Deliverable 3

1.0

Generated by Doxygen 1.8.17

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 boost Namespace Reference

Namespaces

- test_tools

5.2 boost::test_tools Namespace Reference

Namespaces

- tt_detail

5.3 boost::test_tools::tt_detail Namespace Reference

Classes

- struct print_log_value< std::vector< std::string > >

5.4 GPS Namespace Reference

Namespaces

- Earth
- LogFiles

Classes

- class Position
- class Route
- class Track

Typedefs

- using degrees = double
- using radians = double
- using metres = double
- using seconds = unsigned long long int
- using speed = double

Functions

- radians degToRad (degrees)
- radians radToDeg (degrees)
- double sinSqr (radians)
- degrees normaliseDeg (degrees)
- degrees ddmTodd (std::string)

Variables

- const double pi = 3.141592653589793
- const degrees fullRotation = 360
- const degrees halfRotation = fullRotation/2
- const degrees poleLatitude = fullRotation/4
- const degrees antiMeridianLongitude = fullRotation/2

5.4.1 Typedef Documentation

5.4.1.1 degrees

```
using GPS::degrees = typedef double
```

5.4.1.2 metres

```
using GPS::metres = typedef double
```

5.4.1.3 radians

```
using GPS::radians = typedef double
```

5.4.1.4 seconds

```
using GPS::seconds = typedef unsigned long long int
```

5.4.1.5 speed

```
using GPS::speed = typedef double
```

5.4.2 Function Documentation

5.4.2.1 ddmTodd()

```
degrees GPS::ddmTodd (
            std::string ddmStr )
```

5.4.2.2 degToRad()

```
radians GPS::degToRad (
            degrees d )
```

5.4.2.3 normaliseDeg()

```
degrees GPS::normaliseDeg (
            degrees d )
```

5.4.2.4 radToDeg()

```
degrees GPS::radToDeg (
            degrees r )
```

5.4.2.5 sinSqr()

```
double GPS::sinSqr (
            radians x )
```

5.4.3 Variable Documentation

5.4.3.1 antiMeridianLongitude

```
const degrees GPS::antiMeridianLongitude = fullRotation/2
```

5.4.3.2 fullRotation

```
const degrees GPS::fullRotation = 360
```

5.4.3.3 halfRotation

```
const degrees GPS::halfRotation = fullRotation/2
```

5.4.3.4 pi

```
const double GPS::pi = 3.141592653589793
```

5.4.3.5 poleLatitude

```
const degrees GPS::poleLatitude = fullRotation/4
```

5.5 GPS::Earth Namespace Reference

Functions

- degrees latitudeSubtendedBy (metres)
- degrees longitudeSubtendedBy (metres, degrees lat)

Variables

- const Position NorthPole = Position(poleLatitude,0,0)
- const Position EquatorialMeridian = Position(0,0,0)
- const Position EquatorialAntiMeridian = Position(0,antiMeridianLongitude,0)
- const Position CliftonCampus = Position(52.91249953,-1.18402513,58)
- const Position CityCampus = Position(52.9581383,-1.1542364,53)
- const Position Pontianak = Position(0,109.322134,0)
- const metres meanRadius = 6371008.8
- const metres equatorialCircumference = 40075160
- const metres polarCircumference = 40008000

5.5.1 Function Documentation

5.5.1.1 latitudeSubtendedBy()

```
degrees GPS::Earth::latitudeSubtendedBy (
            metres distance )
```

5.5.1.2 longitudeSubtendedBy()

```
degrees GPS::Earth::longitudeSubtendedBy (
            metres distance,
            degrees lat )
```

5.5.2 Variable Documentation

5.5.2.1 CityCampus

```
const Position GPS::Earth::CityCampus = Position(52.9581383,-1.1542364,53)
```

5.5.2.2 CliftonCampus

```
const Position GPS::Earth::CliftonCampus = Position(52.91249953,-1.18402513,58)
```

5.5.2.3 EquatorialAntiMeridian

```
const Position GPS::Earth::EquatorialAntiMeridian = Position(0,antiMeridianLongitude,0)
```

5.5.2.4 equatorialCircumference

```
const metres GPS::Earth::equatorialCircumference = 40075160
```

5.5.2.5 EquatorialMeridian

```
const Position GPS::Earth::EquatorialMeridian = Position(0,0,0)
```

5.5.2.6 meanRadius

```
const metres GPS::Earth::meanRadius = 6371008.8
```

5.5.2.7 NorthPole

```
const Position GPS::Earth::NorthPole = Position(poleLatitude,0,0)
```

5.5.2.8 polarCircumference

```
const metres GPS::Earth::polarCircumference = 40008000
```

5.5.2.9 Pontianak

```
const Position GPS::Earth::Pontianak = Position(0,109.322134,0)
```

5.6 GPS::LogFiles Namespace Reference

Variables

- const std::string logsDir = "../logs/"
- const std::string NMEALogsDir = logsDir + "NMEA/"
- const std::string GPXRoutesDir = logsDir + "GPX/routes/"
- const std::string GPXTracksDir = logsDir + "GPX/tracks/"

5.6.1 Variable Documentation

5.6.1.1 GPXRoutesDir

```
const std::string GPS::LogFiles::GPXRoutesDir = logsDir + "GPX/routes/"
```

5.6.1.2 GPXTracksDir

```
const std::string GPS::LogFiles::GPXTracksDir = logsDir + "GPX/tracks/"
```

5.6.1.3 logsDir

```
const std::string GPS::LogFiles::logsDir = "../logs/"
```

5.6.1.4 NMEALogsDir

```
const std::string GPS::LogFiles::NMEALogsDir = logsDir + "NMEA/"
```

5.7 NMEA Namespace Reference

Typedefs

- using SentenceData = std::pair< std::string, std::vector< std::string > >
- using Route = std::vector< GPS::Position >

