" (default) or "EdgeLines->False" for highlighting or not the edge lines of 3D objects. opt can contain composition of these option sub-strings separated by ";".

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.49 RadObjFlmCur()

```
EXP int CALL RadObjFlmCur (
              int * n,
              double * FlatPts,
              int np,
              double i )
```

Creates a filament polygonal line conductor with current. The line conductor is defined by sequence of points in 3D space.

Parameters

n	[out] reference number of the object created
FlatPts	[in] flat array of x, y and z coordinates of the points (x1, y1, z1, x2, y2, z2,...)
np	[in] number of points (the length of the array FlatPts is 3*np)
i	[in] current

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.50 RadObjFullMag()

```
EXP int CALL RadObjFullMag (
             int * n,
             double * P,
             double * L,
             double * M,
             double * K,
             int nK,
             int grp,
             int mat,
             double * RGB )
```

Creates a parallelepiped block with center point {P[0],P[1],P[2]}, dimensions {L[0],L[1],L[2]} and color {RGB[0],R↵ GB[1],RGB[2]}. The block is magnetized according to {M[0],M[1],M[2]} then subdivided according to {K[0],K[1],K[2]} and added into the container grp. grp should be defined in advance by calling RadObjCnt().

Parameters

n	[out] reference number of the object created
P	[in] three cartesian coordinates of the block center point
L	[in] block dimensions
M	[in] three components of the magnetization vector
K	[in] array of subdivision parameters
nK	[in] length of array of subdivision parameters
grp	[in] reference number of the container object
mat	[in] reference number of the magnetic material
RGB	[in] array of 3 numbers from 0 to 1 specifying intensities of red, green and blue colors

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Author

O.C.

2.1.2.51 RadObjGeoLim()

```
EXP int CALL RadObjGeoLim (
             double * L,
             int obj )
```

Computes coordinates of object extrimities in laboratory frame.

Parameters

L	[out] array of 6 numbers representing cartesian coordinates of object extrimities (xmin, xmax, ymin, ymax, zmin, zmax)
obj	[in] reference number of a 3D object

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.52 RadObjGeoVol()

```
EXP int CALL RadObjGeoVol (
             double * v,
             int obj )
```

Computes geometrical volume of a 3D object.

Parameters

v	[out] volume (in mm^3)
obj	[in] reference number of a 3D object

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.53 RadObjM()

```
EXP int CALL RadObjM (
             double * M,
             int * arMesh,
             int obj )
```

Provides coordinates of geometrical center point(s) and magnetization(s) of the object obj.

Parameters

M	[out] flat array of resulting magnetization array. If this pointer is 0 at input, the actual output data array can be obtained by calling RadUtiDataGet function. This aray will contain coordinates of geometrical center point(s) and magnetic field components. If obj is a container, the array will include the container members' center points and their magnetic field components.
arMesh	[out] flat array defining the structure of resulting magnetization array, i.e. number of dimensions (arMesh[0]) and number of values per dimension. The actual output data array can be obtained by calling RadUtiDataGet function. This aray will contain coordinates of geometrical center point(s) and magnetic field components. If obj is a container, the array will include the container members' center points and their magnetic field components.
obj	[in] reference number of a magnetic field source object

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.54 RadObjMltExtPgn()

```
EXP int CALL RadObjMltExtPgn (
             int * n,
             double * FlatVert,
             int * SlicesLen,
             double * Attitudes,
             int ns,
             double * M )
```

?? TO IMPLEMENT ?? Creates a uniformly magnetized volume obtained by rotation of a planar convex polygon over 2 Pi around pre-defined axis.

Parameters

n	[out] reference number of the object created
P	[in] array of 2 cartesian coordinates defining the position of the rotation axis in the plane perpendicular to this axis
FlatVert	[in] flat array of radial and axial coordinates (r1, z1, r2, z2,...) of vertex points of the cross-section polygon
nv	[in] number of vertex points of the cross-section polygon (the length of the FlatVert array is 2∗nv)
nseg	[in] number of segments in the arc
sym_no	[in] character which can be either 's' or 'n'; depending on the value of this switch, the magnetization vectors in nseg sector polyhedrons either will be linked by rotational symmetry ('s'), or will behave as independent magnetization vectors at any subsequent relaxation
a	[in] character defining the orientation of the arc rotation axis; it can be either to 'x', 'y' or 'z'
M	[in] array of 3 cartesian coordinates of the magnetization vector inside the block

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.Attempts to create one uniformly magnetized convex polyhedron or a set of convex polyhedrons based on slices. The slices are assumed to be convex planar polygons parallel to the XY plane.

