GTEC

version 0.1

Generated by Doxygen 1.8.11

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 igrf Class Reference

`#include <igrf.hpp>`

Public Member Functions

- igrf (std::string fname)
  Constructor with Input file.
- double getMODIP (triple &pos, int &t)
  Function to compute MODIP.

3.1.1 Detailed Description

Author

Muhammad Owais

Date

14/01/17

3.1.2 Constructor & Destructor Documentation

3.1.2.1 igrf::igrf (std::string *fname*)

Constructor with Input file.

Constructs igrf object by reading input IGRF coefficients file.

Parameters

fname	IGRF coefficients file name.

Here is the call graph for this function:

3.1.3 Member Function Documentation

3.1.3.1 double igrf::getMODIP (triple & *pos*, int & *t*)

Function to compute MODIP.

This function compute MODIP (Modified Dip) given ellipsoidal coordinates of the point and time (in unit of years).

Parameters

pos	ellipsoidal coordinates of the point as triple object.
t	time (in unit of years).

Returns

Returns computed MODIP.

The documentation for this class was generated from the following files:

- igrf.hpp
- igrf.cpp

3.2 internalTime Class Reference

`#include <internalTime.hpp>`

Public Member Functions

- internalTime (int Y, int M, int D, int h, int m, int s)
  Constructor with explicit values.
- void parse (std::string strtime)
  Member function parse.
- void toUNIXTime ()
  Member Function, providing UNIX time.
- internalTime ()

Public Attributes

- int year
- int month

  Stores Month as Integer.
- int day

  Stores day as Integer.
- int hour

  Stores hour as Integer.
- int minute

  Stores minute as Integer.
- int second

  Stores second as Integer.
- int UNIX

  Stores Converted UNIX Time as Integer.

3.2.1 Detailed Description

Author

Muhammad Owais

Date

04/12/16

3.2.2 Constructor & Destructor Documentation

3.2.2.1 internalTime::internalTime (int *Y,* int *M,* int *D,* int *h,* int *m,* int *s*)

Constructor with explicit values.

Constructs internalTime object explicitly taking date/time values as parameters. Requires 6 integers (YY↩YY,MM,DD,hh,mm,ss).

Parameters

Y	year(YYYY), given as integer
M	Month(MM), given as integer
D	Day(DD), given as integer
h	Hour(hh), given as integer
m	Minute(mm), given as integer
s	Second(ss), given as integer

3.2.2.2 internalTime::internalTime ( )

Default Constructor.

3.2.3 Member Function Documentation

3.2.3.1 void internalTime::parse (std::string *strtime*)

Member function parse.

Member function parse sets internal values by parsing a given string representing date/time values.

Parameters

strtime	string representing time.

3.2.3.2 void internalTime::toUNIXTime ( )

Member Function, providing UNIX time.

Member function, converting stored time to UNIX time.

3.2.4 Member Data Documentation

3.2.4.1 int internalTime::year

Stores Year as Integer

The documentation for this class was generated from the following files:

- internalTime.hpp
- internalTime.cpp

3.3 navigation Class Reference

`#include <navigation.hpp>`

Public Member Functions

- void read ()
  Member function read.
- navigation (std::vector< std::string > fnames)
  Constructor with Input files.
- void getPositionR (ephemerisR &initialConditions, int h, triple &pos)
  Function to compute GLONASS satellite positions.
- void getPositionGE (ephemerisGE &initial, int t, triple &pos)
  Function to compute GPS/Galileo/BeiDou satellite positions.
- void ecefToEllipsoidal (const triple &ecef, triple &ellipsoid)
  Function to convert ECEF to ellipsoidal coordinates.
- void satElevAzim (triple &markerECEF, triple &sat, triple &markerEllip, double &elevation, double &azimuth)
  Function to compute satellite elevation and azimuth.
- int computeIPP (const triple &marker, const triple &sat, const double &rh, triple &IPP, double &coschi)
  Function to compute IPP and zenith angle over IPP.

Public Attributes

- std::vector< std::string > fileNames

  list of file names to read from
- float version

  Stores RINEX version.
- int leapSeconds

  Stores leapSeconds from Navigation files.
- std::vector< std::vector< ephemerisGE > > ephemeris_G

  Vector to store objects of type ephemerisGE for GPS.
- std::vector< std::vector< ephemerisGE > > ephemeris_E

  Vector to store objects of type ephemerisGE for Galileo.
- std::vector< std::vector< ephemerisR > > ephemeris_R

  Vector to store objects of type ephemerisR for GLONASS.
- std::vector< std::vector< ephemerisGE > > ephemeris_C

  Vector to store objects of type ephemerisGE for BeiDou.

