

## Deliverable 3

1.0

Generated by Doxygen 1.8.17



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Namespace Documentation</b>	<b>9</b>
5.1 boost Namespace Reference	9
5.2 boost::test_tools Namespace Reference	9
5.3 boost::test_tools::tt_detail Namespace Reference	9
5.4 GPS Namespace Reference	9
5.4.1 Typedef Documentation	10
5.4.1.1 degrees	10
5.4.1.2 metres	10
5.4.1.3 radians	10
5.4.1.4 seconds	11
5.4.1.5 speed	11
5.4.2 Function Documentation	11
5.4.2.1 ddmTodd()	11
5.4.2.2 degToRad()	11
5.4.2.3 normaliseDeg()	11
5.4.2.4 radToDeg()	11
5.4.2.5 sinSqr()	11
5.4.3 Variable Documentation	12
5.4.3.1 antiMeridianLongitude	12
5.4.3.2 fullRotation	12
5.4.3.3 halfRotation	12
5.4.3.4 pi	12
5.4.3.5 poleLatitude	12
5.5 GPS::Earth Namespace Reference	12
5.5.1 Function Documentation	13
5.5.1.1 latitudeSubtendedBy()	13
5.5.1.2 longitudeSubtendedBy()	13
5.5.2 Variable Documentation	13
5.5.2.1 CityCampus	13
5.5.2.2 CliftonCampus	13
5.5.2.3 EquatorialAntiMeridian	14
5.5.2.4 equatorialCircumference	14

5.5.2.5 EquatorialMeridian . . . . .	14
5.5.2.6 meanRadius . . . . .	14
5.5.2.7 NorthPole . . . . .	14
5.5.2.8 polarCircumference . . . . .	14
5.5.2.9 Pontianak . . . . .	14
5.6 GPS::LogFiles Namespace Reference . . . . .	14
5.6.1 Variable Documentation . . . . .	15
5.6.1.1 GPXRoutesDir . . . . .	15
5.6.1.2 GPXTracksDir . . . . .	15
5.6.1.3 logsDir . . . . .	15
5.6.1.4 NMEALogsDir . . . . .	15
5.7 NMEA Namespace Reference . . . . .	15
5.7.1 Typedef Documentation . . . . .	15
5.7.1.1 Route . . . . .	16
5.7.1.2 SentenceData . . . . .	16
5.7.2 Function Documentation . . . . .	16
5.7.2.1 extractSentenceData() . . . . .	16
5.7.2.2 hasValidChecksum() . . . . .	16
5.7.2.3 isWellFormedSentence() . . . . .	16
5.7.2.4 positionFromSentenceData() . . . . .	16
5.7.2.5 routeFromLog() . . . . .	16
<b>6 Class Documentation</b> . . . . .	<b>17</b>
6.1 GPS::Position Class Reference . . . . .	17
6.1.1 Constructor & Destructor Documentation . . . . .	17
6.1.1.1 Position() [1/3] . . . . .	17
6.1.1.2 Position() [2/3] . . . . .	18
6.1.1.3 Position() [3/3] . . . . .	18
6.1.2 Member Function Documentation . . . . .	18
6.1.2.1 distanceBetween() . . . . .	18
6.1.2.2 elevation() . . . . .	18
6.1.2.3 latitude() . . . . .	18
6.1.2.4 longitude() . . . . .	18
6.1.2.5 toString() . . . . .	19
6.2 boost::test_tools::tt_detail::print_log_value< std::vector< std::string > > Struct Reference . . . . .	19
6.2.1 Member Function Documentation . . . . .	19
6.2.1.1 operator()() . . . . .	19
6.3 GPS::Route Class Reference . . . . .	19
6.3.1 Constructor & Destructor Documentation . . . . .	20
6.3.1.1 Route() [1/2] . . . . .	20
6.3.1.2 Route() [2/2] . . . . .	21
6.3.2 Member Function Documentation . . . . .	21