Parameters

n	[out] reference number of the object created
FlatVert	[in] flat array of x and y coordinates (x11, y11, x12, y12,..., x21, y21, x22, y22,...) of vertex points of the slices (planar polygons)
SlicesLen	[in] array of integer numbers equal to the numbers of vertex points in each slice; the order of the slices is the same as in the FlatVert array (the length of the FlatVert array is twice the sum of elements of the SlicesLen array)
Attitudes	[in] array of vertical coordinates (or attitudes) of the slices, in the ascending order (which is the same as for the SlicesLen array)
ns	[in] number of slices (or the length of the Attitudes and SlicesLen arrays)
M	[in] array of 3 cartesian coordinates of the magnetization vector inside the whole block

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.55 RadObjMltExtRtg()

```
EXP int CALL RadObjMltExtRtg (
            int * n,
            double * FlatCenPts,
            double * FlatRtgSizes,
            int ns,
            double * M )
```

Attempts to create one uniformly magnetized convex polyhedron or a set of convex polyhedrons based on rectangular slices. The rectangular slices are assumed to be parallel to the XY plane.

Parameters

n	[out] reference number of the object created
FlatCenPts	[in] flat array of x, y and z coordinates (x1, y1, z1, x2, y2, z2,...) of the slices center points
FlatRtgSizes	[in] flat array of sizes of the slice rectangles along x and y (wx1, wy1, wx2, wy2,...)
ns	[in] number of slices (the length of the FlatCenPts array is 3∗ns, the length of the FlatRtgSizes array is 2∗ns)
M	[in] array of 3 cartesian coordinates of the magnetization vector inside the whole block

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.56 RadObjMltExtTri()

```
EXP int CALL RadObjMltExtTri (
              int * n,
              double xc,
              double lx,
              double * FlatVert,
              double * FlatSubd,
              int nv,
              char a,
              double * M,
              char * opt )
```

Creates triangulated extruded polygon block, i.e. an extruded polygon with its bases subdivided by triangulation.

Parameters

n	[out] reference number of the object created
xc	[in] the horizontal coordinate of the block center of gravity
lx	[in] the thickness (extrusion size)
FlatVert	[in] flat array of y and z coordinates (y1, z1, y2, z2,...) of vertex points describing the polygon in 2D
FlatSubd	[in] flat array of subdivision parameters for base polygon segments, two numbers for each segment: the first defining number of sub-segments, the second - "gradient" of the segmentation
nv	[in] number of vertex points of the 2D polygon (the length of the FlatVert and FlatSubd arrays is 2∗nv)
a	[in] character identifying extrusion direction (can be 'x', 'y' or 'z')
M	[in] array of 3 cartesian coordinates of the magnetization vector inside the block
opt	[in] pointer to options string, which can be e.g. "ki->...,TriAngMin->...,TriAreaMax->...,ExtOpt->..." or each of these tokens separately, or 0; default values are "ki->Numb" (rather than "ki->Size"), "TriAngMin->20"

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.57 RadObjPolyhdr()

```
EXP int CALL RadObjPolyhdr (
              int * n,
              double * FlatVert,
              int nv,
              int * FlatFaces,
              int * FacesLen,
              int nf,
              double * M,
              double * M_LinCoef,
              double * J,
              double * J_LinCoef )
```

Creates a uniformly magnetized polyhedron (closed volume limited by planes).