Private Member Functions

- navigation ()
- float eccAnomaly (float M, float e)

  Function to compute eccentricity anomaly Ek.
- void applyRotations (float &Lk, float &ik, float &uk, float &rk, triple &pos)

  This Function apply rotations around uk, ik and Lk.

3.3.1 Detailed Description

Author

Muhammad Owais

Date

05/12/16

3.3.2 Constructor & Destructor Documentation

3.3.2.1 navigation::navigation (std::vector< std::string > *fnames*)

Constructor with Input files.

Constructs navigation object by reading input navigation files defined by fnames.

Parameters

fnames	Vector of Navigation file names.

3.3.2.2 navigation::navigation ( ) [private]

Default Constructor. Hidden, cannot be used.

3.3.3 Member Function Documentation

3.3.3.1 void navigation::applyRotations (float & *Lk*, float & *ik*, float & *uk*, float & *rk*, triple & *pos*) [private]

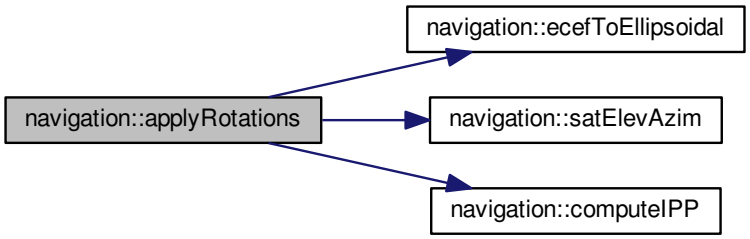
This Function apply rotations around uk, ik and Lk.

This Function apply rotations around uk, ik and Lk, Rotation == | Xk | | rk | | Yk | = R3(-Lk)R1(-ik)R3(-uk) | 0 | | Zk | | 0 | Wwhere R1 and R3 are the rotation matrices defined at: http://www.navipedia.net/index.↩php/Transformation_between_Terrestrial_Frames By Hernández-Pajares, Technical University of Catalonia, Spain.

Parameters

Lk	Longitude of the ascending node LAMBDAk.
ik	Inclination of the orbital plane.
uk	Argument of latitude.
rk	Radial distance rk.
pos	triple object returned with computed coordinates.

Here is the call graph for this function:



3.3.3.2 int navigation::computeIPP (const triple & *marker*, const triple & *sat*, const double & *rh*, triple & *IPP*, double & *coschi*)

Function to compute IPP and zenith angle over IPP.

This function computes IPP (Ionospheric Pierce Point) in ECEF cartesian coordinates and zenith angle over IPP using sphere-line euation. Reference height of ionosphere, marker (receiver station), and satellite position are given as inputs. An integer status is returned describing solution type.

Parameters

marker	ECEF cartesian coordinates for marker (receiver station) as a triple object.
sat	ECEF cartesian coordinates for satellite as a triple object.
rh	Ionosphere reference height in Kilometers.
IPP	Output ECEF cartesian coordinates for IPP as a triple object.
coschi	Output zenith angle over IPP.

3.3.3.3 float navigation::eccAnomaly (float *M,* float *e*) [private]

Function to compute eccentricity anomaly Ek.

This Function computes eccentricity anomaly Ek by Solving (iteratively) the Kepler equation for the eccentricity anomaly, using Newton–Raphson method, Equation –> Mk = Ek - (e $*$ Sin(Ek))

Parameters

M	mean anomaly for reference time tk.
e	eccentricity.

3.3.3.4 void navigation::ecefToEllipsoidal (const triple & *ecef,* triple & *ellipsoid*)

Function to convert ECEF to ellipsoidal coordinates.

This function converts ECEF cartesian coordinates (x, y, z) to ellipsoidal coordinates (φ, λ, h) respectively lattitude, longitude, and height.

Parameters

ecef	ECEF cartesian coordinates.
ellipsoid	Output ellipsoidal coordinates (φ, λ, h) .

