Геодезический калькулятор / Geodetic calculator

Программная документация

Раздел 1

Geodetic calculator/Геодезический калькулятор

1.1 1. Brief / Обзор

- Sovling direct and inverse geodetic tasks on ellipsoid / Решение и обратной геодезической задачи на эллипсоиде
- Conversion GEO coordinates to RAD and virsa-versa / Перевод GEO координат в RAD и обратно
- Conversion GEO coordinates to ECEF and virsa-versa / Перевод GEO координат в ECEF и обратно
- Conversion GEO coordinates to ENU and virsa-versa / Перевод GEO координат в ENU и обратно
- Conversion GEO coordinates to AER and virsa-versa / Перевод GEO координат в AER и обратно
- Conversion ECEF coordinates to ENU and virsa-versa / Перевод ECEF координат в ENU и обратно
- Conversion ECEF coordinates to AER and virsa-versa / Перевод ECEF координат в AER и обратно
- Conversion ENU coordinates to AER and virsa-versa / Перевод ECEF координат в AER и обратно $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$

$$ d = \sqrt{ (x_1-x_2)^2 + (y_1-y_2)^2 + (z_1-z_2)^2 } $$

$\left \begin{array}{rll} \nabla \cdot \mathbf{E} &=& \displaystyle \frac {\rho} {\varepsilon_0} \ \nabla \cdot \mathbf{B} &=& 0 \ \nabla \times \mathbf{E} &=& \displaystyle - \frac{\partial \mathbf{B}} {\partial t} \ \nabla \times \mathbf{B} &=& \displaystyle \mu_0\mathbf{J} + \mu_0 \varepsilon_0 \frac{\partial \mathbf{E}} {\partial t} \ \end{array} \right $

1.2 2. References / Ссылки

Papers / Статьи:

Sites / Сайты:

Repositories / Репозитории:

1.3 3. Dependencies / Зависимости

Boost for console commands parsing, testing / Boost для ввода команд с консоли, тестирования.

Раздел 2

Алфавитный указатель групп

2.1 Группы

Полный список групп.

Раздел 3

Алфавитный указатель пространств имен

3.1 Пространства имен

Полный список пространств имен.

Раздел 4

Иерархический список классов

4.1 Иерархия классов

Иерархия классов.

Раздел 5

Алфавитный указатель классов

5.1 Классы

Классы с их кратким описанием.

Раздел 6

Список файлов

6.1 Файлы

Полный список файлов.

Раздел 7

Группы

7.1 СБ ПМ (Специальная Библиотека Программных Модулей)

Пространства имен

- namespace SPML
  Специальная библиотека программных модулей (СБ ПМ)

7.1.1 Подробное описание

Специальные программные модули

7.2 Геодезический калькулятор

Классы

- struct CCoordCalcSettings
  Настройки программы

Функции

- static std::string GetVersion ()
  Возвращает строку, содержащую информацию о версии
- template<typename T >
  std::string to_string_with_precision (const T a_value, const int n=6)
  Печать в строку с задаваемым числом знаков после запятой
- int main (int argc, char *argv[])
  main - Основная функция

7.2.1 Подробное описание

Решение прямой и обратной геодезических задач на эллипсоиде, переводы координат

7.2.2 Функции

7.2.2.1 GetVersion()

static std::string GetVersion ( ) [static]

Возвращает строку, содержащую информацию о версии

Возвращает

Строка версии в формате DD-MM-YY-VV_COMMENTS, где DD - день, MM - месяц, YY - год, VV - версия, COMMENTS - комментарий(опционально)

См. определение в файле main_geocalc.cpp строка 27

7.2.2.2 main()

int main (
int argc,
char * argv[])

main - Основная функция

Аргументы

argc	- количество аргументов командной строки
argv	- аргументы командной строки

Возвращает

0 - штатная работа, 1 - ошибка

См. определение в файле main_geocalc.cpp строка 85

7.2.2.3 to_string_with_precision()

template<typename T >
std::string to_string_with_precision (
const T a_value,
const int n = 6)

Печать в строку с задаваемым числом знаков после запятой

См. определение в файле main_geocalc.cpp строка 37

Раздел 8

Пространства имен

8.1 Пространство имен SPML

Специальная библиотека программных модулей (СБ ПМ)

Пространства имен

- namespace Compare
  Сравнение чисел
- namespace Consts
  Константы
- namespace Convert
  Переводы единиц
- namespace Geodesy
  Геодезические функции и функции перевода координат
- namespace Units
  Единицы измерения физических величин, форматы чисел

Функции

- std::string GetVersion ()
  Возвращает строку, содержащую информацию о версии
- void ClearConsole ()
  Очистка консоли (терминала) в *nix.

8.1.1 Подробное описание

Специальная библиотека программных модулей (СБ ПМ)

8.1.2 Функции

8.1.2.1 ClearConsole()

void SPML::ClearConsole ( )

Очистка консоли (терминала) в *nix.

См. определение в файле spml.cpp строка 22

8.1.2.2 GetVersion()

std::string SPML::GetVersion ( )

Возвращает строку, содержащую информацию о версии

Возвращает

Строка версии в формате DD-MM-YY-VV_COMMENTS, где DD - день, MM - месяц, YY - год, VV - версия, COMMENTS - комментарий(опционально)

См. определение в файле spml.cpp строка 16

8.2 Пространство имен SPML::Compare

Сравнение чисел

Функции

- bool AreEqualAbs (float first, float second, const float &eps=EPS_F)
  Сравнение двух действительных чисел (по абсолютной разнице)
- bool AreEqualAbs (double first, double second, const double &eps=EPS_D)
  Сравнение двух действительных чисел (по абсолютной разнице)
- bool AreEqualRel (float first, float second, const float &eps=EPS_REL)
  Сравнение двух действительных чисел (по относительной разнице)
- bool AreEqualRel (double first, double second, const double &eps=EPS_REL)
  Сравнение двух действительных чисел (по относительной разнице)
- bool IsZeroAbs (float value, const float &eps=EPS_F)
  Проверка действительного числа на равенство нулю (по абсолютной разнице)
- bool IsZeroAbs (double value, const double &eps=EPS_D)
  Проверка действительного числа на равенство нулю (по абсолютной разнице)

Переменные

- static const float EPS_F = 1.0e-4f
  Абсолютная точность по умолчанию при сравнениях чисел типа float (1.0e-4)
- static const double EPS_D = 1.0e-8
  Абсолютная точность по умолчанию при сравнениях чисел типа double (1.0e-8)
- static const float EPS_REL = 0.01
  Относительная точность по умолчанию

8.2.1 Подробное описание

Сравнение чисел

8.2.2 Функции

8.2.2.1 AreEqualAbs() [1/2]

```
bool SPML::Compare::AreEqualAbs (
                double first,
                double second,
                const double & eps = EPS_D )   [inline]
```

Сравнение двух действительных чисел (по абсолютной разнице)

Возвращает результат: abs(first - second) < eps

Аргументы

in	first	- первое число
in	second	- второе число
in	eps	- абсолютная точность сравнения

Возвращает

true - если разница меньше точности, иначе false

См. определение в файле compare.h строка 51

8.2.2.2 AreEqualAbs() [2/2]

```
bool SPML::Compare::AreEqualAbs (
                float first,
                float second,
                const float & eps = EPS_F )   [inline]
```

Сравнение двух действительных чисел (по абсолютной разнице)

Возвращает результат: abs(first - second) < eps

Аргументы

in	first	- первое число
in	second	- второе число
in	eps	- абсолютная точность сравнения

Возвращает

true - если разница меньше точности, иначе false

См. определение в файле compare.h строка 38

8.2.2.3 AreEqualRel() [1/2]

bool SPML::Compare::AreEqualRel (
double first,
double second,
const double & eps = EPS_REL) [inline]

Сравнение двух действительных чисел (по относительной разнице)

Возарвщает результат: (abs((first - second) / first) < eps) && (abs((first - second) / second) < eps)

Аргументы

in	first	- первое число
in	second	- второе число
in	eps	- относительная точность сравнения

Возвращает

true - если разница меньше точности, иначе false

См. определение в файле compare.h строка 79

8.2.2.4 AreEqualRel() [2/2]

bool SPML::Compare::AreEqualRel (
float first,
float second,
const float & eps = EPS_REL) [inline]

Сравнение двух действительных чисел (по относительной разнице)

Возарвщает результат: (abs((first - second) / first) < eps) && (abs((first - second) / second) < eps)

Аргументы

in	first	- первое число
in	second	- второе число
in	eps	- относительная точность сравнения

Возвращает

true - если разница меньше точности, иначе false

См. определение в файле compare.h строка 65

8.2.2.5 IsZeroAbs() [1/2]

bool SPML::Compare::IsZeroAbs (
double value,
const double & eps = EPS_D) [inline]

Проверка действительного числа на равенство нулю (по абсолютной разнице)

Возвращает результат: abs(value) < eps

Аргументы

in	value	- проверяемое число
in	eps	- абсолютная точность сравнения

Возвращает

true - если разница меньше точности, иначе false

См. определение в файле compare.h строка 105

8.2.2.6 IsZeroAbs() [2/2]

bool SPML::Compare::IsZeroAbs (
float value,
const float & eps = EPS_F) [inline]

Проверка действительного числа на равенство нулю (по абсолютной разнице)

Возвращает результат: abs(value) < eps

Аргументы

in	value	- проверяемое число
in	eps	- абсолютная точность сравнения

Возвращает

true - если разница меньше точности, иначе false

См. определение в файле compare.h строка 93

8.2.3 Переменные

8.2.3.1 EPS_D

const double SPML::Compare::EPS_D = 1.0e-8 [static]

Абсолютная точность по умолчанию при сравнениях чисел типа double (1.0e-8)

См. определение в файле compare.h строка 26

8.2.3.2 EPS_F

const float SPML::Compare::EPS_F = 1.0e-4f [static]

Абсолютная точность по умолчанию при сравнениях чисел типа float (1.0e-4)

См. определение в файле compare.h строка 25

8.2.3.3 EPS_REL

const float SPML::Compare::EPS_REL = 0.01 [static]

Относительная точность по умолчанию

См. определение в файле compare.h строка 27

8.3 Пространство имен SPML::Consts

Константы

Переменные

- const float C_F = 3.0e8f

  Скорость света, [м/с] в одинарной точности (float)
- const double C_D = 3.0e8

  Скорость света, [м/с] в двойной точности (double)
- const double PI_D = std::acos(-1.0)

  Число PI = 3.14... в радианах в двойной точности (double)
- const float PI_F = static_cast<float>(std::acos(-1.0))

  Число PI = 3.14... в радианах в одинарной точности (float)
- const double PI_2_D = 2.0 * std::acos(-1.0)

  Число 2*PI = 6.28... в радианах в двойной точности (double)
- const float PI_2_F = static_cast<float>(2.0 * std::acos(-1.0))

  Число 2*PI = 6.28... в радианах в одинарной точности (float)
- const double PI_05_D = std::acos(0.0)

  Число PI/2 = 1.57... в радианах в двойной точности (double)
- const float PI_05_F = static_cast<float>(std::acos(0.0))

  Число PI/2 = 1.57... в радианах в одинарной точности (float)
- const double PI_025_D = std::acos(-1.0) * 0.25

  Число PI/4 = 0.785... в радианах в двойной точности (double)
- const float PI_025_F = static_cast<float>(std::acos(-1.0) * 0.25)

  Число PI/4 = 0.785... в радианах в одинарной точности (float)

8.3.1 Подробное описание

Константы

8.3.2 Переменные

8.3.2.1 C_D

const double SPML::Consts::C_D = 3.0e8

Скорость света, [м/с] в двойной точности (double)

См. определение в файле consts.h строка 24

8.3.2.2 C_F

const float SPML::Consts::C_F = 3.0e8f

Скорость света, [м/с] в одинарной точности (float)

См. определение в файле consts.h строка 23

8.3.2.3 PI_025_D

const double SPML::Consts::PI_025_D = std::acos(-1.0) * 0.25

Число PI/4 = 0.785... в радианах в двойной точности (double)

См. определение в файле consts.h строка 37

8.3.2.4 PI_025_F

const float SPML::Consts::PI_025_F = static_cast<float>(std::acos(-1.0) * 0.25)