Functions

- bool isWellFormedSentence (std::string)
- bool hasValidChecksum (std::string)
- SentenceData extractSentenceData (std::string)
- GPS::Position positionFromSentenceData (SentenceData)
- Route routeFromLog (std::istream &)

5.7.1 Typedef Documentation

5.7.1.1 Route

```
using NMEA::Route = typedef std::vector<GPS::Position>
```

5.7.1.2 SentenceData

```
using NMEA::SentenceData = typedef std::pair<std::string, std::vector<std::string> >
```

5.7.2 Function Documentation

5.7.2.1 extractSentenceData()

```
SentenceData NMEA::extractSentenceData (
            std::string  )
```

5.7.2.2 hasValidChecksum()

```
bool NMEA::hasValidChecksum (
            std::string  )
```

5.7.2.3 isWellFormedSentence()

```
bool NMEA::isWellFormedSentence (
            std::string  )
```

5.7.2.4 positionFromSentenceData()

```
GPS::Position NMEA::positionFromSentenceData (
            SentenceData  )
```

5.7.2.5 routeFromLog()

```
Route NMEA::routeFromLog (
            std::istream &  )
```

Chapter 6

Class Documentation

6.1 GPS::Position Class Reference

`#include <position.h>`

Public Member Functions

- Position (degrees lat, degrees lon, metres ele=0.0)
- Position (std::string latStr, std::string lonStr, std::string eleStr="0")
- Position (std::string ddmLatStr, char northing, std::string ddmLonStr, char easting, std::string eleSt="0")
- degrees latitude () const
- degrees longitude () const
- metres elevation () const
- std::string toString (bool includeElevation=true) const

Static Public Member Functions

- static metres distanceBetween (Position, Position)

6.1.1 Constructor & Destructor Documentation

6.1.1.1 Position() [1/3]

```
GPS::Position::Position (
            degrees lat,
            degrees lon,
            metres ele = 0.0 )
```

6.1.1.2 Position() [2/3]

```
GPS::Position::Position (
            std::string latStr,
            std::string lonStr,
            std::string eleStr = "0" )
```

6.1.1.3 Position() [3/3]

```
GPS::Position::Position (
            std::string ddmLatStr,
            char northing,
            std::string ddmLonStr,
            char easting,
            std::string eleSt = "0" )
```

6.1.2 Member Function Documentation

6.1.2.1 distanceBetween()

```
metres GPS::Position::distanceBetween (
            Position p1,
            Position p2 )  [static]
```

6.1.2.2 elevation()

```
metres GPS::Position::elevation ( ) const
```

6.1.2.3 latitude()

```
degrees GPS::Position::latitude ( ) const
```

6.1.2.4 longitude()

```
degrees GPS::Position::longitude ( ) const
```

6.1.2.5 toString()

```
std::string GPS::Position::toString (
            bool includeElevation = true ) const
```

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

- position.h
- position.cpp

6.2 boost::test_tools::tt_detail::print_log_value< std::vector< std::string > > Struct Reference

Public Member Functions

- void operator() (std::ostream &os, std::vector< std::string > const &v)

6.2.1 Member Function Documentation

6.2.1.1 operator()()

```
void boost::test_tools::tt_detail::print_log_value< std::vector< std::string > >::operator()
(
            std::ostream & os,
            std::vector< std::string > const & v )  [inline]
```

The documentation for this struct was generated from the following file:

- nmea-tests.cpp

6.3 GPS::Route Class Reference

```
#include <route.h>
```

Inheritance diagram for GPS::Route:

Public Member Functions

- Route (std::string source, bool isFileName, metres granularity=20)
- virtual void setGranularity (metres)
- std::string name () const
- unsigned int numPositions () const
- metres totalLength () const
- metres netLength () const
- metres totalHeightGain () const
- metres netHeightGain () const
- degrees maxGradient () const
- degrees minGradient () const
- degrees steepestGradient () const
- degrees minLatitude () const
- degrees maxLatitude () const
- degrees minLongitude () const
- degrees maxLongitude () const
- metres minElevation () const
- metres maxElevation () const
- Position operator[ ] (unsigned int) const
- Position findPosition (std::string soughtName) const
- std::string findNameOf (Position) const
- unsigned int timesVisited (std::string soughtName) const
- unsigned int timesVisited (Position) const
- bool containsCycles () const

Protected Member Functions

- Route ()
- bool areSameLocation (Position, Position) const

Protected Attributes

- std::vector< Position > positions
- std::vector< std::string > positionNames
- std::string routeName
- metres routeLength
- metres granularity

6.3.1 Constructor & Destructor Documentation

6.3.1.1 Route() [1/2]

```
Route::Route (
            std::string source,
            bool isFileName,
            metres granularity = 20 )
```

6.3.1.2 Route() [2/2]

```
GPS::Route::Route ( )  [inline], [protected]
```

6.3.2 Member Function Documentation

6.3.2.1 areSameLocation()

```
bool Route::areSameLocation (
            Position p1,
            Position p2 ) const  [protected]
```

6.3.2.2 containsCycles()

```
bool Route::containsCycles ( ) const
```

6.3.2.3 findNameOf()

```
std::string Route::findNameOf (
            Position soughtPos ) const
```