3.3.3.5 void navigation::getPositionGE (ephemerisGE & *initial,* int *t,* triple & *pos*)

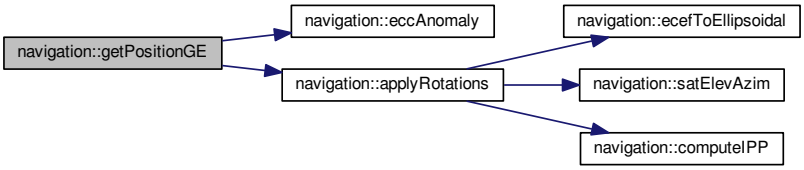
Function to compute GPS/Galileo/BeiDou satellite positions.

This function calculates GPS/Galileo/BeiDou satellite coordinates given an ephemerisGE object and time for which coordinates are required.

Parameters

initial	ephemerisGE object containing initial Keplerian elements.
t	Integer time for which coordinates are to be computed.
pos	triple object returned with computed coordinates.

Here is the call graph for this function:



3.3.3.6 void navigation::getPositionR (ephemerisR & *initialConditions,* int *h,* triple & *pos*)

Function to compute GLONASS satellite positions.

This function calculates GLONASS satellite coordinates given an ephemerisR object, and a step size.

Parameters

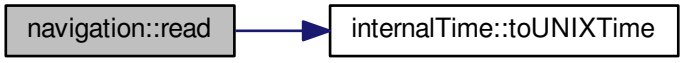
initialConditions	ephemerisR object containing initial conditions.
h	Integer step size for next coordinate.
pos	triple object returned with computed coordinates.

3.3.3.7 void navigation::read ( )

Member function read.

Member function read parses input navigation files and constructs internal navigation structure.

Here is the call graph for this function:



3.3.3.8 void navigation::satElevAzim (triple & *markerECEF,* triple & *sat,* triple & *markerEllip,* double & *elevation,* double & *azimuth*)

Function to compute satellite elevation and azimuth.

This function computes satellite elevation and azimuth given marker (receiver station) position in ECEF and ellipsoidal coordinates and satellite position in ECEF coordinates. This function implements elevation/azimuth computation as described in Transformations between ECEF and ENU coordinates J. Sanz Subirana, J.M. Juan Zornoza and M. Hernández-Pajares, Technical University of Catalonia, Spain.

Parameters

markerECEF	ECEF cartesian coordinates for marker (receiver station) as a triple object.
sat	ellipsoidal coordinates for satellite as a triple object.
markerEllip	ellipsoidal coordinates for marker (receiver station) as a triple object.
elevation	Output satellite elevation.
azimuth	Output satellite azimuth.

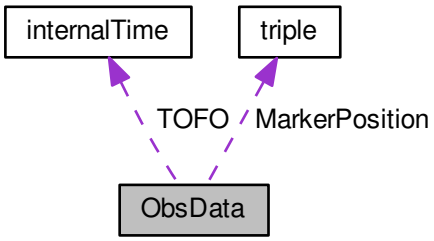
The documentation for this class was generated from the following files:

- navigation.hpp
- navigation.cpp

3.4 ObsData Class Reference

`#include <ObsData.hpp>`

Collaboration diagram for ObsData:



Public Member Functions

- void read ()
  Member function read.
- ObsData (std::vector< std::string > fvec, std::string sysString)
  Constructor with Input files, and system string.
- void cleanUp ()
  Clean-up function.
- void pre_process (int minArcLen, int intrpolIntrvl, int deg)
  Function to perform preprocessing.
- void buildB ()
  Builds matrix B.
- void dumpRawMatrix (const double *mat, int &dim1, int &dim2)
  Function to dump raw matrix.

Public Attributes

- std::vector< std::string > fnames

  list of file names to read from
- triple MarkerPosition

  triple Object to store receiver-station position
- float version

  Stores RINEX version of observation files.
- int interval

  Interval between observations in data file.
- bool hasGPS

  Flag to indicate whether Data file contains GPS Data.
- bool hasGLO

  Flag to indicate whether Data file contains GLONASS Data.
- bool hasGAL

  Flag to indicate whether Data file contains Galileo Data.
- bool hasBEI

  Flag to indicate whether Data file contains BeiDou Data.
- bool readGPS

  Flag to indicate whether to process GPS Data.
- bool readGLO

  Flag to indicate whether to process GLONASS Data.
- bool readGAL

  Flag to indicate whether to process Galileo Data.
- bool readBEI

  Flag to indicate whether to process BeiDou Data.
- bool hasTOFO

  Time of first observation flag.
- std::string TOFO_system

  Time system of first observation from observation Header.
- internalTime TOFO

  internalTime Object to store Time of first observation
- std::vector< int > timeline_main