Число PI/4 = 0.785... в радианах в одинарной точности (float)

См. определение в файле consts.h строка 38

8.3.2.5 PI_05_D

const double SPML::Consts::PI_05_D = std::acos(0.0)

Число PI/2 = 1.57... в радианах в двойной точности (double)

См. определение в файле consts.h строка 34

8.3.2.6 PI_05_F

const float SPML::Consts::PI_05_F = static_cast<float>(std::acos(0.0))

Число PI/2 = 1.57... в радианах в одинарной точности (float)

См. определение в файле consts.h строка 35

8.3.2.7 PI_2_D

const double SPML::Consts::PI_2_D = 2.0 * std::acos(-1.0)

Число 2*PI = 6.28... в радианах в двойной точности (double)

См. определение в файле consts.h строка 31

8.3.2.8 PI_2_F

const float SPML::Consts::PI_2_F = static_cast<float>(2.0 * std::acos(-1.0))

Число 2*PI = 6.28... в радианах в одинарной точности (float)

См. определение в файле consts.h строка 32

8.3.2.9 PI_D

const double SPML::Consts::PI_D = std::acos(-1.0)

Число PI = 3.14... в радианах в двойной точности (double)

См. определение в файле consts.h строка 28

8.3.2.10 PI_F

const float SPML::Consts::PI_F = static_cast<float>(std::acos(-1.0))

Число PI = 3.14... в радианах в одинарной точности (float)

См. определение в файле consts.h строка 29

8.4 Пространство имен SPML::Convert

Переводы единиц

Функции

- float AngleTo360 (float angle, const Units::TAngleUnit &au)
  Приведение угла в [0,360) градусов или [0,2PI) радиан
- double AngleTo360 (double angle, const Units::TAngleUnit &au)
  Приведение угла в [0,360) градусов или [0,2PI) радиан
- float EpsToMP90 (float angle, const Units::TAngleUnit &au)
  Приведение угла места в [-90,90] градусов или [-PI/2, PI/2] радиан
- double EpsToMP90 (double angle, const Units::TAngleUnit &au)
  Приведение угла места в [-90,90] градусов или [-PI/2, PI/2] радиан
- template<class T >
  T AbsAzToRelAz (T absAz, T origin, const Units::TAngleUnit &au)
  Перевод абсолютного азимута относительно севера в азимут относительно указанного направления
- template<class T >
  T RelAzToAbsAz (T relAz, T origin, const Units::TAngleUnit &au)
  Перевод относительного азимута в абсолютный азимут относительно севера

- template<class T >
  T dBtoTimesByP (T dB)

  Перевод [дБ] в разы по мощности
- template<class T >
  T dBtoTimesByU (T dB)

  Перевод [дБ] в разы по напряжению
- void UnixTimeToHourMinSec (int rawtime, int &hour, int &min, int &sec, int &day=dummy_int, int &mon=dummy_int, int &year=dummy_int)

  Перевод целого числа секунд с 00:00:00 01.01.1970 в часы/минуты/секунды/день/месяц/год
- const std::string CurrentDateTimeToString ()

  Получение текущей даты и времени
- double CheckDeltaAngle (double deltaAngle, const SPML::Units::TAngleUnit &au)

  Проверка разницы в углах

Переменные

- const float DgToRdF = static_cast<float>(std::asin(1.0) / 90.0)

  Перевод градусов в радианы (float) путем умножения на данную константу
- const float RdToDgF = static_cast<float>(90.0 / std::asin(1.0))

  Перевод радианов в градусы (float) путем умножения на данную константу
- const double DgToRdD = std::asin(1.0) / 90.0

  Перевод градусов в радианы (double) путем умножения на данную константу
- const double RdToDgD = 90.0 / std::asin(1.0)

  Перевод радианов в градусы (double) путем умножения на данную константу
- const double MsToKmD_half = Consts::C_D * 0.5 * 1.0e-6

  Перевод задержки [мс] в дальность [км] путем умножения на данную константу (по формуле R = C * Tau / 2)
- const double KmToMsD_half = 1.0 / MsToKmD_half

  Перевод дальности [км] в задержку [мс] путем умножения на данную константу (по формуле Tau = 2 * R / C)
- const double McsToKmD_half = Consts::C_D * 0.5 * 1.0e-9

  Перевод задержки [мкс] в дальность [км] путем умножения на данную константу (по формуле R = C * Tau / 2)
- const double KmToMcsD_half = 1.0 / McsToKmD_half

  Перевод дальности [км] в задержку [мкс] путем умножения на данную константу (по формуле Tau = 2 * R / C)
- const double MsToMetersD_full = Consts::C_D * 1.0e-3

  Перевод задержки [мс] в дальность [м] путем умножения на данную константу (по формуле R = C * Tau / 2)
- const double MetersToMsD_full = 1.0 / MsToMetersD_full

  Перевод дальности [м] в задержку [мс] путем умножения на данную константу (по формуле Tau = 2 * R / C)
- const double MsToKmD_full = Consts::C_D * 1.0e-6

  Перевод задержки [мс] в дальность [км] путем умножения на данную константу (по формуле R = C * Tau / 2)
- const double KmToMsD_full = 1.0 / MsToKmD_full

  Перевод дальности [км] в задержку [мс] путем умножения на данную константу (по формуле Tau = 2 * R / C)
- static int dummy_int

8.4.1 Подробное описание

Переводы единиц

8.4.2 Функции

8.4.2.1 AbsAzToRelAz()

```
template<class T >
T SPML::Convert::AbsAzToRelAz (
                T absAz,
                T origin,
                const Units::TAngleUnit & au )   [inline]
```

Перевод абсолютного азимута относительно севера в азимут относительно указанного направления

Аргументы

in	absAz	- абсолютный азимут относительно севера
in	origin	- абсолютный азимут, относительно которого измеряется относительный
in	au	- единицы измерения углов входных/выходных параметров

Возвращает

Азимут относительно указанного направления origin (положительный азимут 0..180 по часовой стрелке, отрицательный 0..-180 против часовой стрелки)

См. определение в файле convert.h строка 119

8.4.2.2 AngleTo360() [1/2]

```
double SPML::Convert::AngleTo360 (
                double angle,
                const Units::TAngleUnit & au )
```

Приведение угла в [0,360) градусов или [0,2PI) радиан

Аргументы

in	angle	- приводимый угол в [град] или [рад] в зависимости от параметра.
in	au	- выбор угловых единиц [град] или [рад]

Возвращает

Значение angle, приведенное в [0,360) градусов или [0,2PI) радиан

См. определение в файле convert.cpp строка 78

8.4.2.3 AngleTo360() [2/2]

```
float SPML::Convert::AngleTo360 (
            float angle,
            const Units::TAngleUnit & au )
```

Приведение угла в [0,360) градусов или [0,2PI) радиан

Аргументы

in	angle	- приводимый угол в [град] или [рад] в зависимости от параметра.
in	au	- выбор угловых единиц [град] или [рад]

Возвращает

Значение angle, приведенное в [0,360) градусов или [0,2PI) радиан

См. определение в файле convert.cpp строка 18

8.4.2.4 CheckDeltaAngle()

```
double SPML::Convert::CheckDeltaAngle (
            double deltaAngle,
            const SPML::Units::TAngleUnit & au )
```

Проверка разницы в углах

Например, A1=10, A2=30, тогда разница=20, но если A1=10, а A2=350, то разница тоже 20, а не A2-A1=340 !

Аргументы

deltaAngle	- Разница в углах
au	- Единицы измерения разницы углов

Возвращает

Разница в углах, приведенная в 0-180 градусов (0-PI/2 радиан)

См. определение в файле convert.cpp строка 248

8.4.2.5 CurrentDateTimeToString()

const std::string SPML::Convert::CurrentDateTimeToString ( )

Получение текущей даты и времени

Возвращает

Возвращает текущую дату в строке формата YYYY-MM-DD.HH:mm:ss

См. определение в файле convert.cpp строка 236

8.4.2.6 dBtoTimesByP()

template<class T >
T SPML::Convert::dBtoTimesByP (
T dB) [inline]

Перевод [дБ] в разы по мощности

Аргументы

in	dB	- децибелы

Возвращает

децибелы, перевденные в разы по мощности

См. определение в файле convert.h строка 149

8.4.2.7 dBtoTimesByU()

template<class T >
T SPML::Convert::dBtoTimesByU (
T dB) [inline]

Перевод [дБ] в разы по напряжению

Аргументы

in	dB	- децибелы

Возвращает

децибелы, перевденные в разы по напряжению

См. определение в файле convert.h строка 161

8.4.2.8 EpsToMP90() [1/2]

double SPML::Convert::EpsToMP90 (
double angle,
const Units::TAngleUnit & au)

Приведение угла места в [-90,90] градусов или [-PI/2, PI/2] радиан

Аргументы

in	angle	- приводимый угол в [град] или [рад] в зависимости от параметра.
in	au	- выбор угловых единиц [град] или [рад]

Возвращает

Значение angle, приведенное в [-90,90] градусов или [-PI/2, PI/2] радиан

См. определение в файле convert.cpp строка 178

8.4.2.9 EpsToMP90() [2/2]

float SPML::Convert::EpsToMP90 (
float angle,
const Units::TAngleUnit & au)

Приведение угла места в [-90,90] градусов или [-PI/2, PI/2] радиан

Аргументы

in	angle	- приводимый угол в [град] или [рад] в зависимости от параметра.
in	au	- выбор угловых единиц [град] или [рад]

Возвращает

Значение angle, приведенное в [-90,90] градусов или [-PI/2, PI/2] радиан

См. определение в файле convert.cpp строка 138

8.4.2.10 RelAzToAbsAz()

```
template<class T >
T SPML::Convert::RelAzToAbsAz (
                T relAz,
                T origin,
                const Units::TAngleUnit & au )   [inline]
```

Перевод относительного азимута в абсолютный азимут относительно севера

Единицы измерения углов входных/выходных параметров согласно TAngleUnit

Аргументы

in	relAz	- относительный азимут
in	origin	- абсолютный азимут, относительно которого измеряется relAz
in	au	- единицы измерения углов входных/выходных параметров

Возвращает

Абсолютный азимут относительно севера

См. определение в файле convert.h строка 135

8.4.2.11 UnixTimeToHourMinSec()

```
void SPML::Convert::UnixTimeToHourMinSec (
                int rawtime,
                int & hour,
                int & min,
                int & sec,
                int & day = dummy_int,
                int & mon = dummy_int,
                int & year = dummy_int )
```

Перевод целого числа секунд с 00:00:00 01.01.1970 в часы/минуты/секунды/день/месяц/год

Аргументы

in	rawtime	- число секунд с 00:00:00 01.01.1970
out	hour	- часы
out	min	- минуты
out	sec	- секунды
out	day	- день
out	mon	- месяц
out	year	- год

См. определение в файле convert.cpp строка 223

8.4.3 Переменные

8.4.3.1 DgToRdD

const double SPML::Convert::DgToRdD = std::asin(1.0) / 90.0

Перевод градусов в радианы (double) путем умножения на данную константу

См. определение в файле convert.h строка 33

8.4.3.2 DgToRdF

const float SPML::Convert::DgToRdF = static_cast<float>(std::asin(1.0) / 90.0)

Перевод градусов в радианы (float) путем умножения на данную константу

См. определение в файле convert.h строка 31

8.4.3.3 dummy_int

int SPML::Convert::dummy_int [static]

См. определение в файле convert.h строка 167

8.4.3.4 KmToMcsD_half

const double SPML::Convert::KmToMcsD_half = 1.0 / McsToKmD_half

Перевод дальности [км] в задержку [мкс] путем умножения на данную константу (по формуле Tau = 2 * R / C)

См. определение в файле convert.h строка 48

8.4.3.5 KmToMsD_full

const double SPML::Convert::KmToMsD_full = 1.0 / MsToKmD_full

Перевод дальности [км] в задержку [мс] путем умножения на данную константу (по формуле Tau = 2 * R / C)

См. определение в файле convert.h строка 62

8.4.3.6 KmToMsD_half

const double SPML::Convert::KmToMsD_half = 1.0 / MsToKmD_half

Перевод дальности [км] в задержку [мс] путем умножения на данную константу (по формуле Tau = 2 * R / C)