6.3.2.1 areSameLocation()	21
6.3.2.2 containsCycles()	21
6.3.2.3 findNameOf()	21
6.3.2.4 findPosition()	21
6.3.2.5 maxElevation()	21
6.3.2.6 maxGradient()	21
6.3.2.7 maxLatitude()	22
6.3.2.8 maxLongitude()	22
6.3.2.9 minElevation()	22
6.3.2.10 minGradient()	22
6.3.2.11 minLatitude()	22
6.3.2.12 minLongitude()	22
6.3.2.13 name()	22
6.3.2.14 netHeightGain()	22
6.3.2.15 netLength()	23
6.3.2.16 numPositions()	23
6.3.2.17 operator[]()	23
6.3.2.18 setGranularity()	23
6.3.2.19 steepestGradient()	23
6.3.2.20 timesVisited() [1/2]	23
6.3.2.21 timesVisited() [2/2]	23
6.3.2.22 totalHeightGain()	24
6.3.2.23 totalLength()	24
6.3.3 Member Data Documentation	24
6.3.3.1 granularity	24
6.3.3.2 positionNames	24
6.3.3.3 positions	24
6.3.3.4 routeLength	24
6.3.3.5 routeName	24
6.4 GPS::Track Class Reference	25
6.4.1 Constructor & Destructor Documentation	25
6.4.1.1 Track()	25
6.4.2 Member Function Documentation	25
6.4.2.1 averageSpeed()	26
6.4.2.2 longestRest()	26
6.4.2.3 maxRateOfAscent()	26
6.4.2.4 maxRateOfDescent()	26
6.4.2.5 maxSpeed()	26
6.4.2.6 restingTime()	26
6.4.2.7 setGranularity()	26
6.4.2.8 totalTime()	27
6.4.2.9 travellingTime()	27

6.4.3 Member Data Documentation	27
6.4.3.1 arrived	27
6.4.3.2 departed	27
<b>7 File Documentation</b>	<b>29</b>
7.1 earth.cpp File Reference	29
7.2 earth.h File Reference	29
7.3 geometry.cpp File Reference	30
7.4 geometry.h File Reference	30
7.5 gpx-tests.cpp File Reference	30
7.5.1 Macro Definition Documentation	31
7.5.1.1 BOOST_TEST_DYN_LINK	31
7.5.1.2 BOOST_TEST_MODULE	31
7.6 logs.cpp File Reference	31
7.7 logs.h File Reference	31
7.8 nmea-tests.cpp File Reference	32
7.8.1 Macro Definition Documentation	34
7.8.1.1 BOOST_TEST_DYN_LINK	34
7.8.1.2 BOOST_TEST_MODULE	34
7.8.2 Function Documentation	34
7.8.2.1 BOOST_AUTO_TEST_CASE() [1/61]	34
7.8.2.2 BOOST_AUTO_TEST_CASE() [2/61]	34
7.8.2.3 BOOST_AUTO_TEST_CASE() [3/61]	35
7.8.2.4 BOOST_AUTO_TEST_CASE() [4/61]	35
7.8.2.5 BOOST_AUTO_TEST_CASE() [5/61]	35
7.8.2.6 BOOST_AUTO_TEST_CASE() [6/61]	35
7.8.2.7 BOOST_AUTO_TEST_CASE() [7/61]	35
7.8.2.8 BOOST_AUTO_TEST_CASE() [8/61]	35
7.8.2.9 BOOST_AUTO_TEST_CASE() [9/61]	35
7.8.2.10 BOOST_AUTO_TEST_CASE() [10/61]	36
7.8.2.11 BOOST_AUTO_TEST_CASE() [11/61]	36
7.8.2.12 BOOST_AUTO_TEST_CASE() [12/61]	36
7.8.2.13 BOOST_AUTO_TEST_CASE() [13/61]	36
7.8.2.14 BOOST_AUTO_TEST_CASE() [14/61]	36
7.8.2.15 BOOST_AUTO_TEST_CASE() [15/61]	36
7.8.2.16 BOOST_AUTO_TEST_CASE() [16/61]	36
7.8.2.17 BOOST_AUTO_TEST_CASE() [17/61]	37
7.8.2.18 BOOST_AUTO_TEST_CASE() [18/61]	37
7.8.2.19 BOOST_AUTO_TEST_CASE() [19/61]	37
7.8.2.20 BOOST_AUTO_TEST_CASE() [20/61]	37
7.8.2.21 BOOST_AUTO_TEST_CASE() [21/61]	37
7.8.2.22 BOOST_AUTO_TEST_CASE() [22/61]	37