  Integer vector to store epochs in UNIX time.
- std::vector< std::vector< float > > GPS_ucTEC

  Vectors to store raw non-calibrated TEC for GPS Satellites.
- std::vector< std::vector< float > > GLO_ucTEC

  Vectors to store raw non-calibrated TEC for GLONASS Satellites.
- std::vector< std::vector< float > > GAL_ucTEC

  Vectors to store raw non-calibrated TEC for Galileo Satellites.
- std::vector< std::vector< float > > BDU_ucTEC

  Vectors to store raw non-calibrated TEC for BeiDou Satellites.
- std::vector< double > S

  Stores vector S (non-calibrated TEC).
- std::vector< int > S_arcnum

  Stores arc numbers for S.
- std::vector< int > S_prn

  Stores Satellite IDs for S.
- int size_of_S

  Indicates size of S.
- int numArcs

Indicates total number of arcs.

- double * B

  Stores matrix B.

- std::vector< ptr_pair > arcs

  Initial non-zero arc pinters.

- std::vector< ptr_pair > arcs2

  Arc pointers without zeros.

- std::vector< ptr_pair > arcs3

  Arc pointers without gaps.

Private Member Functions

- ObsData ()

  default hidden Constructor

- void setSysFlags (std::string sysString)

  Sets system flags.

- void setArcStartEnd ()

  Sets Arc pointers using ptr_pair objects.

- int lagrangeInterpolation (float *target, float *s, float *e, int deg)

  Function to perform lagrange interpolation.

3.4.1 Detailed Description

Author

Muhammad Owais

Date

05/12/16

3.4.2 Constructor & Destructor Documentation

3.4.2.1 ObsData::ObsData (std::vector< std::string > *fvec*, std::string *sysString*)

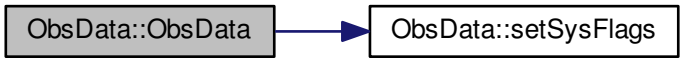
Constructor with Input files, and system string.

Constructs observation object by seting input observation file name vector fnames given file names and setting system flags given system string.

Parameters

fvec	Vector of observation file names.
sysString	string (any combination of 'G','R','E','C') defining constellations being processed.

Here is the call graph for this function:



3.4.3 Member Function Documentation

3.4.3.1 void ObsData::buildB ( )

Builds matrix B.

This function builds and stores matrix B.

3.4.3.2 void ObsData::cleanUp ( )

Clean-up function.

This function cleans up internal workspace, should be called before end of object's lifetime.

3.4.3.3 void ObsData::dumpRawMatrix (const double * *mat*, int & *dim1*, int & *dim2*)

Function to dump raw matrix.

This Function dumps raw matrix to standard output stream, usefull in debugging purposes.

Parameters

mat	pointer to stored matrix.
dim1	First dimension of matrix (number of rows).
dim2	Second dimension of matrix (number of columns).

3.4.3.4 int ObsData::lagrangeInterpolation (float * *target*, float * *s*, float * *e*, int *deg*) [private]

Function to perform lagrange interpolation.

This function performs lagrange Interpolation needed in preprocessing phase, given a required degree for interpolation.

Parameters

target	pointer to the value being interpolated.

Parameters

s	start pointer of the arc.
e	end pointer of the arc.
deg	degree of Interpolation.

3.4.3.5 void ObsData::pre_process (int *minArcLen*, int *intrpolIntrvl*, int *deg*)

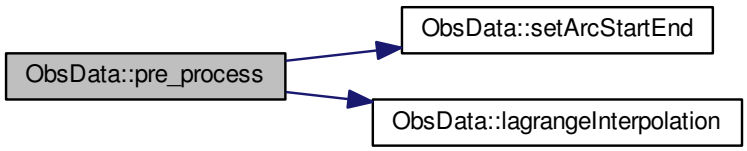
Function to perform preprocessing.

This function performs preprocessing by filling gaps using lagrange interpolation and removing phase jumps using quartiles and Inter Quartile Range.

Parameters

minArcLen	minimum data duration(Seconds) to consider an arc valid.
intrpolIntrvl	Maximum gap duration (Seconds) to interpolate.
deg	Degree of Interpolation, passed to lagrangeInterpolation.