Parameters

n	[out] reference number of the object created
FlatVert	[in] flat array of x, y and z coordinates (x1, y1, z1, x2, y2, z2,...) of the polyhedron vertex points
nv	[in] number of vertex points of the polyhedron (the length of the FlatVert array is 3∗nv)
FlatFaces	[in] flat array of indexes of vertex points defining the polyhedron faces (f1i1, f1i2,..., f2i1, f2i2,...)
FacesLen	[in] array of integer numbers equal to the numbers of vertex points in each face of the polyhedron; the order of the faces is the same as in the FlatFaces array (the length of the FlatFaces array is equal to the sum of elements of the FacesLen array)
nf	[in] number of faces of the polyhedron (or the length of the FacesLen array)
M	[in] array of 3 cartesian components of magnetization vector inside the block
M_LinCoef	[in] array of 9 coefficients of linearly-varying magnetization vector inside the block
J	[in] array of 3 cartesian components of current density vector inside the block
J_LinCoef	[in] array of 9 coefficients of linearly-varying current density vector inside the block

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.58 RadObjRaceTrk()

```
EXP int CALL RadObjRaceTrk (
              int * n,
              double * P,
              double * R,
              double * L,
              double h,
              int nseg,
              char man_auto,
              char a,
              double j )
```

Creates a current carrying racetrack coil. The coil consists of four 90-degree bents connected by four straight parts parallel to the XY plane.

Parameters

n	[out] reference number of the object created
P	[in] array of 3 cartesian coordinates of the racetrack center point
R	[in] array of 2 numbers - inner and outer radii
L	[in] array of 2 numbers - straight section lengths
h	[in] height
nseg	[in] number of segments
man_auto	[in] character which can be either 'm' or 'a'; the magnetic field from the arc is then computed based on the number of segments nseg ('m', i.e. "manual"), or on the general absolute precision level specified by the functions RadFldCmpCrt or RadFldCmpPrc ('a', i.e. "automatic")
a	[in] character defining the orientation of the rotation axis of the arc; it can be either 'x', 'y' or 'z'
j	[in] azimuthal current density

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.59 RadObjRecCur()

```
EXP int CALL RadObjRecCur (
             int * n,
             double * P,
             double * L,
             double * J )
```

Creates a current carrying rectangular parallelepiped block. The parallelepiped block is defined through its center point P[3], dimensions L[3], and current density vector J[3]."

Parameters

n	[out] reference number of the object created
P	[in] array of 3 cartesian coordinates of the block center of gravity
L	[in] array of 3 dimensions of the block
J	[in] array of 3 cartesian coordinates of the current density vector inside the block

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.60 RadObjRecMag()

```
EXP int CALL RadObjRecMag (
             int * n,
             double * P,
             double * L,
             double * M )
```

Creates a uniformly magnetized rectangular parallelepiped. The parallelepiped block is defined through its center point P[3], dimensions L[3], and magnetization M[3]."

Parameters

n	[out] reference number of the object created
P	[in] array of 3 cartesian coordinates of the block center of gravity
L	[in] array of 3 dimensions of the block
M	[in] array of 3 cartesian coordinates of the magnetization vector inside the block

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Author

O.C.

2.1.2.61 RadObjScaleCur()

```
EXP int CALL RadObjScaleCur (
             int n,
             double scaleCoef )
```

Scales current (density) in a 3D object by multiplying it by a constant.

Parameters

n	[out] reference number of the object with current (density) to be scaled
scaleCoef	[in] scaling coefficient

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Author

O.C.

2.1.2.62 RadObjSetM()

```
EXP int CALL RadObjSetM (
              int obj,
              double ∗ M )
```

Sets magnetization of the object obj.

Parameters

obj	[in] reference number of a magnetic field source object
M	[in] flat array of 3 components of the magnetization vector

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.63 RadObjThckPgn()

```
EXP int CALL RadObjThckPgn (
              int ∗ n,
              double xc,
              double lx,
              double ∗ FlatVert,
              int nv,
              char a,
              double ∗ M )
```

Creates a uniformly magnetized extruded polygon. The extrusion axis is directed along X axis of the laboratory frame.

Parameters

n	[out] reference number of the object created
xc	[in] the horizontal coordinate of the block center of gravity
lx	[in] the thickness (extrusion size)
FlatVert	[in] flat array of y and z coordinates (y1, z1, y2, z2,...) of vertex points describing the polygon in 2D
nv	[in] number of vertex points of the 2D polygon (the length of the FlatVert array is 2∗nv)
a	[in] character identifying extrusion direction (can be 'x', 'y' or 'z')
M	[in] array of 3 cartesian coordinates of the magnetization vector inside the block

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.64 RadRlxAuto()

```
EXP int CALL RadRlxAuto (
             double * D,
             int * n,
             int intrc,
             double prec,
             int iter,
             int meth,
             const char * opt )
```

Executes automatic relaxation procedure for the interaction matrix intrc. The relaxation stops whenever the change of magnetization (averaged over all sub-elements) between two successive iterations is smaller than prec or the number of iterations is larger than iter.