См. определение в файле convert.h строка 42

8.4.3.7 McsToKmD_half

const double SPML::Convert::McsToKmD_half = Consts::C_D * 0.5 * 1.0e-9

Перевод задержки [мкс] в дальность [км] путем умножения на данную константу (по формуле R = C * Tau / 2)

См. определение в файле convert.h строка 47

8.4.3.8 MetersToMsD_full

const double SPML::Convert::MetersToMsD_full = 1.0 / MsToMetersD_full

Перевод дальности [м] в задержку [мс] путем умножения на данную константу (по формуле Tau = 2 * R / C)

См. определение в файле convert.h строка 59

8.4.3.9 MsToKmD_full

const double SPML::Convert::MsToKmD_full = Consts::C_D * 1.0e-6

Перевод задержки [мс] в дальность [км] путем умножения на данную константу (по формуле R = C * Tau / 2)

См. определение в файле convert.h строка 61

8.4.3.10 MsToKmD_half

const double SPML::Convert::MsToKmD_half = Consts::C_D * 0.5 * 1.0e-6

Перевод задержки [мс] в дальность [км] путем умножения на данную константу (по формуле R = C * Tau / 2)

См. определение в файле convert.h строка 41

8.4.3.11 MsToMetersD_full

const double SPML::Convert::MsToMetersD_full = Consts::C_D * 1.0e-3

Перевод задержки [мс] в дальность [м] путем умножения на данную константу (по формуле R = C * Tau / 2)

См. определение в файле convert.h строка 58

8.4.3.12 RdToDgD

const double SPML::Convert::RdToDgD = 90.0 / std::asin(1.0)

Перевод радианов в градусы (double) путем умножения на данную константу

См. определение в файле convert.h строка 34

8.4.3.13 RdToDgF

const float SPML::Convert::RdToDgF = static_cast<float>(90.0 / std::asin(1.0))

Перевод радианов в градусы (float) путем умножения на данную константу

См. определение в файле convert.h строка 32

8.5 Пространство имен SPML::Geodesy

Геодезические функции и функции перевода координат

Пространства имен

- namespace Ellipsoids

  Земные эллипсоиды

Классы

- struct AER

  Локальные сферические координаты AER (Azimuth-Elevation-Range, Азимут-Угол места-Дальность)
- class CEllipsoid

  Земной эллипсоид
- struct ENU

  Координаты ENU (East-North-Up)
- struct Geodetic

  Геодезические координаты (широта, долгота, высота)
- struct Geographic

  Географические координаты (широта, долгота)
- struct RAD

  Радиолокационные координаты (расстояние по ортодроме, азимут, конечный азимут)
- struct UVW

  Координаты UVW.
- struct XYZ

  3D декартовы ортогональные координаты (X, Y, Z)

Функции

- void GEOtoRAD (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double latStart, double lonStart, double latEnd, double lonEnd, double &d, double &az, double &azEnd=dummy_double)

  Пересчет географических координат в радиолокационные (Обратная геодезическая задача)
- RAD GEOtoRAD (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geographic &start, const Geographic &end)

  Пересчет географических координат в радиолокационные (Обратная геодезическая задача)
- void RADtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double latStart, double lonStart, double d, double az, double &latEnd, double &lonEnd, double &azEnd=dummy_double)

  Пересчет радиолокационных координат в географические (Прямая геодезическая задача)
- Geographic RADtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geographic &start, const RAD &rad, double &azEnd=dummy_double)

  Пересчет радиолокационных координат в географические (Прямая геодезическая задача)
- void GEOtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double lat, double lon, double h, double &x, double &y, double &z)

  Пересчет широты, долготы, высоты в декартовые геоцентрические координаты
- XYZ GEOtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geodetic point)

  Пересчет широты, долготы, высоты в декартовые геоцентрические координаты
- void ECEFtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double x, double y, double z, double &lat, double &lon, double &h)

  Пересчет декартовых геоцентрических координат в широту, долготу, высоту
- Geodetic ECEFtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, XYZ &point)

  Пересчет декартовых геоцентрических координат в широту, долготу, высоту
- double XYZtoDistance (double x1, double y1, double z1, double x2, double y2, double z2)

  Вычисление расстояния между точками в декартовых координатах
- double XYZtoDistance (const XYZ &point1, const XYZ &point2)

  Вычисление расстояния между точками в декартовых координатах
- void ECEF_offset (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double lat1, double lon1, double h1, double lat2, double lon2, double h2, double &dX, double &dY, double &dZ)

  ECEF смещение (разница в декартовых ECEF координатах двух точек)
- XYZ ECEF_offset (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geodetic &point1, const Geodetic &point2)

  ECEF смещение (разница в декартовых ECEF координатах двух точек)
- void ECEFtoENU (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double x, double y, double z, double lat, double lon, double h, double &xEast, double &yNorth, double &zUp)

  Перевод ECEF координат точки в ENU относительно географических координат опорной точки (lat, lon)
- ENU ECEFtoENU (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const XYZ &ecef, const Geodetic &point)

  Перевод ECEF координат точки в ENU относительно географических координат опорной точки point.
- void ECEFtoENUV (const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double dX, double dY, double dZ, double lat, double lon, double &xEast, double &yNorth, double &zUp)

Перевод ECEF координат точки в ENU относительно географических координат (lat, lon)

- ENU ECEFtoENUV (const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const XYZ &shift, const Geographic &point)

  Перевод ECEF координат точки в ENU относительно географических координат point.
- void ENUtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double e, double n, double u, double lat, double lon, double h, double &x, double &y, double &z)

  Перевод ENU координат точки в ECEF относительно географических координат опорной точки (lat, lon)
- XYZ ENUtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const ENU &enu, const Geodetic &point)

  Перевод ENU координат точки в ECEF относительно географических координат точки point.
- void ENUtoAER (const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double xEast, double yNorth, double zUp, double &az, double &elev, double &slantRange)

  Перевод ENU координат точки в AER координаты
- AER ENUtoAER (const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const ENU &point)

  Перевод ENU координат точки в AER координаты
- void AERtoENU (const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double az, double elev, double slantRange, double &xEast, double &yNorth, double &zUp)

  Перевод AER координат точки в ENU координаты
- ENU AERtoENU (const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const AER &aer)

  Перевод ENU координат точки в AER координаты
- void GEOtoENU (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double lat, double lon, double h, double lat0, double lon0, double h0, double &xEast, double &yNorth, double &zUp)

  Перевод геодезических координат GEO точки point в координаты ENU относительно опорной точки
- ENU GEOtoENU (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geodetic &point, const Geodetic &anchor)

  Перевод геодезических координат GEO точки point в координаты ENU относительно опорной точки
- void ENUtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double xEast, double yNorth, double zUp, double lat0, double lon0, double h0, double &lat, double &lon, double &h)

  Перевод координат ENU в геодезические координаты GEO относительно опорной точки
- Geodetic ENUtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const ENU &point, const Geodetic &anchor)

  Перевод координат ENU в геодезические координаты GEO относительно опорной точки
- void GEOtoAER (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double lat1, double lon1, double h1, double lat2, double lon2, double h2, double &az, double &elev, double &slantRange)

  Вычисление AER координат между двумя геодезическими точками
- AER GEOtoAER (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geodetic &point1, const Geodetic &point2)

  Вычисление AER координат между двумя геодезическими точками
- void AERtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double az, double elev, double slantRange, double lat0, double lon0, double h0, double &lat, double &lon, double &h)

  Перевод AER координат в геодезические относительно опорной точки
- Geodetic AERtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const AER &aer, const Geodetic &anchor)

  Перевод AER координат в геодезические относительно опорной точки

- void AERtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double az, double elev, double slantRange, double lat0, double lon0, double h0, double &x, double &y, double &z)

  Перевод AER координат относительно опорной точки в глобальные декартовые
- XYZ AERtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const AER &aer, const Geodetic &anchor)

  Перевод AER координат относительно опорной точки в глобальные декартовые
- void ECEFtoAER (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double x, double y, double z, double lat0, double lon0, double h0, double &az, double &elev, double &slantRange)

  Перевод AER координат относительно опорной точки в глобальные декартовые
- AER ECEFtoAER (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const XYZ &ecef, const Geodetic &anchor)

  Перевод AER координат относительно опорной точки в глобальные декартовые
- void ENUtoUVW (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double xEast, double yNorth, double zUp, double lat0, double lon0, double &u, double &v, double &w)

  Перевод ENU координат точки в UVW координаты
- UVW ENUtoUVW (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const ENU &enu, const Geographic &point)

  Перевод ENU координат точки в UVW координаты
- double CosAngleBetweenVectors (double x1, double y1, double z1, double x2, double y2, double z2)

  Косинус угла между векторами в евклидовом пространстве
- double CosAngleBetweenVectors (const XYZ &point1, const XYZ &point2)

  Косинус угла между векторами в евклидовом пространстве
- double AngleBetweenVectors (double x1, double y1, double z1, double x2, double y2, double z2)

  Угол между векторами в евклидовом пространстве
- double AngleBetweenVectors (const XYZ &vec1, const XYZ &vec2)

  Угол между векторами в евклидовом пространстве
- void VectorFromTwoPoints (double x1, double y1, double z1, double x2, double y2, double z2, double &xV, double &yV, double &zV)

  Вектор из координат двух точек
- XYZ VectorFromTwoPoints (const XYZ &point1, const XYZ &point2)

  Вектор, полученный из координат двух точек

Переменные

- static double dummy_double

8.5.1 Подробное описание

Геодезические функции и функции перевода координат

8.5.2 Функции

8.5.2.1 AERtoECEF() [1/2]

```
XYZ SPML::Geodesy::AERtoECEF (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                const AER & aer,
                const Geodetic & anchor )
```

Перевод AER координат относительно опорной точки в глобальные декартовые

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	aer	- азимут, угол места, наклонная дальность
in	anchor	- геодезические координаты опорной точки

Возвращает

AER координаты точки в ECEF координатах относительно опорной точки

См. определение в файле geodesy.cpp строка 1556

8.5.2.2 AERtoECEF() [2/2]

```
void SPML::Geodesy::AERtoECEF (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                double az,
                double elev,
                double slantRange,
                double lat0,
                double lon0,
                double h0,
                double & x,
                double & y,
                double & z )
```

Перевод AER координат относительно опорной точки в глобальные декартовые

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	az	- азимут из опорной точки на искомую точку
in	elev	- угол места из опорной точки на искомую точку

Аргументы

in	slantRange	- наклонная дальность от опорной точки до искомой точки
in	lat0	- широта опорной точки
in	lon0	- долгота опорной точки
in	h0	- высота опорной точки
out	x	- ECEF координата X
out	y	- ECEF координата Y
out	z	- ECEF координата X

См. определение в файле geodesy.cpp строка 1494

8.5.2.3 AERtoENU() [1/2]

```
ENU SPML::Geodesy::AERtoENU (
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                const AER & aer )
```

Перевод ENU координат точки в AER координаты

Аргументы

in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	aer	- точка в координатах aer

Возвращает

Координаты ENU точки point

См. определение в файле geodesy.cpp строка 1189

8.5.2.4 AERtoENU() [2/2]

```
void SPML::Geodesy::AERtoENU (
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                double az,
                double elev,
                double slantRange,
                double & xEast,
                double & yNorth,
                double & zUp )
```

Перевод AER координат точки в ENU координаты

Аргументы

in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	az	- азимут (A)
in	elev	- угол места (E)
in	slantRange	- наклонная дальность (R)
out	xEast	- ENU координата X (East)
out	yNorth	- ENU координата Y (North)
out	zUp	- ENU координата X (Up)

См. определение в файле geodesy.cpp строка 1138

8.5.2.5 AERtoGEO() [1/2]

```
Geodetic SPML::Geodesy::AERtoGEO (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                const AER & aer,
                const Geodetic & anchor )
```

Перевод AER координат в геодезические относительно опорной точки

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	aer	- азимут, угол места, наклонная дальность от опорной точки на искомую
in	anchor	- геодезические координаты опорной точки