Here is the call graph for this function:

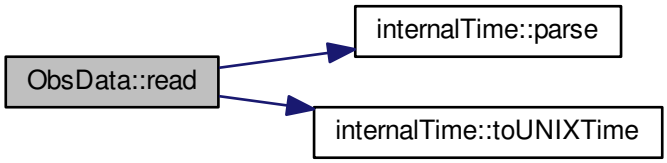


3.4.3.6 void ObsData::read ( )

Member function read.

Member function read parses input observation files and constructs internal observation structure.

Here is the call graph for this function:



3.4.3.7 void ObsData::setArcStartEnd ( ) [private]

Sets Arc pointers using ptr_pair objects.

This function sets Arc pointers to start/end pairs using ptr_pair, which serve as input arcs to preprocessing phase.

3.4.3.8 void ObsData::setSysFlags (std::string *sysString*) [private]

Sets system flags.

This function sets system flags given sysString, to indicate which constellations are processed.

Parameters

sysString	string (any combination of 'G','R','E','C') indicating constellations being processed.

3.4.4 Member Data Documentation

3.4.4.1 std::vector<ptr_pair> ObsData::arcs

Initial non-zero arc pinters.

ptr_pair Object containing Initial non-zero arcs, without preprocessing being applied.

3.4.4.2 std::vector<ptr_pair> ObsData::arcs2

Arc pointers without zeros.

ptr_pair Object containing arcs, without leading and trailing zeros.

3.4.4.3 std::vector<ptr_pair> ObsData::arcs3

Arc pointers without gaps.

ptr_pair Object containing arcs, with gaps removoed by lagrangeInterpolation and phase jumps removed. These are the processed Arcs.

3.4.4.4 double* ObsData::B

Stores matrix B.

This is stored matrix B. B is a boolean matrix relating each value in vector S to a given arc number. The i^{th} row of B has only one non-zero in the j^{th} column, relating i^{th} value in vector S to j^{th} arc number defined by S_arcnum. Size of B is (size_of_S x numArcs).

3.4.4.5 std::vector< std::vector<float> > ObsData::BDU_ucTEC

Vectors to store raw non-calibrated TEC for BeiDou Satellites.

This is a Vector of float-vectors, where first index is the Satellite prn-id and the second index is the raw non-calibrated TEC for BeiDou satellites corresponding to the epoch index in timeline_main.

3.4.4.6 std::vector< std::vector<float> > ObsData::GAL_ucTEC

Vectors to store raw non-calibrated TEC for Galileo Satellites.

This is a Vector of float-vectors, where first index is the Satellite prn-id and the second index is the raw non-calibrated TEC for Galileo satellites corresponding to the epoch index in timeline_main.

3.4.4.7 std::vector< std::vector<float> > ObsData::GLO_ucTEC

Vectors to store raw non-calibrated TEC for GLONASS Satellites.

This is a Vector of float-vectors, where first index is the Satellite prn-id and the second index is the raw non-calibrated TEC for GLONASS satellites corresponding to the epoch index in timeline_main.

3.4.4.8 std::vector< std::vector<float> > ObsData::GPS_ucTEC

Vectors to store raw non-calibrated TEC for GPS Satellites.

This is a Vector of float-vectors, where first index is the Satellite prn-id and the second index is the raw non-calibrated TEC for GPS satellites corresponding to the epoch index in timeline_main.

3.4.4.9 bool ObsData::hasTOFO

Time of first observation flag.

Flag to indicate whether Time of first observation was present in observation Header.

3.4.4.10 int ObsData::numArcs

Indicates total number of arcs.

Indicates total number of arcs formed. Arc numbers are defined by pre_processing phase using pre_process.

3.4.4.11 std::vector<double> ObsData::S

Stores vector S (non-calibrated TEC).

This vector stores all computed non-calibrated TEC values, arranged by epochs. This is the input vector given to the system solver.

3.4.4.12 std::vector<int> ObsData::S_arcnum

Stores arc numbers for S.

This vector stores for each element in S , a corresponding value indicating the its arc number. Arc numbers are defined by pre_processing phase using pre_process.

3.4.4.13 std::vector<int> ObsData::S_prn

Stores Satellite IDs for S.

This vector stores for each element in S , a corresponding value indicating the its Satellite ID.

The documentation for this class was generated from the following files:

- ObsData.hpp
- ObsData.cpp

3.5 ptr_pair Class Reference

`#include <ptr_pair.hpp>`

Public Member Functions

- ptr_pair ()
  Default constructor.
- ptr_pair (float *s, float *e)
  Custom constructor.