Parameters

D	[out] an array of four numbers specifying: [0] average absolute change in magnetization after previous iteration over all the objects participating in the relaxation, [1] maximum absolute value of magnetization over all the objects participating in the relaxation, [2] maximum absolute value of magnetic field strength over central points of all the objects participating in the relaxation, and [3] actual number of iterations done. The values [0]-[2] are those of last iteration.
n	[out] length of array D
intrc	[in] an integer number referencing the interaction matrix
prec	[in] a real number specifying an absolute precision value for magnetization (in Tesla), to be reached by the end of the relaxation
iter	[in] maximum number of iterations permitted to reach the specified precision
meth	[in] an integer number specifying the method of relaxation to be used (values 0, 3 - 5 can be used; 0 means default method)
opt	[in] pointer to an option string, which can be "ResetM->True" (default) or "ResetM->False"

Returns

integer error code (0 : no error, $>$0 : error number, $<$0 : warning number)

Author

O.C.

2.1.2.65 RadRlxMan()

```
EXP int CALL RadRlxMan (
             double * D,
             int * n,
             int intrc,
             int meth,
             int iter,
             double rlxpar )
```

Executes manual relaxation procedure for the interaction matrix intrc.

Parameters

D	[out] an array of four numbers specifying: [0] average absolute change in magnetization after previous iteration over all the objects participating in the relaxation, [1] maximum absolute value of magnetization over all the objects participating in the relaxation, [2] maximum absolute value of magnetic field strength over central points of all the objects participating in the relaxation, and [3] actual number of iterations done. The values [0]-[2] are those of last iteration.
n	[out] length of array D
intrc	[in] an integer number referencing the interaction matrix
meth	[in] an integer number specifying the method of relaxation to be used
iter	[in] an integer number specifying number of iterations to be made
rlxpar	[in] a floating point number between 0 and 1 specifying the relaxation parameter

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.66 RadRlxPre()

```
EXP int CALL RadRlxPre (
              int * n,
              int obj,
              int srcobj )
```

Builds interaction matrix for the object obj.

Parameters

n	[out] reference number of the interaction matrix created
obj	[in] reference number of the object for which the interaction matrix should be created
srcobj	[in] reference number of the object creating additional constant magnetic field

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.67 RadRlxUpdSrc()

```
EXP int CALL RadRlxUpdSrc (
              int intrc )
```

Updates external field data for the relaxation (to take into account e.g. modification of currents in coils, if any) without rebuilding the interaction matrix.

Parameters

intrc	[in] an integer number referencing the interaction object

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.68 RadSolve()

```
EXP int CALL RadSolve (
              double * D,
              int * n,
              int obj,
              double prec,
              int iter,
              int meth )
```

Builds an interaction matrix and performs a relaxation procedure. The relaxation stops whenever the change of magnetization (averaged over all sub-elements) between two successive iterations is smaller than prec or the number of iterations is larger than iter. The interaction matrix is deleted.

Parameters

D	[out] an array of four numbers specifying: [0] average absolute change in magnetization after previous iteration over all the objects participating in the relaxation, [1] maximum absolute value of magnetization over all the objects participating in the relaxation, [2] maximum absolute value of magnetic field strength over central points of all the objects participating in the relaxation, and [3] actual number of iterations done. The values [0]-[2] are those of last iteration.
n	[out] length of array D
obj	[in] an integer number specifying the object to solve for magnetization
prec	[in] a real number specifying an absolute precision value for magnetization (in Tesla), to be reached by the end of the relaxation
iter	[in] maximum number of iterations permitted to reach the specified precision
meth	[in] an integer number specifying the method of relaxation to be used (values 0, 3 - 5 can be used; 0 means default method)

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

P.E., O.C.

2.1.2.69 RadTrfCmbL()

```
EXP int CALL RadTrfCmbL (
            int * fintrf,
            int origtrf,
            int trf )
```

Multiplies original space transformation origtrf by another transformation trf from left.