Возвращает

Геодезические координаты конечной точки координат AER относительно опорной точки

См. определение в файле geodesy.cpp строка 1486

8.5.2.6 AERtoGEO() [2/2]

```
void SPML::Geodesy::AERtoGEO (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
```

double az,
double elev,
double slantRange,
double lat0,
double lon0,
double h0,
double & lat,
double & lon,
double & h)

Перевод AER координат в геодезические относительно опорной точки

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	az	- азимут из опорной точки на искомую точку
in	elev	- угол места из опорной точки на искомую точку
in	slantRange	- наклонная дальность от опорной точки до искомой точки
in	lat0	- широта опорной точки
in	lon0	- долгота опорной точки
in	h0	- высота опорной точки
out	lat	- широта искомой точки
out	lon	- долгота искомой точки
out	h	- высота искомой точки

См. определение в файле geodesy.cpp строка 1419

8.5.2.7 AngleBetweenVectors() [1/2]

double SPML::Geodesy::AngleBetweenVectors (
const XYZ & vec1,
const XYZ & vec2)

Угол между векторами в евклидовом пространстве

Предполагается, что оба вектора начинаются в точке (0, 0, 0)

Аргументы

in	vec1	- вектор 1
in	vec2	- вектор 2

Возвращает

Угол между векторами, [радиан]

См. определение в файле geodesy.cpp строка 1724

8.5.2.8 AngleBetweenVectors() [2/2]

```
double SPML::Geodesy::AngleBetweenVectors (
                double x1,
                double y1,
                double z1,
                double x2,
                double y2,
                double z2 )
```

Угол между векторами в евклидовом пространстве

Предполагается, что оба вектора начинаются в точке (0, 0, 0)

Аргументы

in	x1	- X координата 1 точки
in	y1	- Y координата 1 точки
in	z1	- Z координата 1 точки
in	x2	- X координата 2 точки
in	y2	- Y координата 2 точки
in	z2	- Z координата 2 точки

Возвращает

Угол между векторами, [радиан]

См. определение в файле geodesy.cpp строка 1719

8.5.2.9 CosAngleBetweenVectors() [1/2]

```
double SPML::Geodesy::CosAngleBetweenVectors (
                const XYZ & point1,
                const XYZ & point2 )
```

Косинус угла между векторами в евклидовом пространстве

Предполагается, что оба вектора начинаются в точке (0, 0, 0)

Аргументы

in	point1	- 1 точка
in	point2	- 2 точка

Возвращает

Косинус угла между векторами, [радиан]

См. определение в файле geodesy.cpp строка 1713

8.5.2.10 CosAngleBetweenVectors() [2/2]

```
double SPML::Geodesy::CosAngleBetweenVectors (
                double x1,
                double y1,
                double z1,
                double x2,
                double y2,
                double z2 )
```

Косинус угла между векторами в евклидовом пространстве

Предполагается, что оба вектора начинаются в точке (0, 0, 0)

Аргументы

in	x1	- X координата 1 точки
in	y1	- Y координата 1 точки
in	z1	- Z координата 1 точки
in	x2	- X координата 2 точки
in	y2	- Y координата 2 точки
in	z2	- Z координата 2 точки

Возвращает

Косинус угла между векторами, [радиан]

См. определение в файле geodesy.cpp строка 1700

8.5.2.11 ECEF_offset() [1/2]

```
XYZ SPML::Geodesy::ECEF_offset (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                const Geodetic & point1,
                const Geodetic & point2 )
```

ECEF смещение (разница в декартовых ECEF координатах двух точек)

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	point1	- точка 1
in	point2	- точка 2

Возвращает

ECEF смещение

См. определение в файле geodesy.cpp строка 844

8.5.2.12 ECEF_offset() [2/2]

```
void SPML::Geodesy::ECEF_offset (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                double lat1,
                double lon1,
                double h1,
                double lat2,
                double lon2,
                double h2,
                double & dX,
                double & dY,
                double & dZ )
```

ECEF смещение (разница в декартовых ECEF координатах двух точек)

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	lat1	- широта 1 точки
in	lon1	- долгота 1 точки
in	h1	- высота 1 точки
in	lat2	- широта 2 точки
in	lon2	- долгота 2 точки
in	h2	- высота 2 точки
out	dX	- смещение по оси X
out	dY	- смещение по оси Y
out	dZ	- смещение по оси Z

См. определение в файле geodesy.cpp строка 758

8.5.2.13 ECEFtoAER() [1/2]

```
AER SPML::Geodesy::ECEFtoAER (
        const CEllipsoid & ellipsoid,
        const Units::TRangeUnit & rangeUnit,
        const Units::TAngleUnit & angleUnit,
        const XYZ & ecef,
        const Geodetic & anchor )
```

Перевод AER координат относительно опорной точки в глобальные декартовые

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	ecef	- ECEF глобальные декратовы координаты
in	anchor	- геодезические координаты опорной точки

Возвращает

AER координаты точки в ECEF координатах относительно опорной точки

См. определение в файле geodesy.cpp строка 1631

8.5.2.14 ECEFtoAER() [2/2]

```
void SPML::Geodesy::ECEFtoAER (
        const CEllipsoid & ellipsoid,
        const Units::TRangeUnit & rangeUnit,
        const Units::TAngleUnit & angleUnit,
        double x,
        double y,
        double z,
        double lat0,
        double lon0,
        double h0,
        double & az,
        double & elev,
        double & slantRange )
```

Перевод AER координат относительно опорной точки в глобальные декартовые

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности

Аргументы

in	angleUnit	- единицы измерения углов
in	x	- ECEF координата X
in	y	- ECEF координата Y
in	z	- ECEF координата X
in	lat0	- широта опорной точки
in	lon0	- долгота опорной точки
in	h0	- высота опорной точки
out	az	- азимут из опорной точки на искомую точку
out	elev	- угол места из опорной точки на искомую точку
out	slantRange	- наклонная дальность от опорной точки до искомой точки

См. определение в файле geodesy.cpp строка 1564

8.5.2.15 ECEFtoENU() [1/2]

```
ENU SPML::Geodesy::ECEFtoENU (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                const XYZ & ecef,
                const Geodetic & point )
```

Перевод ECEF координат точки в ENU относительно географических координат опорной точки point.

https://gssc.esa.int/navipedia/index.php/Transformations_between_ECEF_and_ENU_coordinates

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	ecef	- ECEF координаты
in	point	- опорная точка

Возвращает

Координаты ENU точки point

См. определение в файле geodesy.cpp строка 924

8.5.2.16 ECEFtoENU() [2/2]

```
void SPML::Geodesy::ECEFtoENU (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                double x,
                double y,
                double z,
                double lat,
                double lon,
                double h,
                double & xEast,
                double & yNorth,
                double & zUp )
```

Перевод ECEF координат точки в ENU относительно географических координат опорной точки (lat, lon)

https://gssc.esa.int/navipedia/index.php/Transformations_between_ECEF_and_ENU_coordinates

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	x	- ECEF координата X
in	y	- ECEF координата Y
in	z	- ECEF координата Z
in	lat	- широта опорной точки
in	lon	- долгота опорной точки
in	h	- высота опорной точки
out	xEast	- ENU координата X (East)
out	yNorth	- ENU координата Y (North)
out	zUp	- ENU координата X (Up)

См. определение в файле geodesy.cpp строка 853

8.5.2.17 ECEFtoENUV() [1/2]

```
ENU SPML::Geodesy::ECEFtoENUV (
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                const XYZ & shift,
                const Geographic & point )
```

Перевод ECEF координат точки в ENU относительно географических координат point.

Аргументы

in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	shift	- смещение по декартовым осям
in	point	- точка

Возвращает

Координаты ENU точки point

См. определение в файле geodesy.cpp строка 996

8.5.2.18 ECEFtoENUV() [2/2]

```
void SPML::Geodesy::ECEFtoENUV (
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                double dX,
                double dY,
                double dZ,
                double lat,
                double lon,
                double & xEast,
                double & yNorth,
                double & zUp )
```

Перевод ECEF координат точки в ENU относительно географических координат (lat, lon)

Аргументы

in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	dX	- смещение по оси X
in	dY	- смещение по оси Y
in	dZ	- смещение по оси Z
in	lat	- широта точки
in	lon	- долгота точки
out	xEast	- ENU координата X (East)
out	yNorth	- ENU координата Y (North)
out	zUp	- ENU координата X (Up)

См. определение в файле geodesy.cpp строка 932

8.5.2.19 ECEFtoGEO() [1/2]

```
void SPML::Geodesy::ECEFtoGEO (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                double x,
                double y,
                double z,
                double & lat,
                double & lon,
                double & h )
```

Пересчет декартовых геоцентрических координат в широту, долготу, высоту

Декартовые геоцентрические координаты (ECEF): ось X - через пересечение гринвичского меридиана и экватора, ось Y - через пересечение меридиана 90 [град] восточной долготы и экватора, ось Z - через северный полюс
Источник - Olson, D. K. (1996). Converting Earth-Centered, Earth-Fixed Coordinates to Geodetic Coordinates. IEEE Transactions on Aerospace and Electronic Systems, 32(1), 473–476. https://doi.↩org/10.1109/7.481290

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	x	- координата по оси X
in	y	- координата по оси Y
in	z	- координата по оси Z
out	lat	- широта точки
out	lon	- долгота точки
out	h	- высота точки над поверхностью эллипсоида

См. определение в файле geodesy.cpp строка 616

8.5.2.20 ECEFtoGEO() [2/2]

```
Geodetic SPML::Geodesy::ECEFtoGEO (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                XYZ & point )
```

Пересчет декартовых геоцентрических координат в широту, долготу, высоту

Декартовые геоцентрические координаты (ECEF): ось X - через пересечение гринвичского меридиана и экватора, ось Y - через пересечение меридиана 90 [град] восточной долготы и экватора, ось Z - через северный полюс
Источник - Olson, D. K. (1996). Converting Earth-Centered, Earth-Fixed Coordinates to Geodetic Coordinates. IEEE Transactions on Aerospace and Electronic Systems, 32(1), 473–476. https://doi.↩org/10.1109/7.481290

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	point	- точка в координатах ECEF

Возвращает

Геодезические (широта, долгота, высота) координаты точки point

См. определение в файле geodesy.cpp строка 735

8.5.2.21 ENUtoAER() [1/2]

```
AER SPML::Geodesy::ENUtoAER (
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                const ENU & point )
```

Перевод ENU координат точки в AER координаты

Аргументы

in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	point	- точка в координатах ENU

Возвращает

Координаты AER точки point

См. определение в файле geodesy.cpp строка 1131

8.5.2.22 ENUtoAER() [2/2]

```
void SPML::Geodesy::ENUtoAER (
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                double xEast,
                double yNorth,
                double zUp,
                double & az,
                double & elev,
                double & slantRange )
```

Перевод ENU координат точки в AER координаты

Аргументы

in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	xEast	- ENU координата X (East)
in	yNorth	- ENU координата Y (North)
in	zUp	- ENU координата X (Up)
out	az	- азимут
out	elev	- угол места
out	slantRange	- наклонная дальность

См. определение в файле geodesy.cpp строка 1079

8.5.2.23 ENUtoECEF() [1/2]

```
XYZ SPML::Geodesy::ENUtoECEF (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                const ENU & enu,
                const Geodetic & point )
```

Перевод ENU координат точки в ECEF относительно географических координат точки point.

https://gssc.esa.int/navipedia/index.php/Transformations_between_ECEF_and_ENU_coordinates

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	enu	- East, North, Up координаты точки
in	point	- точка

Возвращает

Координаты ECEF точки point

См. определение в файле geodesy.cpp строка 1070

8.5.2.24 ENUtoECEF() [2/2]

```
void SPML::Geodesy::ENUtoECEF (
                const CEllipsoid & ellipsoid,
```

const Units::TRangeUnit & rangeUnit,
const Units::TAngleUnit & angleUnit,
double e,
double n,
double u,
double lat,
double lon,
double h,
double & x,
double & y,
double & z)

Перевод ENU координат точки в ECEF относительно географических координат опорной точки (lat, lon)

https://gssc.esa.int/navipedia/index.php/Transformations_between_ECEF_and_ENU_coordinates

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	e	- East
in	n	- North
in	u	- Up
in	lat	- широта опорной точки
in	lon	- долгота опорной точки
in	h	- высота опорной точки
out	x	- ECEF координата X
out	y	- ECEF координата Y
out	z	- ECEF координата X