---

7.8.2.23 BOOST_AUTO_TEST_CASE() [23/61]	37
7.8.2.24 BOOST_AUTO_TEST_CASE() [24/61]	38
7.8.2.25 BOOST_AUTO_TEST_CASE() [25/61]	38
7.8.2.26 BOOST_AUTO_TEST_CASE() [26/61]	38
7.8.2.27 BOOST_AUTO_TEST_CASE() [27/61]	38
7.8.2.28 BOOST_AUTO_TEST_CASE() [28/61]	38
7.8.2.29 BOOST_AUTO_TEST_CASE() [29/61]	38
7.8.2.30 BOOST_AUTO_TEST_CASE() [30/61]	38
7.8.2.31 BOOST_AUTO_TEST_CASE() [31/61]	39
7.8.2.32 BOOST_AUTO_TEST_CASE() [32/61]	39
7.8.2.33 BOOST_AUTO_TEST_CASE() [33/61]	39
7.8.2.34 BOOST_AUTO_TEST_CASE() [34/61]	39
7.8.2.35 BOOST_AUTO_TEST_CASE() [35/61]	39
7.8.2.36 BOOST_AUTO_TEST_CASE() [36/61]	39
7.8.2.37 BOOST_AUTO_TEST_CASE() [37/61]	39
7.8.2.38 BOOST_AUTO_TEST_CASE() [38/61]	40
7.8.2.39 BOOST_AUTO_TEST_CASE() [39/61]	40
7.8.2.40 BOOST_AUTO_TEST_CASE() [40/61]	40
7.8.2.41 BOOST_AUTO_TEST_CASE() [41/61]	40
7.8.2.42 BOOST_AUTO_TEST_CASE() [42/61]	40
7.8.2.43 BOOST_AUTO_TEST_CASE() [43/61]	40
7.8.2.44 BOOST_AUTO_TEST_CASE() [44/61]	40
7.8.2.45 BOOST_AUTO_TEST_CASE() [45/61]	41
7.8.2.46 BOOST_AUTO_TEST_CASE() [46/61]	41
7.8.2.47 BOOST_AUTO_TEST_CASE() [47/61]	41
7.8.2.48 BOOST_AUTO_TEST_CASE() [48/61]	41
7.8.2.49 BOOST_AUTO_TEST_CASE() [49/61]	41
7.8.2.50 BOOST_AUTO_TEST_CASE() [50/61]	41
7.8.2.51 BOOST_AUTO_TEST_CASE() [51/61]	41
7.8.2.52 BOOST_AUTO_TEST_CASE() [52/61]	42
7.8.2.53 BOOST_AUTO_TEST_CASE() [53/61]	42
7.8.2.54 BOOST_AUTO_TEST_CASE() [54/61]	42
7.8.2.55 BOOST_AUTO_TEST_CASE() [55/61]	42
7.8.2.56 BOOST_AUTO_TEST_CASE() [56/61]	42
7.8.2.57 BOOST_AUTO_TEST_CASE() [57/61]	42
7.8.2.58 BOOST_AUTO_TEST_CASE() [58/61]	42
7.8.2.59 BOOST_AUTO_TEST_CASE() [59/61]	43
7.8.2.60 BOOST_AUTO_TEST_CASE() [60/61]	43
7.8.2.61 BOOST_AUTO_TEST_CASE() [61/61]	43
7.8.3 Variable Documentation	43
7.8.3.1 epsilon	43
7.8.3.2 gllPos	43