Parameters

fintrf	[out] reference number of the final space transformation
origtrf	[in] reference number of the original space transformation to be multiplied
trf	[in] reference number of another space transformation

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.70 RadTrfCmbR()

```
EXP int CALL RadTrfCmbR (
            int * fintrf,
            int origtrf,
            int trf )
```

Multiplies original space transformation origtrf by another transformation trf from right.

Parameters

fintrf	[out] reference number of the final space transformation
origtrf	[in] reference number of the original space transformation to be multiplied
trf	[in] reference number of another space transformation

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.71 RadTrfInv()

```
EXP int CALL RadTrfInv (
              int * trf )
```

Creates a field inversion.

Parameters

trf	[out] reference number of the symmetry object created

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.72 RadTrfMlt()

```
EXP int CALL RadTrfMlt (
              int * objout,
              int obj,
              int trf,
              int mlt )
```

Creates mlt-1 symmetry objects of the object obj. Each symmetry object is derived from the previous one by applying the transformation trf to it. Following this, the object obj becomes equivalent to mlt different objects.

Parameters

objout	[out] reference number of the final object with symmetries applied
obj	[in] reference number of the original object to which symmetries should be applied
trf	[in] reference number of a space transformation
mlt	[in] multiplicity of the space transformation

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.73 RadTrfOrnt()

```
EXP int CALL RadTrfOrnt (
             int * objout,
             int obj,
             int trf )
```

Orients object obj by applying transformation trf to it once.

Parameters

objout	[out] reference number of the final object with space transformation applied
obj	[in] reference number of the original object
trf	[in] reference number of a space transformation

2.1.2.74 RadTrfPlSym()

```
EXP int CALL RadTrfPlSym (
             int * trf,
             double * P,
             double * N )
```

Creates a symmetry with respect to plane defined by a point and a normal vector.

Parameters

trf	[out] reference number of the symmetry object created
P	[in] array of 3 numbers representing cartesian coordinates of a point in the plane
N	[in] array of 3 numbers representing cartesian coordinates of a vector normal to the plane

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.75 RadTrfRot()

```
EXP int CALL RadTrfRot (
             int * trf,
             double * P,
             double * V,
             double phi )
```

Creates a rotation.

Parameters

trf	[out] reference number of the symmetry object created
P	[in] array of 3 numbers representing cartesian coordinates of a point belonging to the rotation axis
V	[in] array of 3 numbers representing cartesian coordinates of a vector parallel to the rotation axis
phi	[in] rotation angle

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.76 RadTrfTrsl()

```
EXP int CALL RadTrfTrsl (
             int * trf,
             double * V )
```

Creates a translation.

Parameters

trf	[out] reference number of the symmetry object created
V	[in] array of 3 numbers representing cartesian coordinates of the translation vector

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.77 RadTrfZerPara()

```
EXP int CALL RadTrfZerPara (
             int * objout,
             int obj,
             double * P,
             double * N )
```

Creates an object mirror with respect to a plane. The object mirror possesses the same geometry as obj, but its magnetization and/or current densities are modified in such a way that the magnetic field produced by the obj and its mirror in the plane of mirroring is perpendicular to this plane.

Parameters

objout	[out] reference number of the final object
obj	[in] an integer number referencing the original object
P	[in] array of 3 cartesian coordinates of a pointg in the mirror plane
N	[in] array of 3 cartesian coordinates of a vector normal to the mirror plane

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

P.E., O.C.

2.1.2.78 RadTrfZerPerp()

```
EXP int CALL RadTrfZerPerp (
             int * objout,
             int obj,
             double * P,
             double * N )
```

Creates an object mirror with respect to a plane. The object mirror possesses the same geometry as obj, but its magnetization and/or current densities are modified in such a way that the magnetic field produced by the obj and its mirror in the plane of mirroring is parallel to this plane.

Parameters

objout	[out] reference number of the object
obj	[in] an integer number referencing the original object
P	[in] array of 3 cartesian coordinates of a pointg in the mirror plane
N	[in] array of 3 cartesian coordinates of a vector normal to the mirror plane

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

P.E., O.C.

2.1.2.79 RadUtiDataGet()

```
EXP int CALL RadUtiDataGet (
             char * pcData,
             const char typeData[3],
             long key = 0 )
```

Returns data resulting from previous calculations in cases when the data size was not known 'a priori', e.g. after executing functions RadObjM, RadObjCenFld, RadUtiDmp,...