6.3.2.4 findPosition()

```
Position Route::findPosition (
            std::string soughtName ) const
```

6.3.2.5 maxElevation()

```
metres Route::maxElevation ( ) const
```

6.3.2.6 maxGradient()

```
degrees Route::maxGradient ( ) const
```

6.3.2.7 maxLatitude()

```
degrees Route::maxLatitude ( ) const
```

6.3.2.8 maxLongitude()

```
degrees Route::maxLongitude ( ) const
```

6.3.2.9 minElevation()

```
metres Route::minElevation ( ) const
```

6.3.2.10 minGradient()

```
degrees Route::minGradient ( ) const
```

6.3.2.11 minLatitude()

```
degrees Route::minLatitude ( ) const
```

6.3.2.12 minLongitude()

```
degrees Route::minLongitude ( ) const
```

6.3.2.13 name()

```
std::string Route::name ( ) const
```

6.3.2.14 netHeightGain()

```
metres Route::netHeightGain ( ) const
```

6.3.2.15 netLength()

```
metres Route::netLength ( ) const
```

6.3.2.16 numPositions()

```
unsigned int Route::numPositions ( ) const
```

6.3.2.17 operator[]()

```
Position Route::operator[] (
            unsigned int idx ) const
```

6.3.2.18 setGranularity()

```
void Route::setGranularity (
            metres  )  [virtual]
```

Reimplemented in GPS::Track.

6.3.2.19 steepestGradient()

```
degrees Route::steepestGradient ( ) const
```

6.3.2.20 timesVisited() [1/2]

```
unsigned int Route::timesVisited (
            Position soughtPos ) const
```

6.3.2.21 timesVisited() [2/2]

```
unsigned int Route::timesVisited (
            std::string soughtName ) const
```

6.3.2.22 totalHeightGain()

```
metres Route::totalHeightGain ( ) const
```

6.3.2.23 totalLength()

```
metres Route::totalLength ( ) const
```

6.3.3 Member Data Documentation

6.3.3.1 granularity

```
metres GPS::Route::granularity  [protected]
```

6.3.3.2 positionNames

```
std::vector<std::string> GPS::Route::positionNames  [protected]
```

6.3.3.3 positions

```
std::vector<Position> GPS::Route::positions  [protected]
```

6.3.3.4 routeLength

```
metres GPS::Route::routeLength  [protected]
```

6.3.3.5 routeName

```
std::string GPS::Route::routeName  [protected]
```

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

- route.h
- route.cpp

6.4 GPS::Track Class Reference

`#include <track.h>`

Inheritance diagram for GPS::Track:

Collaboration diagram for GPS::Track:

Public Member Functions

- Track (std::string source, bool isFileName, metres granularity=10)
- void setGranularity (metres) override
- seconds totalTime () const
- seconds travellingTime () const
- seconds restingTime () const
- seconds longestRest () const
- speed maxSpeed () const
- speed averageSpeed (bool includeRests) const
- speed maxRateOfAscent () const
- speed maxRateOfDescent () const

Protected Attributes

- std::vector< seconds > arrived
- std::vector< seconds > departed

Additional Inherited Members

6.4.1 Constructor & Destructor Documentation

6.4.1.1 Track()

```
Track::Track (
            std::string source,
            bool isFileName,
            metres granularity = 10 )
```

6.4.2 Member Function Documentation

6.4.2.1 averageSpeed()

```
speed Track::averageSpeed (
            bool includeRests ) const
```

6.4.2.2 longestRest()

```
seconds Track::longestRest ( ) const
```

6.4.2.3 maxRateOfAscent()

```
speed Track::maxRateOfAscent ( ) const
```

6.4.2.4 maxRateOfDescent()

```
speed Track::maxRateOfDescent ( ) const
```

6.4.2.5 maxSpeed()

```
speed Track::maxSpeed ( ) const
```

6.4.2.6 restingTime()

```
seconds Track::restingTime ( ) const
```

6.4.2.7 setGranularity()

```
void Track::setGranularity (
            metres  )  [override], [virtual]
```

Reimplemented from GPS::Route.

6.4.2.8 totalTime()

```
seconds Track::totalTime ( ) const
```

6.4.2.9 travellingTime()

```
seconds Track::travellingTime ( ) const
```

6.4.3 Member Data Documentation

6.4.3.1 arrived

```
std::vector<seconds> GPS::Track::arrived  [protected]
```

6.4.3.2 departed

```
std::vector<seconds> GPS::Track::departed  [protected]
```

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

- track.h
- track.cpp

Chapter 7

File Documentation

7.1 earth.cpp File Reference

```
#include <cmath>
#include "geometry.h"
#include "earth.h"
```
Include dependency graph for earth.cpp:

7.2 earth.h File Reference

```
#include "position.h"
```
Include dependency graph for earth.h: This graph shows which files directly or indirectly include this file:

Namespaces

- GPS
- GPS::Earth

Functions

- degrees GPS::Earth::latitudeSubtendedBy (metres)
- degrees GPS::Earth::longitudeSubtendedBy (metres, degrees lat)

Variables

- const Position GPS::Earth::NorthPole = Position(poleLatitude,0,0)
- const Position GPS::Earth::EquatorialMeridian = Position(0,0,0)
- const Position GPS::Earth::EquatorialAntiMeridian = Position(0,antiMeridianLongitude,0)
- const Position GPS::Earth::CliftonCampus = Position(52.91249953,-1.18402513,58)
- const Position GPS::Earth::CityCampus = Position(52.9581383,-1.1542364,53)
- const Position GPS::Earth::Pontianak = Position(0,109.322134,0)
- const metres GPS::Earth::meanRadius = 6371008.8
- const metres GPS::Earth::equatorialCircumference = 40075160
- const metres GPS::Earth::polarCircumference = 40008000