См. определение в файле geodesy.cpp строка 1004

8.5.2.25 ENUtoGEO() [1/2]

Geodetic SPML::Geodesy::ENUtoGEO (
const CEllipsoid & ellipsoid,
const Units::TRangeUnit & rangeUnit,
const Units::TAngleUnit & angleUnit,
const ENU & point,
const Geodetic & anchor)

Перевод координат ENU в геодезические координаты GEO относительно опорной точки

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	point	- ENU координаты точки
in	anchor	- геодезические координаты опорной точки

Возвращает

геодезические координаты точки point

См. определение в файле geodesy.cpp строка 1334

8.5.2.26 ENUtoGEO() [2/2]

void SPML::Geodesy::ENUtoGEO (
const CEllipsoid & ellipsoid,
const Units::TRangeUnit & rangeUnit,
const Units::TAngleUnit & angleUnit,
double xEast,
double yNorth,
double zUp,
double lat0,
double lon0,
double h0,
double & lat,
double & lon,
double & h)

Перевод координат ENU в геодезические координаты GEO относительно опорной точки

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	xEast	- ENU координата X (East)
in	yNorth	- ENU координата Y (North)
in	zUp	- ENU координата X (Up)
in	lat0	- широта опорной точки
in	lon0	- долгота опорной точки
in	h0	- высота опорной точки
out	lat	- широта точки
out	lon	- долгота точки
out	h	- высота точки

См. определение в файле geodesy.cpp строка 1268

8.5.2.27 ENUtoUVW() [1/2]

UVW SPML::Geodesy::ENUtoUVW (
const CEllipsoid & ellipsoid,
const Units::TRangeUnit & rangeUnit,

const Units::TAngleUnit & angleUnit,
const ENU & enu,
const Geographic & point)

Перевод ENU координат точки в UVW координаты

https://gssc.esa.int/navipedia/index.php/Transformations_between_ECEF_and_ENU_coordinates

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	enu	- East, North, Up координаты точки
in	point	- точка

Возвращает

Координаты ECEF точки point

См. определение в файле geodesy.cpp строка 1692

8.5.2.28 ENUtoUVW() [2/2]

void SPML::Geodesy::ENUtoUVW (
const CEllipsoid & ellipsoid,
const Units::TRangeUnit & rangeUnit,
const Units::TAngleUnit & angleUnit,
double xEast,
double yNorth,
double zUp,
double lat0,
double lon0,
double & u,
double & v,
double & w)

Перевод ENU координат точки в UVW координаты

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	xEast	- East
in	yNorth	- North
in	zUp	- Up
in	lat0	- широта опорной точки
in	lon0	- долгота опорной точки
out	u	- U координата
out	v	- V координата
out	w	- W координата

См. определение в файле geodesy.cpp строка 1639

8.5.2.29 GEOtoAER() [1/2]

```
AER SPML::Geodesy::GEOtoAER (
        const CEllipsoid & ellipsoid,
        const Units::TRangeUnit & rangeUnit,
        const Units::TAngleUnit & angleUnit,
        const Geodetic & point1,
        const Geodetic & point2 )
```

Вычисление AER координат между двумя геодезическими точками

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	point1	- геодезические координаты 1 точки
in	point2	- геодезические координаты 2 точки

Возвращает

Координаты AER между точками point1 и point2

См. определение в файле geodesy.cpp строка 1410

8.5.2.30 GEOtoAER() [2/2]

```
void SPML::Geodesy::GEOtoAER (
        const CEllipsoid & ellipsoid,
        const Units::TRangeUnit & rangeUnit,
        const Units::TAngleUnit & angleUnit,
        double lat1,
        double lon1,
        double h1,
        double lat2,
        double lon2,
        double h2,
        double & az,
        double & elev,
        double & slantRange )
```

Вычисление AER координат между двумя геодезическими точками

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	lat1	- широта 1 точки
in	lon1	- долгота 1 точки
in	h1	- высота 1 точки
in	lat2	- широта 2 точки
in	lon2	- долгота 2 точки
in	h2	- высота 2 точки
out	az	- азимут из 1 точки на 2 точку
out	elev	- угол места из 1 точки на 2 точку
out	slantRange	- наклонная дальность

См. определение в файле geodesy.cpp строка 1343

8.5.2.31 GEOtoECEF() [1/2]

```
XYZ SPML::Geodesy::GEOtoECEF (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                const Geodetic point )
```

Пересчет широты, долготы, высоты в декартовые геоцентрические координаты

При отсутствии высоты или расположении точки на поверхности эллипсоида, задать координату высоты h = 0. Декартовые геоцентрические координаты (ECEF): ось X - через пересечение гринвичского меридиана и экватора, ось Y - через пересечение меридиана 90 [град] восточной долготы и экватора, ось Z - через северный полюс.
Источник - Морозов В.П. Курс сфероидической геодезии. Изд. 2, перераб и доп. М.,Недра, 1979, 296 с., стр 191

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	point	- геодезические координаты точки

Возвращает

ECEF координаты точки point

См. определение в файле geodesy.cpp строка 478

8.5.2.32 GEOtoECEF() [2/2]

```
void SPML::Geodesy::GEOtoECEF (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                double lat,
                double lon,
                double h,
                double & x,
                double & y,
                double & z )
```

Пересчет широты, долготы, высоты в декартовые геоцентрические координаты

При отсутствии высоты или расположении точки на поверхности эллипсоида, задать координату высоты h = 0. Декартовые геоцентрические координаты (ECEF): ось X - через пересечение гринвичского меридиана и экватора, ось Y - через пересечение меридиана 90 [град] восточной долготы и экватора, ось Z - через северный полюс.
Источник - Морозов В.П. Курс сфероидической геодезии. Изд. 2, перераб и доп. М.,Недра, 1979, 296 с., стр 191

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	lat	- широта точки
in	lon	- долгота точки
in	h	- высота точки над поверхностью эллипсоида
out	x	- координата по оси X
out	y	- координата по оси Y
out	z	- координата по оси Z

См. определение в файле geodesy.cpp строка 409

8.5.2.33 GEOtoENU() [1/2]

```
ENU SPML::Geodesy::GEOtoENU (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                const Geodetic & point,
                const Geodetic & anchor )
```

Перевод геодезических координат GEO точки point в координаты ENU относительно опорной точки

Аргументы

in	ellipsoid	- земной эллипсоид

Аргументы

in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	point	- геодезические координаты точки
in	anchor	- геодезические координаты опорной точки, относительно которой переводим

Возвращает

Координаты ENU точки point

См. определение в файле geodesy.cpp строка 1259

8.5.2.34 GEOtoENU() [2/2]

```
void SPML::Geodesy::GEOtoENU (
        const CEllipsoid & ellipsoid,
        const Units::TRangeUnit & rangeUnit,
        const Units::TAngleUnit & angleUnit,
        double lat,
        double lon,
        double h,
        double lat0,
        double lon0,
        double h0,
        double & xEast,
        double & yNorth,
        double & zUp )
```

Перевод геодезических координат GEO точки point в координаты ENU относительно опорной точки

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	lat	- широта точки
in	lon	- долгота точки
in	h	- высота точки
in	lat0	- широта опорной точки
in	lon0	- долгота опорной точки
in	h0	- высота опорной точки
out	xEast	- ENU координата X (East)
out	yNorth	- ENU координата Y (North)
out	zUp	- ENU координата X (Up)

См. определение в файле geodesy.cpp строка 1197

8.5.2.35 GEOtoRAD() [1/2]

```
RAD SPML::Geodesy::GEOtoRAD (
        const CEllipsoid & ellipsoid,
        const Units::TRangeUnit & rangeUnit,
        const Units::TAngleUnit & angleUnit,
        const Geographic & start,
        const Geographic & end )
```

Пересчет географических координат в радиолокационные (Обратная геодезическая задача)

Расчет на эллипсоиде по формулам Винсента:
Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics on the Ellipsoid with application of nested equations". Survey Review. XXIII (176): 88–93.

Расчет на сфере:
Морозов В.П. Курс сфероидической геодезии. Изд. 2, перераб и доп. М.,Недра, 1979, 296 с., стр 97-100

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	start	- начальная точка
in	end	- конечная точка

Возвращает

Радиолокационные координаты (расстояние между начальной и конечной точками по ортодроме, азимут из начальной точки на конечную, азимут в конечной точке)

См. определение в файле geodesy.cpp строка 229

8.5.2.36 GEOtoRAD() [2/2]

```
void SPML::Geodesy::GEOtoRAD (
        const CEllipsoid & ellipsoid,
        const Units::TRangeUnit & rangeUnit,
        const Units::TAngleUnit & angleUnit,
        double latStart,
        double lonStart,
        double latEnd,
        double lonEnd,
        double & d,
        double & az,
        double & azEnd = dummy_double )
```

Пересчет географических координат в радиолокационные (Обратная геодезическая задача)

Расчет на эллипсоиде по формулам Винсента:
Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics on the Ellipsoid with application of nested equations". Survey Review. XXIII (176): 88–93.

Расчет на сфере:
Морозов В.П. Курс сфероидической геодезии. Изд. 2, перераб и доп. М.,Недра, 1979, 296 с., стр 97-100

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	latStart	- широта начальной точки
in	lonStart	- долгота начальной точки
in	latEnd	- широта конечной точки
in	lonEnd	- долгота конечной точки
out	d	- расстояние между начальной и конечной точками по ортодроме
out	az	- азимут из начальной точки на конечную
out	azEnd	- азимут в конечной точке

См. определение в файле geodesy.cpp строка 61

8.5.2.37 RADtoGEO() [1/2]

```
Geographic SPML::Geodesy::RADtoGEO (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                const Geographic & start,
                const RAD & rad,
                double & azEnd = dummy_double )
```

Пересчет радиолокационных координат в географические (Прямая геодезическая задача)

Расчет на эллипсоиде по формулам Винсента:
Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics on the Ellipsoid with application of nested equations". Survey Review. XXIII (176): 88–93.

Расчет на сфере:
Морозов В.П. Курс сфероидической геодезии. Изд. 2, перераб и доп. М.,Недра, 1979, 296 с., стр 97-100

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	start	- географические координаты начальной точки
in	rad	- радиолокационные координаты пути (расстояние между начальной и конечной точками по ортодроме, азимут из начальной точки на конечную)
out	azEnd	- прямой азимут в конечной точке

Возвращает

Географические координаты конечной точки

См. определение в файле geodesy.cpp строка 400

8.5.2.38 RADtoGEO() [2/2]

```
void SPML::Geodesy::RADtoGEO (
                const CEllipsoid & ellipsoid,
                const Units::TRangeUnit & rangeUnit,
                const Units::TAngleUnit & angleUnit,
                double latStart,
                double lonStart,
                double d,
                double az,
                double & latEnd,
                double & lonEnd,
                double & azEnd = dummy_double )
```

Пересчет радиолокационных координат в географические (Прямая геодезическая задача)

Расчет на эллипсоиде по формулам Винсента:
Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics on the Ellipsoid with application of nested equations". Survey Review. XXIII (176): 88–93.