---

7.8.3.3 percentageAccuracy	43
7.8.3.4 rmcPos	43
7.8.3.5 validGLLSentence	44
7.8.3.6 validMSSSentence	44
7.8.3.7 validRMCSentence	44
7.9 parseNMEA.cpp File Reference	44
7.10 parseNMEA.h File Reference	44
7.11 position.cpp File Reference	45
7.12 position.h File Reference	45
7.13 route.cpp File Reference	46
7.14 route.h File Reference	46
7.15 timesVisited(string).cpp File Reference	46
7.15.1 Function Documentation	47
7.15.1.1 BOOST_AUTO_TEST_CASE() [1/9]	47
7.15.1.2 BOOST_AUTO_TEST_CASE() [2/9]	47
7.15.1.3 BOOST_AUTO_TEST_CASE() [3/9]	48
7.15.1.4 BOOST_AUTO_TEST_CASE() [4/9]	48
7.15.1.5 BOOST_AUTO_TEST_CASE() [5/9]	48
7.15.1.6 BOOST_AUTO_TEST_CASE() [6/9]	48
7.15.1.7 BOOST_AUTO_TEST_CASE() [7/9]	48
7.15.1.8 BOOST_AUTO_TEST_CASE() [8/9]	48
7.15.1.9 BOOST_AUTO_TEST_CASE() [9/9]	49
7.15.2 Variable Documentation	49
7.15.2.1 isFileName	49
7.16 track.cpp File Reference	49
7.17 track.h File Reference	49
7.18 types.h File Reference	50
<b>Index</b>	<b>51</b>



# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">boost</a>	9
<a href="#">boost::test_tools</a>	9
<a href="#">boost::test_tools::tt_detail</a>	9
<a href="#">GPS</a>	9
<a href="#">GPS::Earth</a>	12
<a href="#">GPS::LogFiles</a>	14
<a href="#">NMEA</a>	15



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

GPS::Position . . . . .	17
boost::test_tools::tt_detail::print_log_value< std::vector< std::string > > . . . . .	19
GPS::Route . . . . .	19
GPS::Track . . . . .	25



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">GPS::Position</a> . . . . .	17
<a href="#">boost::test_tools::tt_detail::print_log_value&lt; std::vector&lt; std::string &gt; &gt;</a> . . . . .	19
<a href="#">GPS::Route</a> . . . . .	19
<a href="#">GPS::Track</a> . . . . .	25



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

<a href="#">earth.cpp</a>	29
<a href="#">earth.h</a>	29
<a href="#">geometry.cpp</a>	30
<a href="#">geometry.h</a>	30
<a href="#">gpx-tests.cpp</a>	30
<a href="#">logs.cpp</a>	31
<a href="#">logs.h</a>	31
<a href="#">nmea-tests.cpp</a>	32
<a href="#">parseNMEA.cpp</a>	44
<a href="#">parseNMEA.h</a>	44
<a href="#">position.cpp</a>	45
<a href="#">position.h</a>	45
<a href="#">route.cpp</a>	46
<a href="#">route.h</a>	46
<a href="#">timesVisited(string).cpp</a>	46
<a href="#">track.cpp</a>	49
<a href="#">track.h</a>	49
<a href="#">types.h</a>	50





## 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` `glPos` = `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,0.000,0.0,0.0,150914,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