7.3 geometry.cpp File Reference

```
#include <cmath>
#include "geometry.h"
```
Include dependency graph for geometry.cpp:

Namespaces

- GPS

Functions

- radians GPS::degToRad (degrees)
- radians GPS::radToDeg (degrees)
- double GPS::sinSqr (radians)
- degrees GPS::normaliseDeg (degrees)

7.4 geometry.h File Reference

```
#include "types.h"
```
Include dependency graph for geometry.h: This graph shows which files directly or indirectly include this file:

Namespaces

- GPS

Functions

- radians GPS::degToRad (degrees)
- radians GPS::radToDeg (degrees)
- double GPS::sinSqr (radians)
- degrees GPS::normaliseDeg (degrees)

Variables

- const double GPS::pi = 3.141592653589793
- const degrees GPS::fullRotation = 360
- const degrees GPS::halfRotation = fullRotation/2
- const degrees GPS::poleLatitude = fullRotation/4
- const degrees GPS::antiMeridianLongitude = fullRotation/2

7.5 gpx-tests.cpp File Reference

```
#include <boost/test/unit_test.hpp>
```
Include dependency graph for gpx-tests.cpp:

Macros

- #define BOOST_TEST_DYN_LINK
- #define BOOST_TEST_MODULE GPXTests

7.5.1 Macro Definition Documentation

7.5.1.1 BOOST_TEST_DYN_LINK

```
#define BOOST_TEST_DYN_LINK
```

7.5.1.2 BOOST_TEST_MODULE

```
#define BOOST_TEST_MODULE GPXTests
```

7.6 logs.cpp File Reference

```
#include <string>
#include "logs.h"
```

Include dependency graph for logs.cpp:

Namespaces

- GPS
- GPS::LogFiles

7.7 logs.h File Reference

```
#include <string>
```

Include dependency graph for logs.h: This graph shows which files directly or indirectly include this file:

Namespaces

- GPS
- GPS::LogFiles

Variables

- const std::string GPS::LogFiles::logsDir = "../logs/"
- const std::string GPS::LogFiles::NMEALogsDir = logsDir + "NMEA/"
- const std::string GPS::LogFiles::GPXRoutesDir = logsDir + "GPX/routes/"
- const std::string GPS::LogFiles::GPXTracksDir = logsDir + "GPX/tracks/"

7.8 nmea-tests.cpp File Reference

```
#include <boost/test/unit_test.hpp>
#include <string>
#include <stdexcept>
#include <vector>
#include <utility>
#include <ostream>
#include <fstream>
#include <sstream>
#include <iostream>
#include "logs.h"
#include "parseNMEA.h"
```

Include dependency graph for nmea-tests.cpp:

Classes

- struct boost::test_tools::tt_detail::print_log_value< std::vector< std::string > >

Namespaces

- boost
- boost::test_tools
- boost::test_tools::tt_detail

Macros

- #define BOOST_TEST_DYN_LINK
- #define BOOST_TEST_MODULE ParseNMEATests

Functions

- BOOST_AUTO_TEST_CASE (WellFormedNoFields)
- BOOST_AUTO_TEST_CASE (WellFormedOneField)
- BOOST_AUTO_TEST_CASE (WellFormedTwoFields)
- BOOST_AUTO_TEST_CASE (WellFormedEmptyField)
- BOOST_AUTO_TEST_CASE (WellFormedManyFields)
- BOOST_AUTO_TEST_CASE (WellFormedWithLowercaseHexCharacters)
- BOOST_AUTO_TEST_CASE (WellFormedWithUppercaseHexCharacters)
- BOOST_AUTO_TEST_CASE (WellFormedTypicalSentences)
- BOOST_AUTO_TEST_CASE (IllFormedMissingSuffix)
- BOOST_AUTO_TEST_CASE (IllFormedDollar)