Расчет на сфере:
Морозов В.П. Курс сфероидической геодезии. Изд. 2, перераб и доп. М.,Недра, 1979, 296 с., стр 97-100

Аргументы

in	ellipsoid	- земной эллипсоид
in	rangeUnit	- единицы измерения дальности
in	angleUnit	- единицы измерения углов
in	latStart	- широта начальной точки
in	lonStart	- долгота начальной точки
in	d	- расстояние между начальной и конечной точками по ортодроме
in	az	- азимут из начальной точки на конечную
out	latEnd	- широта конечной точки
out	lonEnd	- долгота конечной точки
out	azEnd	- прямой азимут в конечной точке

См. определение в файле geodesy.cpp строка 238

8.5.2.39 VectorFromTwoPoints() [1/2]

```
XYZ SPML::Geodesy::VectorFromTwoPoints (
                const XYZ & point1,
                const XYZ & point2 )
```

Вектор, полученный из координат двух точек

Предполагается, что результирующий вектор начинается в точке (0, 0, 0)

Аргументы

in	point1	- 1 точка
in	point2	- 2 точка

Возвращает

Вектор, полученный из координат двух точек

См. определение в файле geodesy.cpp строка 1736

8.5.2.40 VectorFromTwoPoints() [2/2]

```
void SPML::Geodesy::VectorFromTwoPoints (
                double x1,
                double y1,
                double z1,
                double x2,
                double y2,
                double z2,
                double & xV,
                double & yV,
                double & zV )
```

Вектор из координат двух точек

Предполагается, что результирующий вектор начинается в точке (0, 0, 0)

Аргументы

in	x1	- X координата 1 точки
in	y1	- Y координата 1 точки
in	z1	- Z координата 1 точки
in	x2	- X координата 2 точки
in	y2	- Y координата 2 точки
in	z2	- Z координата 2 точки
out	xV	- X координата вектора с началом в точке 1 и концом в точке 2
out	yV	- Y координата вектора с началом в точке 1 и концом в точке 2
out	zV	- Z координата вектора с началом в точке 1 и концом в точке 2

См. определение в файле geodesy.cpp строка 1729

8.5.2.41 XYZtoDistance() [1/2]

```
double SPML::Geodesy::XYZtoDistance (
                const XYZ & point1,
                const XYZ & point2 )
```

Вычисление расстояния между точками в декартовых координатах

Вычисляется как $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$

Аргументы

in	point1	- 1 точка
in	point2	- 2 точка

Возвращает

Расстояние между двумя точками в декартовых координатах

См. определение в файле geodesy.cpp строка 753

8.5.2.42 XYZtoDistance() [2/2]

```
double SPML::Geodesy::XYZtoDistance (
                double x1,
                double y1,
                double z1,
                double x2,
                double y2,
                double z2 )
```

Вычисление расстояния между точками в декартовых координатах

Вычисляется как $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$

Внимание

Единицы измерения выхода соответствуют единицам измерения входа

Аргументы

in	x1	- координата первой точки по оси X
in	y1	- координата первой точки по оси Y
in	z1	- координата первой точки по оси Z
in	x2	- координата второй точки по оси X
in	y2	- координата второй точки по оси Y
in	z2	- координата второй точки по оси Z

Возвращает

Расстояние между двумя точками в декартовых координатах

См. определение в файле geodesy.cpp строка 743

8.5.3 Переменные

8.5.3.1 dummy_double

double SPML::Geodesy::dummy_double [static]

См. определение в файле geodesy.h строка 424

8.6 Пространство имен SPML::Geodesy::Ellipsoids

Земные эллипсоиды

Функции

- static CEllipsoid WGS84 ()
  Эллипсоид WGS84 (EPSG:7030)
- static CEllipsoid GRS80 ()
  Эллипсоид GRS80 (EPSG:7019)
- static CEllipsoid PZ90 ()
  Эллипсоид ПЗ-90 (EPSG:7054)
- static CEllipsoid Krassowsky1940 ()
  Эллипсоид Красовского 1940 (EPSG:7024)
- static CEllipsoid Sphere6371 ()
  Сфера радиусом 6371000.0 [м] (EPSG:7035)
- static CEllipsoid Sphere6378 ()
  Сфера радиусом 6378000.0 [м].
- static CEllipsoid SphereKrassowsky1940 ()
  Сфера радиусом большой полуоси эллипсоида Красовского 1940 (EPSG:7024)
- static const __attribute__ ((unused)) std
  Возвращает доступные предопределенные эллипсоидоы

8.6.1 Подробное описание

Земные эллипсоиды

8.6.2 Функции

8.6.2.1 __attribute__()

static const SPML::Geodesy::Ellipsoids::__attribute__ (
(unused)) [static]

Возвращает доступные предопределенные эллипсоидоы

Возвращает

Вектор предопределенных эллипсоидов

См. определение в файле geodesy.h строка 225

8.6.2.2 GRS80()

static CEllipsoid SPML::Geodesy::Ellipsoids::GRS80 ( ) [static]

Эллипсоид GRS80 (EPSG:7019)

Главная полуось 6378137.0, обратное сжатие 298.257222101

См. определение в файле geodesy.h строка 168

8.6.2.3 Krassowsky1940()

static CEllipsoid SPML::Geodesy::Ellipsoids::Krassowsky1940 ( ) [static]

Эллипсоид Красовского 1940 (EPSG:7024)

Главная полуось 6378245.0, обратное сжатие 298.3

См. определение в файле geodesy.h строка 186

8.6.2.4 PZ90()

static CEllipsoid SPML::Geodesy::Ellipsoids::PZ90 ( ) [static]

Эллипсоид ПЗ-90 (EPSG:7054)

Главная полуось 6378136.0, обратное сжатие 298.257839303

См. определение в файле geodesy.h строка 177

8.6.2.5 Sphere6371()

static CEllipsoid SPML::Geodesy::Ellipsoids::Sphere6371 ( ) [static]

Сфера радиусом 6371000.0 [м] (EPSG:7035)

Обратное сжатие - бесконечность

См. определение в файле geodesy.h строка 196

8.6.2.6 Sphere6378()

static CEllipsoid SPML::Geodesy::Ellipsoids::Sphere6378 ( ) [static]

Сфера радиусом 6378000.0 [м].

Обратное сжатие - бесконечность

См. определение в файле geodesy.h строка 206

8.6.2.7 SphereKrassowsky1940()

static CEllipsoid SPML::Geodesy::Ellipsoids::SphereKrassowsky1940 ( ) [static]

Сфера радиусом большой полуоси эллипсоида Красовского 1940 (EPSG:7024)

Обратное сжатие - бесконечность

См. определение в файле geodesy.h строка 215

8.6.2.8 WGS84()

static CEllipsoid SPML::Geodesy::Ellipsoids::WGS84 ( ) [static]

Эллипсоид WGS84 (EPSG:7030)

Главная полуось 6378137.0, обратное сжатие 298.257223563

См. определение в файле geodesy.h строка 159

8.7 Пространство имен SPML::Units

Единицы измерения физических величин, форматы чисел

Перечисления

- enum TNumberFormat : int { NF_Fixed = 0 , NF_Scientific = 1 }
  Формат числа
- enum TAngleUnit : int { AU_Radian = 0 , AU_Degree = 1 }
  Размерность угловых единиц
- enum TRangeUnit : int { RU_Meter = 0 , RU_Kilometer = 1 }
  Размерность единиц дальности

8.7.1 Подробное описание

Единицы измерения физических величин, форматы чисел

8.7.2 Перечисления

8.7.2.1 TAngleUnit

enum SPML::Units::TAngleUnit : int

Размерность угловых единиц

Элементы перечислений

AU_Radian	Радиан
AU_Degree	Градус

См. определение в файле units.h строка 31

8.7.2.2 TNumberFormat

enum SPML::Units::TNumberFormat : int

Формат числа

Элементы перечислений

NF_Fixed	Отображение фиксированного числа знаков после запятой
NF_Scientific	Отображение в научном формате 1e+000.

См. определение в файле units.h строка 22

8.7.2.3 TRangeUnit

enum SPML::Units::TRangeUnit : int

Размерность единиц дальности

Элементы перечислений

RU_Meter	Метр
RU_Kilometer	Километр

См. определение в файле units.h строка 40

Раздел 9

Классы

9.1 Структура SPML::Geodesy::AER

Локальные сферические координаты AER (Azimuth-Elevation-Range, Азимут-Угол места-Дальность)

#include <geodesy.h>

Открытые члены

- AER ()
  Конструктор по умолчанию
- AER (double a, double e, double r)
  Параметрический конструктор

Открытые атрибуты

- double A
  Азимут
- double E
  Угол места
- double R
  Дальность

9.1.1 Подробное описание

Локальные сферические координаты AER (Azimuth-Elevation-Range, Азимут-Угол места-Дальность)

См. определение в файле geodesy.h строка 397

9.1.2 Конструктор(ы)

9.1.2.1 AER() [1/2]

SPML::Geodesy::AER::AER ( ) [inline]

Конструктор по умолчанию

См. определение в файле geodesy.h строка 406

9.1.2.2 AER() [2/2]

SPML::Geodesy::AER::AER (
double a,
double e,
double r) [inline]

Параметрический конструктор

Аргументы

a	- азимут
e	- угол места
r	- дальность

См. определение в файле geodesy.h строка 415

9.1.3 Данные класса

9.1.3.1 A

double SPML::Geodesy::AER::A

Азимут

См. определение в файле geodesy.h строка 399

9.1.3.2 E

double SPML::Geodesy::AER::E

Угол места

См. определение в файле geodesy.h строка 400

9.1.3.3 R

double SPML::Geodesy::AER::R

Дальность

См. определение в файле geodesy.h строка 401

Объявления и описания членов структуры находятся в файле:

- geodesy.h

9.2 Структура CCoordCalcSettings

Настройки программы

Открытые члены

- CCoordCalcSettings ()
  Конструктор по умолчанию

Открытые атрибуты

- int Precision
  Число цифр после запятой при печати в консоль результата
- SPML::Units::TAngleUnit AngleUnit
  Единицы измерения углов
- SPML::Units::TRangeUnit RangeUnit
  Единицы измерения дальностей
- int EllipsoidNumber
  Эллипсоид на котором решаем геодезические задачи
- std::vector< double > Input
  Входной массив

9.2.1 Подробное описание

Настройки программы

См. определение в файле main_geocalc.cpp строка 49

9.2.2 Конструктор(ы)

9.2.2.1 CCoordCalcSettings()

CCoordCalcSettings::CCoordCalcSettings ( ) [inline]

Конструктор по умолчанию

См. определение в файле main_geocalc.cpp строка 64

9.2.3 Данные класса

9.2.3.1 AngleUnit

SPML::Units::TAngleUnit CCoordCalcSettings::AngleUnit

Единицы измерения углов

См. определение в файле main_geocalc.cpp строка 52

9.2.3.2 EllipsoidNumber

int CCoordCalcSettings::EllipsoidNumber

Эллипсоид на котором решаем геодезические задачи

См. определение в файле main_geocalc.cpp строка 58

9.2.3.3 Input

std::vector<double> CCoordCalcSettings::Input

Входной массив

См. определение в файле main_geocalc.cpp строка 59

9.2.3.4 Precision

int CCoordCalcSettings::Precision

Число цифр после запятой при печати в консоль результата

См. определение в файле main_geocalc.cpp строка 51

9.2.3.5 RangeUnit

SPML::Units::TRangeUnit CCoordCalcSettings::RangeUnit

Единицы измерения дальностей

См. определение в файле main_geocalc.cpp строка 53

Объявления и описания членов структуры находятся в файле:

- main_geocalc.cpp

9.3 Класс SPML::Geodesy::CEllipsoid

Земной эллипсоид

#include <geodesy.h>

Открытые члены

- std::string Name () const
  Имя эллипсоида
- double A () const
  Большая полуось эллипсоида (экваториальный радиус)
- double B () const
  Малая полуось эллипсоида (полярный радиус)
- double F () const
  Сжатие f = (a - b) / a.
- double Invf () const
  Обратное сжатие Invf = a / (a - b)
- double EccentricityFirst () const
  Первый эксцентриситет эллипсоида e1 = sqrt((a * a) - (b * b)) / a;.
- double EccentricityFirstSquared () const
  Квадрат первого эксцентриситета эллипсоида es1 = 1 - ((b * b) / (a * a));.
- double EccentricitySecond () const
  Второй эксцентриситет эллипсоида e2 = sqrt((a * a) - (b * b)) / b;.
- double EccentricitySecondSquared () const
  Квадрат второго эксцентриситета эллипсоида es2 = ((a * a) / (b * b)) - 1;.
- CEllipsoid ()
  Конструктор по умолчанию
- CEllipsoid (std::string ellipsoidName, double semiMajorAxis, double semiMinorAxis, double inverseFlattening, bool isInvfDef)
  Параметрический конструктор эллипсоида

Закрытые данные

- std::string name

  Название эллипсоида
- double a

  Большая полуось (экваториальный радиус), [м].
- double b

  Малая полуось (полярный радиус) , [м].
- double invf

  Обратное сжатие invf = a / (a - b)
- double f

  Сжатие f = (a - b) / a.