Public Attributes

- float * start
  Start pointer.
- float * end
  End pointer.

3.5.1 Detailed Description

Author

Muhammad Owais

Date

05/12/16

3.5.2 Constructor & Destructor Documentation

3.5.2.1 ptr_pair::ptr_pair ( )

Default constructor.

Default constructur, creates ptr_pair object with NULLL start and end pointers.

3.5.2.2 ptr_pair::ptr_pair (float * *s*, float * *e*)

Custom constructor.

Constructur, creates ptr_pair object with start and end pointers set to given pointers.

Parameters

s	Input start pointer for new ptr_pair object.
e	Input end pointer for new ptr_pair object.

The documentation for this class was generated from the following files:

- ptr_pair.hpp
- ptr_pair.cpp

3.6 triple Class Reference

`#include <triple.hpp>`

Public Member Functions

- void dump (std::ostream &s)
  Member function dump.
- triple ()
  Default constructor.
- triple (const double &x, const double &y, const double &z)
  Custom constructor.

Public Attributes

- double X
  Stores X Coordinate.
- double Y
  Stores Y Coordinate.
- double Z
  Stores Z Coordinate.

3.6.1 Detailed Description

Author

Muhammad Owais

Date

05/12/16

3.6.2 Constructor & Destructor Documentation

3.6.2.1 triple::triple ( )

Default constructor.

Constructor creating triple object initialized to zero.

3.6.2.2 triple::triple (const double & *x*, const double & *y*, const double & *z*)

Custom constructor.

Constructor creating triple object with three doubles given as input.

Parameters

x	X Coordinate
y	Y Coordinate
z	Z Coordinate

3.6.3 Member Function Documentation

3.6.3.1 void triple::dump (std::ostream & *s*)

Member function dump.

Member function dump output coordinates into a given output stream.

Parameters

s	output stream

The documentation for this class was generated from the following files:

- triple.hpp
- triple.cpp

Chapter 4

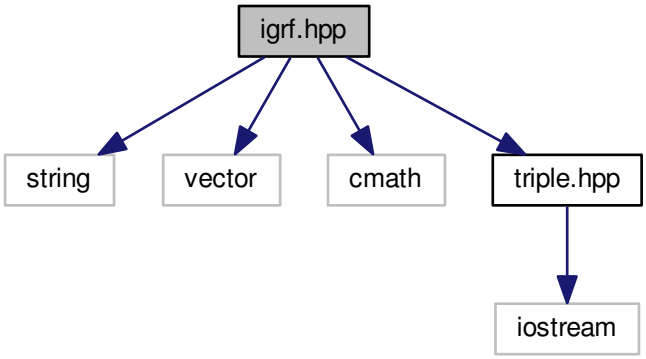
File Documentation

4.1 igrf.hpp File Reference

This class implements IGRF model.

```
#include <string>
#include <vector>
#include <cmath>
#include "triple.hpp"
```

Include dependency graph for igrf.hpp:



Classes

- class igrf

4.1.1 Detailed Description

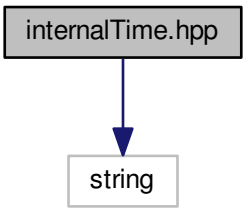
This class implements IGRF model.

This class implements IGRF (International Geomagnetic Reference Field) model, as defined in IGRF Web Site. An instance of this class could be created using a generation of IGRF coefficients, currently IGRF-12 which would be valid for years 1900 to 2020.

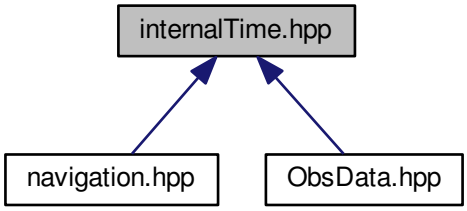
4.2 internalTime.hpp File Reference

Class defining internal time format.

`#include <string>`
Include dependency graph for internalTime.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class internalTime

4.2.1 Detailed Description

Class defining internal time format.