- BOOST_AUTO_TEST_CASE (IllFormedGP)
- BOOST_AUTO_TEST_CASE (IllFormedType)
- BOOST_AUTO_TEST_CASE (IllFormedReservedCharInField)
- BOOST_AUTO_TEST_CASE (IllFormedStar)
- BOOST_AUTO_TEST_CASE (IllFormedChecksum)
- BOOST_AUTO_TEST_CASE (ValidChecksumMinimalSentence)
- BOOST_AUTO_TEST_CASE (IncorrectChecksumMinimalSentence)
- BOOST_AUTO_TEST_CASE (ValidChecksumTypicalSentences)
- BOOST_AUTO_TEST_CASE (IncorrectChecksumTypicalSentences)
- BOOST_AUTO_TEST_CASE (CorrectChecksumWithUppercaseHexDigit)
- BOOST_AUTO_TEST_CASE (CorrectChecksumWithLowercaseHexDigit)
- BOOST_AUTO_TEST_CASE (IncorrectChecksumsithUppercaseHexDigit)
- BOOST_AUTO_TEST_CASE (IncorrectChecksumWithLowercaseHexDigit)
- BOOST_AUTO_TEST_CASE (ExtractZeroFields)
- BOOST_AUTO_TEST_CASE (ExtractOneField)
- BOOST_AUTO_TEST_CASE (ExtractTwoFields)
- BOOST_AUTO_TEST_CASE (ExtractGLL)
- BOOST_AUTO_TEST_CASE (ExtractGGA)
- BOOST_AUTO_TEST_CASE (ExtractRMC)
- BOOST_AUTO_TEST_CASE (ExtractMSS)
- BOOST_AUTO_TEST_CASE (GLL_NW)
- BOOST_AUTO_TEST_CASE (GLL_NE)
- BOOST_AUTO_TEST_CASE (GLL_SE)
- BOOST_AUTO_TEST_CASE (GLL_SW)
- BOOST_AUTO_TEST_CASE (RMC_NW)
- BOOST_AUTO_TEST_CASE (RMC_NE)
- BOOST_AUTO_TEST_CASE (RMC_SE)
- BOOST_AUTO_TEST_CASE (RMC_SW)
- BOOST_AUTO_TEST_CASE (GGA_NW)
- BOOST_AUTO_TEST_CASE (GGA_NE)
- BOOST_AUTO_TEST_CASE (GGA_SE)
- BOOST_AUTO_TEST_CASE (GGA_SW)
- BOOST_AUTO_TEST_CASE (GGA_NegativeElevation)
- BOOST_AUTO_TEST_CASE (UnsupportedFormat)
- BOOST_AUTO_TEST_CASE (EmptyFieldVector)
- BOOST_AUTO_TEST_CASE (MissingFieldsGLL)
- BOOST_AUTO_TEST_CASE (MissingFieldsRMC)
- BOOST_AUTO_TEST_CASE (MissingFieldsGGA)
- BOOST_AUTO_TEST_CASE (InvalidFieldData)
- BOOST_AUTO_TEST_CASE (EmptyLog)
- BOOST_AUTO_TEST_CASE (LogWithOneValidSentence)
- BOOST_AUTO_TEST_CASE (LogWithTwoValidSentences)
- BOOST_AUTO_TEST_CASE (LogWithBlankLines)
- BOOST_AUTO_TEST_CASE (LogWithoutLineBreaks)
- BOOST_AUTO_TEST_CASE (LogWithIllFormedSentences)
- BOOST_AUTO_TEST_CASE (LogWithInvalidChecksums)
- BOOST_AUTO_TEST_CASE (LogWithUnsupportedSentenceTypes)
- BOOST_AUTO_TEST_CASE (LogWithMissingFields)
- BOOST_AUTO_TEST_CASE (LogWithInvalidFields)
- BOOST_AUTO_TEST_CASE (LargeLog_GLL)
- BOOST_AUTO_TEST_CASE (LargeLog_GGA_RMC)

Variables

- const double epsilon = 0.0001
- const double percentageAccuracy = 0.0001
- const std::string validGLLSentence = "$GPGLL,5425.31,N,107.03,W,82610∗69"
- const std::string validRMCSentence = "$GPRMC,113922.000,A,3722.5993,N,00559.2458,W,0.000,0.↩ 00,150914,,A∗62"
- const std::string validMSSSentence = "$GPMSS,55,27,318.0,100,∗66"
- const GPS::Position gllPos = GPS::Position("5425.31",'N',"107.03",'W')
- const GPS::Position rmcPos = GPS::Position("3722.5993",'N',"00559.2458",'W')

7.8.1 Macro Definition Documentation

7.8.1.1 BOOST_TEST_DYN_LINK

```
#define BOOST_TEST_DYN_LINK
```

7.8.1.2 BOOST_TEST_MODULE

```
#define BOOST_TEST_MODULE ParseNMEATests
```

7.8.2 Function Documentation

7.8.2.1 BOOST_AUTO_TEST_CASE() [1/61]

```
BOOST_AUTO_TEST_CASE (
            CorrectChecksumWithLowercaseHexDigit  )
```

7.8.2.2 BOOST_AUTO_TEST_CASE() [2/61]

```
BOOST_AUTO_TEST_CASE (
            CorrectChecksumWithUppercaseHexDigit  )
```