9.3.1 Подробное описание

Земной эллипсоид

См. определение в файле geodesy.h строка 33

9.3.2 Конструктор(ы)

9.3.2.1 CEllipsoid() [1/2]

SPML::Geodesy::CEllipsoid::CEllipsoid ( )

Конструктор по умолчанию

См. определение в файле geodesy.cpp строка 19

9.3.2.2 CEllipsoid() [2/2]

```
SPML::Geodesy::CEllipsoid::CEllipsoid (
                std::string ellipsoidName,
                double semiMajorAxis,
                double semiMinorAxis,
                double inverseFlattening,
                bool isInvfDef )
```

Параметрический конструктор эллипсоида

Аргументы

in	ellipsoidName	- название эллипсоида
in	semiMajorAxis	- большая полуось (экваториальный радиус)
in	semiMinorAxis	- малая полуось (полярный радиус)
in	inverseFlattening	- обратное сжатие invf = a / (a - b)
in	isInvfDef	- обратное сжатие задано (малая полуось расчитана из большой и обратного сжатия)

См. определение в файле geodesy.cpp строка 27

9.3.3 Методы

9.3.3.1 A()

double SPML::Geodesy::CEllipsoid::A ( ) const [inline]

Большая полуось эллипсоида (экваториальный радиус)

Возвращает

Возвращает большую полуось эллипсоида (экваториальный радиус) в [м]

См. определение в файле geodesy.h строка 49

9.3.3.2 B()

double SPML::Geodesy::CEllipsoid::B ( ) const [inline]

Малая полуось эллипсоида (полярный радиус)

Возвращает

Возвращает малую полуось эллипсоида (полярный радиус) в [м]

См. определение в файле geodesy.h строка 58

9.3.3.3 EccentricityFirst()

double SPML::Geodesy::CEllipsoid::EccentricityFirst ( ) const [inline]

Первый эксцентриситет эллипсоида e1 = sqrt((a * a) - (b * b)) / a;.

Возвращает

Возвращает первый эксцентриситет эллипсоида

См. определение в файле geodesy.h строка 85

9.3.3.4 EccentricityFirstSquared()

double SPML::Geodesy::CEllipsoid::EccentricityFirstSquared ( ) const [inline]

Квадрат первого эксцентриситета эллипсоида es1 = 1 - ((b * b) / (a * a));.

Возвращает

Возвращает квадрат первого эксцентриситета эллипсоида

См. определение в файле geodesy.h строка 94

9.3.3.5 EccentricitySecond()

double SPML::Geodesy::CEllipsoid::EccentricitySecond ( ) const [inline]

Второй эксцентриситет эллипсоида e2 = sqrt((a * a) - (b * b)) / b;.

Возвращает

Возвращает второй эксцентриситет эллипсоида

См. определение в файле geodesy.h строка 103

9.3.3.6 EccentricitySecondSquared()

double SPML::Geodesy::CEllipsoid::EccentricitySecondSquared ( ) const [inline]

Квадрат второго эксцентриситета эллипсоида es2 = ((a * a) / (b * b)) - 1;.

Возвращает

Возвращает второй эксцентриситет эллипсоида

См. определение в файле geodesy.h строка 112

9.3.3.7 F()

double SPML::Geodesy::CEllipsoid::F ( ) const [inline]

Сжатие f = (a - b) / a.

Возвращает

Возвращает сжатие

См. определение в файле geodesy.h строка 67

9.3.3.8 Invf()

double SPML::Geodesy::CEllipsoid::Invf ( ) const [inline]

Обратное сжатие Invf = a / (a - b)

Возвращает

Возвращает обратное сжатие

См. определение в файле geodesy.h строка 76

9.3.3.9 Name()

std::string SPML::Geodesy::CEllipsoid::Name ( ) const [inline]

Имя эллипсоида

Возвращает

Возвращает строку с именем эллипсоида

См. определение в файле geodesy.h строка 40

9.3.4 Данные класса

9.3.4.1 a

double SPML::Geodesy::CEllipsoid::a [private]

Большая полуось (экваториальный радиус), [м].

См. определение в файле geodesy.h строка 134

9.3.4.2 b

double SPML::Geodesy::CEllipsoid::b [private]

Малая полуось (полярный радиус) , [м].

См. определение в файле geodesy.h строка 135

9.3.4.3 f

double SPML::Geodesy::CEllipsoid::f [private]

Сжатие f = (a - b) / a.

См. определение в файле geodesy.h строка 137

9.3.4.4 invf

double SPML::Geodesy::CEllipsoid::invf [private]

Обратное сжатие invf = a / (a - b)

См. определение в файле geodesy.h строка 136

9.3.4.5 name

std::string SPML::Geodesy::CEllipsoid::name [private]

Название эллипсоида

См. определение в файле geodesy.h строка 133

Объявления и описания членов классов находятся в файлах:

- geodesy.h
- geodesy.cpp

9.4 Структура SPML::Geodesy::ENU

Координаты ENU (East-North-Up)

#include <geodesy.h>

Открытые члены

- ENU ()
  Конструктор по умолчанию
- ENU (double e, double n, double u)
  Параметрический конструктор

Открытые атрибуты

- double E

  East координата
- double N

  North координата
- double U

  Up координата

9.4.1 Подробное описание

Координаты ENU (East-North-Up)

См. определение в файле geodesy.h строка 345

9.4.2 Конструктор(ы)

9.4.2.1 ENU() [1/2]

SPML::Geodesy::ENU::ENU ( ) [inline]

Конструктор по умолчанию

См. определение в файле geodesy.h строка 354

9.4.2.2 ENU() [2/2]

SPML::Geodesy::ENU::ENU (
double e,
double n,
double u) [inline]

Параметрический конструктор

Аргументы

e	- East координата
n	- North координата
u	- Up координата

См. определение в файле geodesy.h строка 363

9.4.3 Данные класса

9.4.3.1 E

double SPML::Geodesy::ENU::E

East координата

См. определение в файле geodesy.h строка 347

9.4.3.2 N

double SPML::Geodesy::ENU::N

North координата

См. определение в файле geodesy.h строка 348

9.4.3.3 U

double SPML::Geodesy::ENU::U

Up координата

См. определение в файле geodesy.h строка 349

Объявления и описания членов структуры находятся в файле:

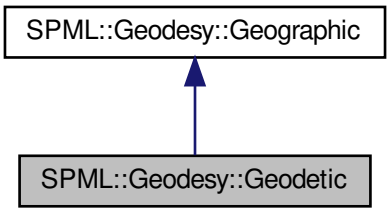
- geodesy.h

9.5 Структура SPML::Geodesy::Geodetic

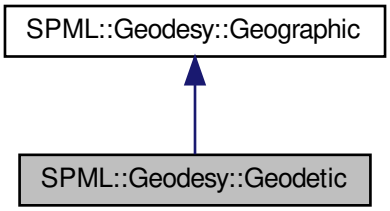
Геодезические координаты (широта, долгота, высота)

#include <geodesy.h>

Граф наследования:SPML::Geodesy::Geodetic:



Граф связей класса SPML::Geodesy::Geodetic:



Открытые члены

- Geodetic ()
  Конструктор по умолчанию
- Geodetic (double lat, double lon, double h)
  Параметрический конструктор

Открытые атрибуты

- double Height
  Высота

9.5.1 Подробное описание

Геодезические координаты (широта, долгота, высота)

См. определение в файле geodesy.h строка 268

9.5.2 Конструктор(ы)

9.5.2.1 Geodetic() [1/2]

SPML::Geodesy::Geodetic::Geodetic ( ) [inline]

Конструктор по умолчанию

См. определение в файле geodesy.h строка 275

9.5.2.2 Geodetic() [2/2]

SPML::Geodesy::Geodetic::Geodetic (
double lat,
double lon,
double h) [inline]

Параметрический конструктор

Аргументы

lat	- широта
lon	- долгота
h	- высота

См. определение в файле geodesy.h строка 284

9.5.3 Данные класса

9.5.3.1 Height

double SPML::Geodesy::Geodetic::Height

Высота

См. определение в файле geodesy.h строка 270

Объявления и описания членов структуры находятся в файле:

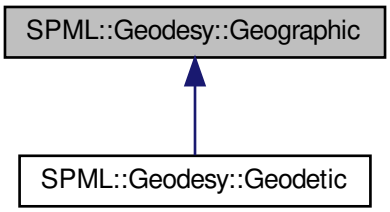
- geodesy.h

9.6 Структура SPML::Geodesy::Geographic

Географические координаты (широта, долгота)

#include <geodesy.h>

Граф наследования:SPML::Geodesy::Geographic:



Открытые члены

- Geographic ()
  Конструктор по умолчанию
- Geographic (double lat, double lon)
  Параметрический конструктор

Открытые атрибуты

- double Lat
  Широта
- double Lon
  Долгота

9.6.1 Подробное описание

Географические координаты (широта, долгота)

См. определение в файле geodesy.h строка 244

9.6.2 Конструктор(ы)

9.6.2.1 Geographic() [1/2]

SPML::Geodesy::Geographic::Geographic ( ) [inline]

Конструктор по умолчанию

См. определение в файле geodesy.h строка 252

9.6.2.2 Geographic() [2/2]

SPML::Geodesy::Geographic::Geographic (
double lat,
double lon) [inline]

Параметрический конструктор

Аргументы

lat	- широта
lon	- долгота

См. определение в файле geodesy.h строка 260

9.6.3 Данные класса

9.6.3.1 Lat

double SPML::Geodesy::Geographic::Lat

Широта

См. определение в файле geodesy.h строка 246

9.6.3.2 Lon

double SPML::Geodesy::Geographic::Lon

Долгота

См. определение в файле geodesy.h строка 247

Объявления и описания членов структуры находятся в файле:

- geodesy.h

9.7 Структура SPML::Geodesy::RAD

Радиолокационные координаты (расстояние по ортодроме, азимут, конечный азимут)

#include <geodesy.h>

Открытые члены

- RAD ()
  Азимут в точке объекта
- RAD (double r, double az, double azEnd)
  Параметрический конструктор

Открытые атрибуты

- double R
- double Az
  Дальность
- double AzEnd
  Азимут в точке наблюдения

9.7.1 Подробное описание

Радиолокационные координаты (расстояние по ортодроме, азимут, конечный азимут)

Имеют два азимута: Az - это начальный азимут в точке наблюдения, AzEnd - конечный азимут (по ортодроме) на дальности R

См. определение в файле geodesy.h строка 293

9.7.2 Конструктор(ы)

9.7.2.1 RAD() [1/2]

SPML::Geodesy::RAD::RAD ( ) [inline]

Азимут в точке объекта

Конструктор по умолчанию

См. определение в файле geodesy.h строка 302

9.7.2.2 RAD() [2/2]

SPML::Geodesy::RAD::RAD (
double r,
double az,
double azEnd) [inline]

Параметрический конструктор

Аргументы

r	- дальность
az	- азимут
azEnd	- конечный азимут

См. определение в файле geodesy.h строка 311

9.7.3 Данные класса

9.7.3.1 Az

double SPML::Geodesy::RAD::Az

Дальность

См. определение в файле geodesy.h строка 296

9.7.3.2 AzEnd

double SPML::Geodesy::RAD::AzEnd

Азимут в точке наблюдения

См. определение в файле geodesy.h строка 297

9.7.3.3 R

double SPML::Geodesy::RAD::R

См. определение в файле geodesy.h строка 295

Объявления и описания членов структуры находятся в файле:

- geodesy.h

9.8 Структура SPML::Geodesy::UVW

Координаты UVW.

#include <geodesy.h>

Открытые члены

- UVW ()
  Конструктор по умолчанию
- UVW (double u, double v, double w)
  Параметрический конструктор

Открытые атрибуты

- double U
  U координата
- double V
  V координата
- double W
  W координата

9.8.1 Подробное описание

Координаты UVW.