- antiMeridianLongitude
  - GPS, [12](#)
- areSameLocation
  - GPS::Route, [21](#)
- arrived
  - GPS::Track, [27](#)
- averageSpeed
  - GPS::Track, [25](#)
- boost, [9](#)
- boost::test\_tools, [9](#)
- boost::test\_tools::tt\_detail, [9](#)
- boost::test\_tools::tt\_detail::print\_log\_value< std::vector< std::string > >, [19](#)
- operator(), [19](#)
- BOOST\_AUTO\_TEST\_CASE
  - nmea-tests.cpp, [34–43](#)
  - timesVisited(string).cpp, [47, 48](#)
- BOOST\_TEST\_DYN\_LINK
  - gpx-tests.cpp, [31](#)
  - nmea-tests.cpp, [34](#)
- BOOST\_TEST\_MODULE
  - gpx-tests.cpp, [31](#)
  - nmea-tests.cpp, [34](#)
- CityCampus
  - GPS::Earth, [13](#)
- CliftonCampus
  - GPS::Earth, [13](#)
- containsCycles
  - GPS::Route, [21](#)
- ddmTodd
  - GPS, [11](#)
- degrees
  - GPS, [10](#)
- degToRad
  - GPS, [11](#)
- departed
  - GPS::Track, [27](#)
- distanceBetween
  - GPS::Position, [18](#)
- earth.cpp, [29](#)
- earth.h, [29](#)
- elevation
  - GPS::Position, [18](#)
- epsilon
  - nmea-tests.cpp, [43](#)
- EquatorialAntiMeridian
  - GPS::Earth, [13](#)
- equatorialCircumference
  - GPS::Earth, [14](#)
- EquatorialMeridian
  - GPS::Earth, [14](#)
- extractSentenceData
  - NMEA, [16](#)
- findNameOf
  - GPS::Route, [21](#)
- findPosition
  - GPS::Route, [21](#)
- fullRotation
  - GPS, [12](#)
- geometry.cpp, [30](#)
- geometry.h, [30](#)
- gllPos
  - nmea-tests.cpp, [43](#)
- GPS, [9](#)
  - antiMeridianLongitude, [12](#)
  - ddmTodd, [11](#)
  - degrees, [10](#)
  - degToRad, [11](#)
  - fullRotation, [12](#)
  - halfRotation, [12](#)
  - metres, [10](#)
  - normaliseDeg, [11](#)
  - pi, [12](#)
  - poleLatitude, [12](#)
  - radians, [10](#)
  - radToDeg, [11](#)
  - seconds, [10](#)
  - sinSqr, [11](#)
  - speed, [11](#)
- GPS::Earth, [12](#)
  - CityCampus, [13](#)
  - CliftonCampus, [13](#)
  - EquatorialAntiMeridian, [13](#)
  - equatorialCircumference, [14](#)
  - EquatorialMeridian, [14](#)
  - latitudeSubtendedBy, [13](#)
  - longitudeSubtendedBy, [13](#)
  - meanRadius, [14](#)
  - NorthPole, [14](#)
  - polarCircumference, [14](#)
  - Pontianak, [14](#)
- GPS::LogFiles, [14](#)
  - GPXRoutesDir, [15](#)
  - GPXTracksDir, [15](#)

- logsDir, [15](#)
- NMEALogsDir, [15](#)
- GPS::Position, [17](#)
  - distanceBetween, [18](#)
  - elevation, [18](#)
  - latitude, [18](#)
  - longitude, [18](#)
  - Position, [17](#), [18](#)
  - toString, [18](#)
- GPS::Route, [19](#)
  - areSameLocation, [21](#)
  - containsCycles, [21](#)
  - findNameOf, [21](#)
  - findPosition, [21](#)
  - granularity, [24](#)
  - maxElevation, [21](#)
  - maxGradient, [21](#)
  - maxLatitude, [21](#)
  - maxLongitude, [22](#)
  - minElevation, [22](#)
  - minGradient, [22](#)
  - minLatitude, [22](#)
  - minLongitude, [22](#)
  - name, [22](#)
  - netHeightGain, [22](#)
  - netLength, [22](#)
  - numPositions, [23](#)
  - operator[], [23](#)
  - positionNames, [24](#)
  - positions, [24](#)
  - Route, [20](#)
  - routeLength, [24](#)
  - routeName, [24](#)
  - setGranularity, [23](#)
  - steepestGradient, [23](#)
  - timesVisited, [23](#)
  - totalHeightGain, [23](#)
  - totalLength, [24](#)
- GPS::Track, [25](#)
  - arrived, [27](#)
  - averageSpeed, [25](#)
  - departed, [27](#)
  - longestRest, [26](#)
  - maxRateOfAscent, [26](#)
  - maxRateOfDescent, [26](#)
  - maxSpeed, [26](#)
  - restingTime, [26](#)
  - setGranularity, [26](#)
  - totalTime, [26](#)
  - Track, [25](#)
  - travellingTime, [27](#)
- gpx-tests.cpp, [30](#)
  - BOOST\_TEST\_DYN\_LINK, [31](#)
  - BOOST\_TEST\_MODULE, [31](#)
- GPXRoutesDir
  - GPS::LogFiles, [15](#)
- GPXTracksDir
  - GPS::LogFiles, [15](#)
- granularity
  - GPS::Route, [24](#)
- halfRotation
  - GPS, [12](#)
- hasValidChecksum
  - NMEA, [16](#)
- isFileName
  - timesVisited(string).cpp, [49](#)
- isWellFormedSentence
  - NMEA, [16](#)
- latitude
  - GPS::Position, [18](#)
- latitudeSubtendedBy
  - GPS::Earth, [13](#)
- logs.cpp, [31](#)
- logs.h, [31](#)
- logsDir
  - GPS::LogFiles, [15](#)
- longestRest
  - GPS::Track, [26](#)
- longitude
  - GPS::Position, [18](#)
- longitudeSubtendedBy
  - GPS::Earth, [13](#)
- maxElevation
  - GPS::Route, [21](#)
- maxGradient
  - GPS::Route, [21](#)
- maxLatitude
  - GPS::Route, [21](#)
- maxLongitude
  - GPS::Route, [22](#)
- maxRateOfAscent
  - GPS::Track, [26](#)
- maxRateOfDescent
  - GPS::Track, [26](#)
- maxSpeed
  - GPS::Track, [26](#)
- meanRadius
  - GPS::Earth, [14](#)
- metres
  - GPS, [10](#)
- minElevation
  - GPS::Route, [22](#)
- minGradient
  - GPS::Route, [22](#)
- minLatitude
  - GPS::Route, [22](#)
- minLongitude
  - GPS::Route, [22](#)
- name
  - GPS::Route, [22](#)
- netHeightGain
  - GPS::Route, [22](#)