7.8.2.3 BOOST_AUTO_TEST_CASE() [3/61]

```
BOOST_AUTO_TEST_CASE (
            EmptyFieldVector  )
```

7.8.2.4 BOOST_AUTO_TEST_CASE() [4/61]

```
BOOST_AUTO_TEST_CASE (
            EmptyLog  )
```

7.8.2.5 BOOST_AUTO_TEST_CASE() [5/61]

```
BOOST_AUTO_TEST_CASE (
            ExtractGGA  )
```

7.8.2.6 BOOST_AUTO_TEST_CASE() [6/61]

```
BOOST_AUTO_TEST_CASE (
            ExtractGLL  )
```

7.8.2.7 BOOST_AUTO_TEST_CASE() [7/61]

```
BOOST_AUTO_TEST_CASE (
            ExtractMSS  )
```

7.8.2.8 BOOST_AUTO_TEST_CASE() [8/61]

```
BOOST_AUTO_TEST_CASE (
            ExtractOneField  )
```

7.8.2.9 BOOST_AUTO_TEST_CASE() [9/61]

```
BOOST_AUTO_TEST_CASE (
            ExtractRMC  )
```

7.8.2.10 BOOST_AUTO_TEST_CASE() [10/61]

```
BOOST_AUTO_TEST_CASE (
             ExtractTwoFields  )
```

7.8.2.11 BOOST_AUTO_TEST_CASE() [11/61]

```
BOOST_AUTO_TEST_CASE (
             ExtractZeroFields  )
```

7.8.2.12 BOOST_AUTO_TEST_CASE() [12/61]

```
BOOST_AUTO_TEST_CASE (
             GGA_NE  )
```

7.8.2.13 BOOST_AUTO_TEST_CASE() [13/61]

```
BOOST_AUTO_TEST_CASE (
             GGA_NegativeElevation  )
```

7.8.2.14 BOOST_AUTO_TEST_CASE() [14/61]

```
BOOST_AUTO_TEST_CASE (
             GGA_NW  )
```

7.8.2.15 BOOST_AUTO_TEST_CASE() [15/61]

```
BOOST_AUTO_TEST_CASE (
             GGA_SE  )
```

7.8.2.16 BOOST_AUTO_TEST_CASE() [16/61]

```
BOOST_AUTO_TEST_CASE (
             GGA_SW  )
```

7.8.2.17 BOOST_AUTO_TEST_CASE() [17/61]

```
BOOST_AUTO_TEST_CASE (
             GLL_NE  )
```

7.8.2.18 BOOST_AUTO_TEST_CASE() [18/61]

```
BOOST_AUTO_TEST_CASE (
             GLL_NW  )
```

7.8.2.19 BOOST_AUTO_TEST_CASE() [19/61]

```
BOOST_AUTO_TEST_CASE (
             GLL_SE  )
```

7.8.2.20 BOOST_AUTO_TEST_CASE() [20/61]

```
BOOST_AUTO_TEST_CASE (
             GLL_SW  )
```

7.8.2.21 BOOST_AUTO_TEST_CASE() [21/61]

```
BOOST_AUTO_TEST_CASE (
             IllFormedChecksum  )
```

7.8.2.22 BOOST_AUTO_TEST_CASE() [22/61]

```
BOOST_AUTO_TEST_CASE (
             IllFormedDollar  )
```

7.8.2.23 BOOST_AUTO_TEST_CASE() [23/61]

```
BOOST_AUTO_TEST_CASE (
             IllFormedGP  )
```

7.8.2.24 BOOST_AUTO_TEST_CASE() [24/61]

```
BOOST_AUTO_TEST_CASE (
            IllFormedMissingSuffix  )
```

7.8.2.25 BOOST_AUTO_TEST_CASE() [25/61]

```
BOOST_AUTO_TEST_CASE (
            IllFormedReservedCharInField  )
```

7.8.2.26 BOOST_AUTO_TEST_CASE() [26/61]

```
BOOST_AUTO_TEST_CASE (
            IllFormedStar  )
```

7.8.2.27 BOOST_AUTO_TEST_CASE() [27/61]

```
BOOST_AUTO_TEST_CASE (
            IllFormedType  )
```

7.8.2.28 BOOST_AUTO_TEST_CASE() [28/61]

```
BOOST_AUTO_TEST_CASE (
            IncorrectChecksumMinimalSentence  )
```

7.8.2.29 BOOST_AUTO_TEST_CASE() [29/61]

```
BOOST_AUTO_TEST_CASE (
            IncorrectChecksumsithUppercaseHexDigit  )
```

7.8.2.30 BOOST_AUTO_TEST_CASE() [30/61]

```
BOOST_AUTO_TEST_CASE (
            IncorrectChecksumTypicalSentences  )
```

7.8.2.31 BOOST_AUTO_TEST_CASE() [31/61]

```
BOOST_AUTO_TEST_CASE (
            IncorrectChecksumWithLowercaseHexDigit  )
```