Parameters

size	[out] pointer to the resulting data (to be allocated in calling function)
typeData	[in] string identifying type of the data: "mad" for multi-dim. array of double, "mai" for multi-dim. array of integer, "bin" for byte array, "asc" for ASCII string, "d" for double, "i" for integer
key	[in] additional identifier of the data to be extracted, e.g. to ensure thread safety (not implemented yet)

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.80 RadUtiDel()

```
EXP int CALL RadUtiDel (
             int * n,
             int obj )
```

Deletes object obj.

Parameters

n	[out] dummy
obj	[in] reference number of the object to be deleted

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.81 RadUtiDelAll()

```
EXP int CALL RadUtiDelAll (
             int * n )
```

Deletes all previously created objects.

Parameters

n	[out] dummy

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.82 RadUtiDmp()

```
EXP int CALL RadUtiDmp (
             char * OutStr,
             int * pSize,
             int * arObj,
             int nObj,
             char * AscOrBin )
```

Outputs information about object obj.

Parameters

OutStr	[out] string containing information about obj
arObj	[in] array of object reference numbers to show information for
nObj	[in] length of array of object reference numbers
AscOrBin	[in] string specifying format of the output information string, can be "asc" (for ASCII) or "bin" (for Binary)

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.83 RadUtiDmpPrs()

```
EXP int CALL RadUtiDmpPrs (
            int * arElem,
            int * nElem,
            unsigned char * sBytes,
            int nBytes )
```

Outputs information about object obj after reading it from internal buffer.

Parameters

OutStr	[out] string containing information about obj
AscOrBin	[in] string specifying format of the output information string, can be "asc" (for ASCII) or "bin" (for Binary)

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.Gives necessary length of the string to include dump information about object obj.

Parameters

size	[out] size of string containing information about obj
arObj	[in] array of object reference numbers to show information for
nObj	[in] length of array of object reference numbers
AscOrBin	[in] string specifying format of the output information string, can be "asc" (for ASCII) or "bin" (for Binary)
doEraseBuf	[in] switch specifying whether the output buffer has to be erased or not (leaving the buffer unerased allows not to repeat the dump operation at a subsequent call of RadUtiDmp)

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

O.C.

2.1.2.84 RadUtiIntrptTim()

```
EXP int CALL RadUtiIntrptTim (
             double * d,
             double t )
```

Sets interruption time quanta in seconds for platforms with no preemptive multitasking.

Parameters

t	[in] interruption time quanta [s]

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

P.E., O.C.

2.1.2.85 RadUtiVer()

```
EXP int CALL RadUtiVer (
             double * d )
```

Identifies the version number of the Radia DLL.

Parameters

d	[out] version number

Returns

integer error code (0 : no error, >0 : error number, <0 : warning number)

Author

P.E., O.C.

2.1.2.86 RadUtiYeldFuncSet()

```
EXP int CALL RadUtiYeldFuncSet (
             int(*)() pExtFunc )
```

2.1.2.87 RadWarGet()

```
EXP const char* CALL RadWarGet (
             int er )
```

Returns the warning message associated to warning number. This function cannot be called from Visual Basic For Applications.

Parameters

er	[in] warning number

Returns

the chain of characters representing the warning string

Author

P. Elleaume

Version

1.0

See also

GetWarningSize and GetWarningText

2.1.2.88 RadWarGetSize()

```
EXP int CALL RadWarGetSize (
             int * siz,
             int er )
```

Returns the length of the warning message not counting "\0".

Parameters

siz	[out] length of the warning message
er	[in] warning number

Returns

integer error code (0 : No Error, >0 : Error Number, <0 : Warning Number)

Author

P. Elleaume

Version

1.0

See also

dllGetWarningText

2.1.2.89 RadWarGetText()

```
EXP int CALL RadWarGetText (
              char * t,
              int er )
```

Returns the text of warning message associated to error number.

Parameters

t	[out] warning message string
er	[in] error number

Returns

integer error code (0 : No Error, >0 : Error Number, <0 : Warning Number)

Author

P. Elleaume

Version

1.0

See also

GetWarningSize

Index