- netLength
  - GPS::Route, 22
- NMEA, 15
  - extractSentenceData, 16
  - hasValidChecksum, 16
  - isWellFormedSentence, 16
  - positionFromSentenceData, 16
  - Route, 15
  - routeFromLog, 16
  - SentenceData, 16
- nmea-tests.cpp, 32
  - BOOST\_AUTO\_TEST\_CASE, 34–43
  - BOOST\_TEST\_DYN\_LINK, 34
  - BOOST\_TEST\_MODULE, 34
  - epsilon, 43
  - gllPos, 43
  - percentageAccuracy, 43
  - rmcPos, 43
  - validGLLSentence, 43
  - validMSSSentence, 44
  - validRMCSentence, 44
- NMEALogsDir
  - GPS::LogFiles, 15
- normaliseDeg
  - GPS, 11
- NorthPole
  - GPS::Earth, 14
- numPositions
  - GPS::Route, 23
- operator()
  - boost::test\_tools::tt\_detail::print\_log\_value<
    - std::vector< std::string > >, 19
- operator[]
  - GPS::Route, 23
- parseNMEA.cpp, 44
- parseNMEA.h, 44
- percentageAccuracy
  - nmea-tests.cpp, 43
- pi
  - GPS, 12
- polarCircumference
  - GPS::Earth, 14
- poleLatitude
  - GPS, 12
- Pontianak
  - GPS::Earth, 14
- Position
  - GPS::Position, 17, 18
- position.cpp, 45
- position.h, 45
- positionFromSentenceData
  - NMEA, 16
- positionNames
  - GPS::Route, 24
- positions
  - GPS::Route, 24
- radians
  - GPS, 10
- radToDeg
  - GPS, 11
- restingTime
  - GPS::Track, 26
- rmcPos
  - nmea-tests.cpp, 43
- Route
  - GPS::Route, 20
  - NMEA, 15
- route.cpp, 46
- route.h, 46
- routeFromLog
  - NMEA, 16
- routeLength
  - GPS::Route, 24
- routeName
  - GPS::Route, 24
- seconds
  - GPS, 10
- SentenceData
  - NMEA, 16
- setGranularity
  - GPS::Route, 23
  - GPS::Track, 26
- sinSqr
  - GPS, 11
- speed
  - GPS, 11
- steepestGradient
  - GPS::Route, 23
- timesVisited
  - GPS::Route, 23
- timesVisited(string).cpp, 46
  - BOOST\_AUTO\_TEST\_CASE, 47, 48
  - isFileName, 49
- toString
  - GPS::Position, 18
- totalHeightGain
  - GPS::Route, 23
- totalLength
  - GPS::Route, 24
- totalTime
  - GPS::Track, 26
- Track
  - GPS::Track, 25
- track.cpp, 49
- track.h, 49
- travellingTime
  - GPS::Track, 27
- types.h, 50
- validGLLSentence
  - nmea-tests.cpp, 43
- validMSSSentence
  - nmea-tests.cpp, 44

validRMCSentence  
  nmea-tests.cpp, [44](#)