7.8.2.32 BOOST_AUTO_TEST_CASE() [32/61]

```
BOOST_AUTO_TEST_CASE (
            InvalidFieldData  )
```

7.8.2.33 BOOST_AUTO_TEST_CASE() [33/61]

```
BOOST_AUTO_TEST_CASE (
            LargeLog_GGA_RMC  )
```

7.8.2.34 BOOST_AUTO_TEST_CASE() [34/61]

```
BOOST_AUTO_TEST_CASE (
            LargeLog_GLL  )
```

7.8.2.35 BOOST_AUTO_TEST_CASE() [35/61]

```
BOOST_AUTO_TEST_CASE (
            LogWithBlankLines  )
```

7.8.2.36 BOOST_AUTO_TEST_CASE() [36/61]

```
BOOST_AUTO_TEST_CASE (
            LogWithIllFormedSentences  )
```

7.8.2.37 BOOST_AUTO_TEST_CASE() [37/61]

```
BOOST_AUTO_TEST_CASE (
            LogWithInvalidChecksums  )
```

7.8.2.38 BOOST_AUTO_TEST_CASE() [38/61]

```
BOOST_AUTO_TEST_CASE (
            LogWithInvalidFields  )
```

7.8.2.39 BOOST_AUTO_TEST_CASE() [39/61]

```
BOOST_AUTO_TEST_CASE (
            LogWithMissingFields  )
```

7.8.2.40 BOOST_AUTO_TEST_CASE() [40/61]

```
BOOST_AUTO_TEST_CASE (
            LogWithOneValidSentence  )
```

7.8.2.41 BOOST_AUTO_TEST_CASE() [41/61]

```
BOOST_AUTO_TEST_CASE (
            LogWithoutLineBreaks  )
```

7.8.2.42 BOOST_AUTO_TEST_CASE() [42/61]

```
BOOST_AUTO_TEST_CASE (
            LogWithTwoValidSentences  )
```

7.8.2.43 BOOST_AUTO_TEST_CASE() [43/61]

```
BOOST_AUTO_TEST_CASE (
            LogWithUnsupportedSentenceTypes  )
```

7.8.2.44 BOOST_AUTO_TEST_CASE() [44/61]

```
BOOST_AUTO_TEST_CASE (
            MissingFieldsGGA  )
```

7.8.2.45 BOOST_AUTO_TEST_CASE() [45/61]

```
BOOST_AUTO_TEST_CASE (
             MissingFieldsGLL  )
```

7.8.2.46 BOOST_AUTO_TEST_CASE() [46/61]

```
BOOST_AUTO_TEST_CASE (
             MissingFieldsRMC  )
```

7.8.2.47 BOOST_AUTO_TEST_CASE() [47/61]

```
BOOST_AUTO_TEST_CASE (
             RMC_NE  )
```

7.8.2.48 BOOST_AUTO_TEST_CASE() [48/61]

```
BOOST_AUTO_TEST_CASE (
             RMC_NW  )
```

7.8.2.49 BOOST_AUTO_TEST_CASE() [49/61]

```
BOOST_AUTO_TEST_CASE (
             RMC_SE  )
```

7.8.2.50 BOOST_AUTO_TEST_CASE() [50/61]

```
BOOST_AUTO_TEST_CASE (
             RMC_SW  )
```

7.8.2.51 BOOST_AUTO_TEST_CASE() [51/61]

```
BOOST_AUTO_TEST_CASE (
             UnsupportedFormat  )
```

7.8.2.52 BOOST_AUTO_TEST_CASE() [52/61]

```
BOOST_AUTO_TEST_CASE (
            ValidChecksumMinimalSentence  )
```

7.8.2.53 BOOST_AUTO_TEST_CASE() [53/61]

```
BOOST_AUTO_TEST_CASE (
            ValidChecksumTypicalSentences  )
```

7.8.2.54 BOOST_AUTO_TEST_CASE() [54/61]

```
BOOST_AUTO_TEST_CASE (
            WellFormedEmptyField  )
```

7.8.2.55 BOOST_AUTO_TEST_CASE() [55/61]

```
BOOST_AUTO_TEST_CASE (
            WellFormedManyFields  )
```

7.8.2.56 BOOST_AUTO_TEST_CASE() [56/61]

```
BOOST_AUTO_TEST_CASE (
            WellFormedNoFields  )
```

7.8.2.57 BOOST_AUTO_TEST_CASE() [57/61]

```
BOOST_AUTO_TEST_CASE (
            WellFormedOneField  )
```

7.8.2.58 BOOST_AUTO_TEST_CASE() [58/61]

```
BOOST_AUTO_TEST_CASE (
            WellFormedTwoFields  )
```

7.8.2.59 BOOST_AUTO_TEST_CASE() [59/61]

```
BOOST_AUTO_TEST_CASE (
            WellFormedTypicalSentences  )
```

7.8.2.60 BOOST_AUTO_TEST_CASE() [60/61]

```
BOOST_AUTO_TEST_CASE (
            WellFormedWithLowercaseHexCharacters  )
```

7.8.2.61 BOOST_AUTO_TEST_CASE() [61/61]

```
BOOST_AUTO_TEST_CASE (
            WellFormedWithUppercaseHexCharacters  )
```

7.8.3 Variable Documentation

7.8.3.1 epsilon

```
const double epsilon = 0.0001
```

7.8.3.2 gllPos

```
const GPS::Position gllPos = GPS::Position("5425.31",'N',"107.03",'W')
```

7.8.3.3 percentageAccuracy

```
const double percentageAccuracy = 0.0001
```

7.8.3.4 rmcPos

```
const GPS::Position rmcPos = GPS::Position("3722.5993",'N',"00559.2458",'W')
```

7.8.3.5 validGLLSentence

```
const std::string validGLLSentence = "$GPGLL,5425.31,N,107.03,W,82610*69"
```

7.8.3.6 validMSSSentence

```
const std::string validMSSSentence = "$GPMSS,55,27,318.0,100,*66"
```

7.8.3.7 validRMCSentence

```
const std::string validRMCSentence = "$GPRMC,113922.000,A,3722.5993,N,00559.2458,W,0.000,0.↩
00,150914,,A*62"
```

7.9 parseNMEA.cpp File Reference

```
#include "earth.h"
#include "parseNMEA.h"
```

Include dependency graph for parseNMEA.cpp:

Namespaces

- NMEA

Functions

- bool NMEA::isWellFormedSentence (std::string)
- bool NMEA::hasValidChecksum (std::string)
- SentenceData NMEA::extractSentenceData (std::string)
- GPS::Position NMEA::positionFromSentenceData (SentenceData)
- Route NMEA::routeFromLog (std::istream &)

7.10 parseNMEA.h File Reference

```
#include <string>
#include <list>
#include <vector>
#include <utility>
#include <istream>
#include "position.h"
```

Include dependency graph for parseNMEA.h: This graph shows which files directly or indirectly include this file:

Namespaces

- NMEA

Typedefs

- using NMEA::SentenceData = std::pair< std::string, std::vector< std::string > >
- using NMEA::Route = std::vector< GPS::Position >

Functions

- bool NMEA::isWellFormedSentence (std::string)
- bool NMEA::hasValidChecksum (std::string)
- SentenceData NMEA::extractSentenceData (std::string)
- GPS::Position NMEA::positionFromSentenceData (SentenceData)
- Route NMEA::routeFromLog (std::istream &)

7.11 position.cpp File Reference

```
#include <cassert>
#include <cmath>
#include <sstream>
#include <stdexcept>
#include "geometry.h"
#include "earth.h"
#include "position.h"
```