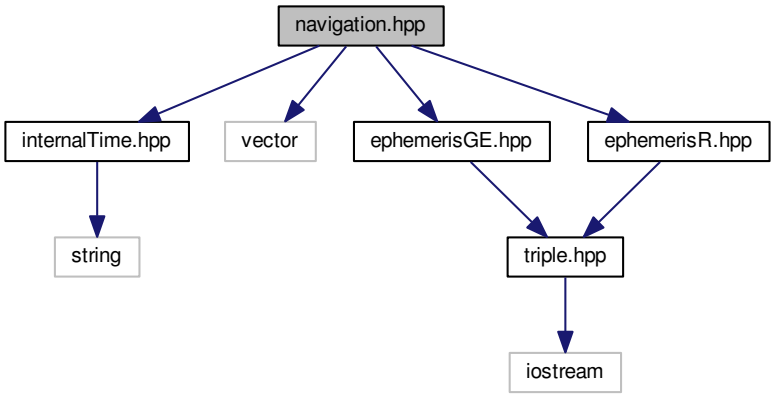
This Class Defines Internal time which is based on Unix Time. It stores the normal Date/Time as (Year,Month,Day,Hour,Minute,Second), while also providing equivalent UNIX Time. An instance of this class could be generated by explicitely providing normal Date/Time values or by providing a string which would be parse to store time in both formats.

4.3 navigation.hpp File Reference

This is class navigation data.

```
#include "internalTime.hpp"
#include <vector>
#include "ephemerisGE.hpp"
#include "ephemerisR.hpp"
```

Include dependency graph for navigation.hpp:



Classes

- class navigation

4.3.1 Detailed Description

This is class navigation data.

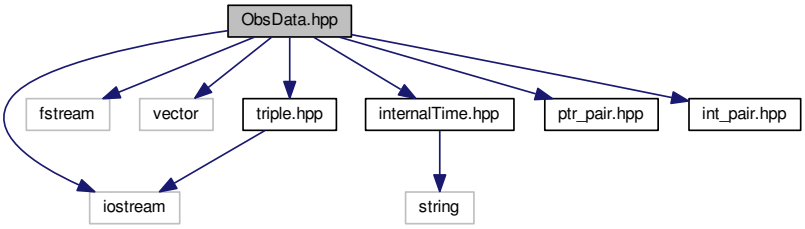
This class defines navigation data, stored after reading RINEX navigation files, for different constellations.

4.4 ObsData.hpp File Reference

Class defining observation data.

```
#include <iostream>
#include <fstream>
#include <vector>
#include "internalTime.hpp"
#include "triple.hpp"
#include "ptr_pair.hpp"
#include "int_pair.hpp"
```

Include dependency graph for ObsData.hpp:



Classes

- class ObsData

4.4.1 Detailed Description

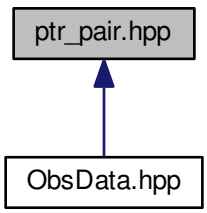
Class defining observation data.

This Class Defines observation data handling, including reading from observation files and storing in internal data structure, the raw non-calibrated TEC from phase observables. This class also includes preprocessing routines being applied to internal data structure, and allot of dump routines for debugging and ploting arc states.

4.5 ptr_pair.hpp File Reference

Class defining pointer pairs.

This graph shows which files directly or indirectly include this file:



Classes

- class ptr_pair

4.5.1 Detailed Description

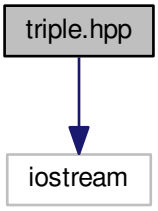
Class defining pointer pairs.

This Class Defines pointer pairs objects used in preprocessing to define arcs. Each arc coud be defined as a ptr_pair object having a start pointer (pointer to first value in arc) and an end pointer (pointer to last value in arc).

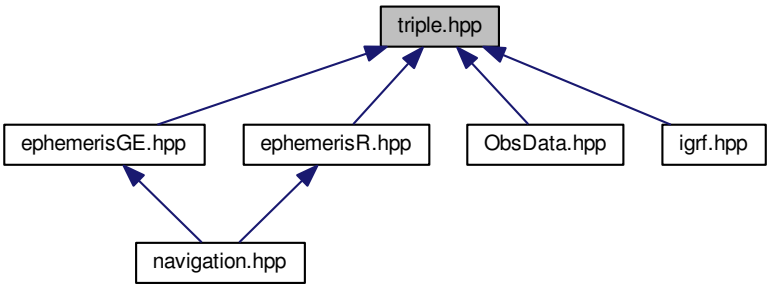
4.6 triple.hpp File Reference

This class defines a 3-D Coordinate.

`#include <iostream>`
Include dependency graph for triple.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class triple

4.6.1 Detailed Description

This class defines a 3-D Coordinate.

Index