Include dependency graph for position.cpp:

Namespaces

- GPS

Functions

- degrees GPS::ddmTodd (std::string)

7.12 position.h File Reference

```
#include <string>
#include "types.h"
```

Include dependency graph for position.h: This graph shows which files directly or indirectly include this file:

Classes

- class GPS::Position

Namespaces

- GPS

Functions

- degrees GPS::ddmTodd (std::string)

7.13 route.cpp File Reference

```
#include <sstream>
#include <fstream>
#include <iostream>
#include <cassert>
#include <cmath>
#include <algorithm>
#include <iterator>
#include <stdexcept>
#include "geometry.h"
#include "xml/element.h"
#include "xml/parser.h"
#include "route.h"
```

Include dependency graph for route.cpp:

7.14 route.h File Reference

```
#include <string>
#include <vector>
#include "types.h"
#include "position.h"
#include "xml/parser.h"
```

Include dependency graph for route.h: This graph shows which files directly or indirectly include this file:

Classes

- class GPS::Route

Namespaces

- GPS

7.15 timesVisited(string).cpp File Reference

```
#include <boost/test/unit_test.hpp>
#include "logs.h"
#include "route.h"
```

Include dependency graph for timesVisited(string).cpp:

Functions

- BOOST_AUTO_TEST_CASE (singleton_route)

  A simple route with one point and one name to check.
- BOOST_AUTO_TEST_CASE (position_not_visited)

  A simple route with one point and checking the number of times for a position that wasn't visited.
- BOOST_AUTO_TEST_CASE (singleton_route_with_spaces)

  A simple route with one point and one name to check, with leading and trailing spaces to check the constructor.
- BOOST_AUTO_TEST_CASE (bad_input_string)

  A simple route with one point and checking invalid_argument is thrown when a blank string is passed in.
- BOOST_AUTO_TEST_CASE (two_consecutive_visits)

  A simple route with two positions that are the same location.
- BOOST_AUTO_TEST_CASE (one_name_multiple_positions)

  A complex route where many positions share the same name.
- BOOST_AUTO_TEST_CASE (one_position_many_visits)

  A complex route where one position is visited many times.
- BOOST_AUTO_TEST_CASE (one_name_many_positions_and_many_visits)

  A complex route where there are many positions with the same name, visited multiple times.
- BOOST_AUTO_TEST_CASE (one_position_many_names)

  A complex route where there is one position visited multiple times, but with different names.

Variables

- const bool isFileName = false

  all data for this test suite is passed in as strings, not files.

7.15.1 Function Documentation

7.15.1.1 BOOST_AUTO_TEST_CASE() [1/9]

```
BOOST_AUTO_TEST_CASE (
            bad_input_string  )
```

A simple route with one point and checking invalid_argument is thrown when a blank string is passed in.

7.15.1.2 BOOST_AUTO_TEST_CASE() [2/9]

```
BOOST_AUTO_TEST_CASE (
            one_name_many_positions_and_many_visits  )
```

A complex route where there are many positions with the same name, visited multiple times.

7.15.1.3 BOOST_AUTO_TEST_CASE() [3/9]

```
BOOST_AUTO_TEST_CASE (
            one_name_multiple_positions  )
```

A complex route where many positions share the same name.

7.15.1.4 BOOST_AUTO_TEST_CASE() [4/9]

```
BOOST_AUTO_TEST_CASE (
            one_position_many_names  )
```

A complex route where there is one position visited multiple times, but with different names.

7.15.1.5 BOOST_AUTO_TEST_CASE() [5/9]

```
BOOST_AUTO_TEST_CASE (
            one_position_many_visits  )
```

A complex route where one position is visited many times.

7.15.1.6 BOOST_AUTO_TEST_CASE() [6/9]

```
BOOST_AUTO_TEST_CASE (
            position_not_visited  )
```

A simple route with one point and checking the number of times for a position that wasn't visited.

7.15.1.7 BOOST_AUTO_TEST_CASE() [7/9]

```
BOOST_AUTO_TEST_CASE (
            singleton_route  )
```

A simple route with one point and one name to check.

7.15.1.8 BOOST_AUTO_TEST_CASE() [8/9]

```
BOOST_AUTO_TEST_CASE (
            singleton_route_with_spaces  )
```

A simple route with one point and one name to check, with leading and trailing spaces to check the constructor.

7.15.1.9 BOOST_AUTO_TEST_CASE() [9/9]

```
BOOST_AUTO_TEST_CASE (
            two_consecutive_visits  )
```

A simple route with two positions that are the same location.

7.15.2 Variable Documentation

7.15.2.1 isFileName

```
const bool isFileName = false
```

all data for this test suite is passed in as strings, not files.

7.16 track.cpp File Reference

```
#include <sstream>
#include <fstream>
#include <iostream>
#include <cassert>
#include <cmath>
#include <stdexcept>
#include "geometry.h"
#include "xml/element.h"
#include "xml/parser.h"
#include "track.h"
```

Include dependency graph for track.cpp:

7.17 track.h File Reference

```
#include <string>
#include <vector>
#include "types.h"
#include "position.h"
#include "route.h"
#include "xml/parser.h"
```

Include dependency graph for track.h: This graph shows which files directly or indirectly include this file:

Classes

- class GPS::Track

Namespaces

- GPS

7.18 types.h File Reference

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

Namespaces

- GPS

Typedefs

- using GPS::degrees = double
- using GPS::radians = double
- using GPS::metres = double
- using GPS::seconds = unsigned long long int
- using GPS::speed = double

Index