См. определение в файле geodesy.h строка 371

9.8.2 Конструктор(ы)

9.8.2.1 UVW() [1/2]

SPML::Geodesy::UVW::UVW ( ) [inline]

Конструктор по умолчанию

См. определение в файле geodesy.h строка 380

9.8.2.2 UVW() [2/2]

SPML::Geodesy::UVW::UVW (
double u,
double v,
double w) [inline]

Параметрический конструктор

Аргументы

u	- U координата
v	- V координата
w	- W координата

См. определение в файле geodesy.h строка 389

9.8.3 Данные класса

9.8.3.1 U

double SPML::Geodesy::UVW::U

U координата

См. определение в файле geodesy.h строка 373

9.8.3.2 V

double SPML::Geodesy::UVW::V

V координата

См. определение в файле geodesy.h строка 374

9.8.3.3 W

double SPML::Geodesy::UVW::W

W координата

См. определение в файле geodesy.h строка 375

Объявления и описания членов структуры находятся в файле:

- geodesy.h

9.9 Структура SPML::Geodesy::XYZ

3D декартовы ортогональные координаты (X, Y, Z)

#include <geodesy.h>

Открытые члены

- XYZ ()
  Конструктор по умолчанию
- XYZ (double x, double y, double z)
  Параметрический конструктор

Открытые атрибуты

- double X
  X координата
- double Y
  Y координата
- double Z
  Z координата

9.9.1 Подробное описание

3D декартовы ортогональные координаты (X, Y, Z)

См. определение в файле geodesy.h строка 319

9.9.2 Конструктор(ы)

9.9.2.1 XYZ() [1/2]

SPML::Geodesy::XYZ::XYZ ( ) [inline]

Конструктор по умолчанию

См. определение в файле geodesy.h строка 328

9.9.2.2 XYZ() [2/2]

SPML::Geodesy::XYZ::XYZ (
double x,
double y,
double z) [inline]

Параметрический конструктор

Аргументы

x	- X координата
y	- Y координата
z	- Z координата

См. определение в файле geodesy.h строка 337

9.9.3 Данные класса

9.9.3.1 X

double SPML::Geodesy::XYZ::X

X координата

См. определение в файле geodesy.h строка 321

9.9.3.2 Y

double SPML::Geodesy::XYZ::Y

Y координата

См. определение в файле geodesy.h строка 322

9.9.3.3 Z

double SPML::Geodesy::XYZ::Z

Z координата

См. определение в файле geodesy.h строка 323

Объявления и описания членов структуры находятся в файле:

- geodesy.h

Раздел 10

Файлы

10.1 Файл main_geocalc.cpp

Консольный геодезический калькулятор

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <boost/program_options.hpp>
#include <spml.h>
```

Классы

- struct CCoordCalcSettings
  Настройки программы

Функции

- static std::string GetVersion ()
  Возвращает строку, содержащую информацию о версии
- template<typename T >
  std::string to_string_with_precision (const T a_value, const int n=6)
  Печать в строку с задаваемым числом знаков после запятой
- int main (int argc, char *argv[])
  main - Основная функция

10.1.1 Подробное описание

Консольный геодезический калькулятор

Дата

21.12.22 - создан

Автор

Соболев А.А.

См. определение в файле main_geocalc.cpp

10.2 main_geocalc.cpp

См. документацию.

```
00001 //----------------------------------------------------------------------------------------------------------------
00012
00013 // System includes:
00014 #include <iostream>
00015 #include <iomanip>
00016 #include <vector>
00017 #include <boost/program_options.hpp>
00018
00019 // SPML includes:
00020 #include <spml.h>
00021
00022 //----------------------------------------------------------------------------------------------------------------
00027 static std::string GetVersion()
00028 {
00029     return "GEOCALC_22.12.2022_v01_Develop";
00030 }
00031
00032 //----------------------------------------------------------------------------------------------------------------
00036 template <typename T>
00037 std::string to_string_with_precision( const T a_value, const int n = 6 )
00038 {
00039     std::ostringstream out;
00040     out.precision(n);
00041     out « std::fixed « a_value;
00042     return out.str();
00043 }
00044
00045 //----------------------------------------------------------------------------------------------------------------
00049 struct CCoordCalcSettings
00050 {
00051     int Precision;
00052     SPML::Units::TAngleUnit AngleUnit;
00053     SPML::Units::TRangeUnit RangeUnit;
00054 //     SPML::Units::TAngleUnit AngleUnitIn;     ///< Единицы измерения углов (входные данные)
00055 //     SPML::Units::TRangeUnit RangeUnitIn;     ///< Единицы измерения дальностей (входные данные)
00056 //     SPML::Units::TAngleUnit AngleUnitOut;    ///< Единицы измерения углов результата (выходные данные)
00057 //     SPML::Units::TRangeUnit RangeUnitOut;    ///< Единицы измерения дальностей результата (выходные данные)
00058     int EllipsoidNumber;
00059     std::vector<double> Input;
00060
00064     CCoordCalcSettings()
00065     {
00066         Precision = 6;
00067         AngleUnit = SPML::Units::TAngleUnit::AU_Degree;
00068         RangeUnit = SPML::Units::TRangeUnit::RU_Kilometer;
00069 //         AngleUnitIn = SPML::Units::TAngleUnit::AU_Degree;
00070 //         RangeUnitIn = SPML::Units::TRangeUnit::RU_Kilometer;
00071 //         AngleUnitOut = SPML::Units::TAngleUnit::AU_Degree;
00072 //         RangeUnitOut = SPML::Units::TRangeUnit::RU_Kilometer;
00073         EllipsoidNumber = 0;
00074         Input.clear();
00075     }
00076 };
00077
00078 //----------------------------------------------------------------------------------------------------------------
00085 int main( int argc, char *argv[] )
00086 {
00087     CCoordCalcSettings settings; // Параметры приложения
00088     auto ellipsoids = SPML::Geodesy::Ellipsoids::GetPredefinedEllipsoids(); // Используемые эллипсоиды
00089     //------------------------------------------------------------------------------------------------------------
00090     // Зададим параметры запуска приложения
00091     namespace po = boost::program_options;
00092     po::options_description desc( "~-= GEODETIC CALCULATOR =-~"
00093         "\n\nРешение геодезических задач и перевод координат (в двойной точности)"
00094         "\nSolve geodetic problems and convert coordinates (double precision)"
00095         "\n\nПараметры/Parameters", 220 ); // 220 - задает ширину строки вывода в терминал
00096     desc.add_options()
00097     // Справочные параметры:
00098     ( "help", "Показать эту справку и выйти/Show this text and exit" )
00099     ( "ver", "Показать версию и выйти/Show version and exit" )
00100     // Задающие параметры:
00101     ( "pr", po::value<int>( &settings.Precision )->default_value( settings.Precision ),
00102         "Число знаков после запятой при печати в косноль/Number of digits after dot while printing to console" )
00103     // Единицы входа дальности/углов
00104     ( "deg", "Вход в градусах (по умолчанию)/Input in degrees (default)" )
00105     ( "rad", "Вход в радианах/Input in radians" )
00106     ( "km", "Вход в километрах (по умолчанию)/Input in kilometers (default)" )
00107     ( "me", "Вход в метрах/Input in meters" )
00108     // Единицы выхода дальности/углов
00109 //     ( "outdeg", "Выход в градусах (по умолчанию)/Output in degrees (default)" )
00110 //     ( "outrad", "Выход в радианах/Output in radians" )
00111 //     ( "outkm", "Выход в километрах (по умолчанию)/Output in kilometers (default)" )
```

```
00112 //    ( "outmeter", "Выход в метрах/Output in meters" )
00113     // На каком эллипсоиде считать
00114 //    ( "el", po::value<int>( &settings.EllipsoidNumber )->default_value( settings.EllipsoidNumber ),
00115 //        ellipsoidsStringAll.c_str() )
00116     ( "el", po::value<std::string>()->default_value( "wgs84" ), "Доступыне эллипсоиды/Avaliable ellipsoids: wgs84,
      grs80, pz90, krasovsky1940, sphere6371, sphere6378" )
00117     ( "els", "Показать список доступных эллипсоидов и их параметры" )
00118     // Проверка
00119     ( "check", "Проверка решением обратной задачи/Check by solving inverse task" )
00120     // Задачи:
00121     ( "geo2rad", po::value<std::vector<double»( &settings.Input )->multitoken(),
00122         "args: LatStart LonStart LatEnd LonEnd" )
00123     ( "rad2geo", po::value<std::vector<double»( &settings.Input )->multitoken(),
00124         "args: LatStart LonStart Range Azimuth" )
00125     //
00126     ( "geo2ecef", po::value<std::vector<double»( &settings.Input )->multitoken(),
00127         "args: Lat Lon Height" )
00128     ( "ecef2geo", po::value<std::vector<double»( &settings.Input )->multitoken(),
00129         "args: X Y Z" )
00130     //
00131     ( "ecefdist", po::value<std::vector<double»( &settings.Input )->multitoken(),
00132         "args: X1 Y1 Z1 X2 Y2 Z2" )
00133     ( "ecefoffset", po::value<std::vector<double»( &settings.Input )->multitoken(),
00134         "args: X1 Y1 Z1 X2 Y2 Z2" )
00135     //
00136     ( "ecef2enu", po::value<std::vector<double»( &settings.Input )->multitoken(),
00137         "args: X Y Z Lat0 Lon0" )
00138     ( "enu2ecef", po::value<std::vector<double»( &settings.Input )->multitoken(),
00139         "args: E N U Lat0 Lon0" )
00140     //
00141     ( "enu2aer", po::value<std::vector<double»( &settings.Input )->multitoken(),
00142         "args: E N U" )
00143     ( "aer2enu", po::value<std::vector<double»( &settings.Input )->multitoken(),
00144         "args: A E R" )
00145     //
00146     ( "geo2enu", po::value<std::vector<double»( &settings.Input )->multitoken(),
00147         "args: Lat Lon Height Lat0 Lon0 Height0" )
00148     ( "enu2geo", po::value<std::vector<double»( &settings.Input )->multitoken(),
00149         "args: E N U Lat0 Lon0 Height0" )
00150     //
00151     ( "geo2aer", po::value<std::vector<double»( &settings.Input )->multitoken(),
00152         "args: Lat Lon Height Lat0 Lon0 Height0" )
00153     ( "aer2geo", po::value<std::vector<double»( &settings.Input )->multitoken(),
00154         "args: A E R Lat0 Lon0 Height0" )
00155     //
00156     ( "ecef2aer", po::value<std::vector<double»( &settings.Input )->multitoken(),
00157         "args: X Y Z Lat0 Lon0 Height0" )
00158     ( "aer2ecef", po::value<std::vector<double»( &settings.Input )->multitoken(),
00159         "args: A E R Lat0 Lon0 Height0" )
00160     ;
00161     po::options_description cla; // Аргументы командной строки (command line arguments)
00162     cla.add( desc );
00163     po::variables_map vm;
00164 //    po::store( po::command_line_parser( argc, argv ).options( cla ).run(), vm );
00165     po::store( po::parse_command_line( argc, argv, cla, po::command_line_style::unix_style ^
      po::command_line_style::allow_short ), vm );
00166     po::notify( vm );
00167     //-------------------------------------------------------------------------------------------------------
00168     // Обработаем аргументы запуска приложения
00169     //-------------------------------------------------------------------------------------------------------
00170     if( vm.count( "help" ) ) {
00171         std::cout « desc « std::endl;
00172         return EXIT_SUCCESS;
00173     }
00174     if( vm.count( "ver" ) ) {
00175         std::cout « SPML::GetVersion() « std::endl;
00176         std::cout « GetVersion() « std::endl;
00177         return EXIT_SUCCESS;
00178     }
00179     if( vm.count( "pr" ) ) {
00180         settings.Precision = vm["pr"].as<int>();
00181     }
00182     //-------------------------------------------------------------------------------------------------------
00183     // Единицы углов
00184     if( vm.count( "deg" ) ) {
00185         settings.AngleUnit = SPML::Units::AU_Degree;
00186     }
00187     if( vm.count( "rad" ) ) {
00188         settings.AngleUnit = SPML::Units::AU_Radian;
00189     }
00190 //    if( vm.count( "outkm" ) ) {
00191 //        settings.AngleUnitOut = SPML::Units::AU_Degree;
00192 //    }
00193 //    if( vm.count( "outmeter" ) ) {
00194 //        settings.AngleUnitOut = SPML::Units::AU_Radian;
00195 //    }
00196     //-------------------------------------------------------------------------------------------------------
```

```
00197     // Единицы расстояния
00198     if( vm.count( "km" ) ) {
00199         settings.RangeUnit = SPML::Units::RU_Kilometer;
00200     }
00201     if( vm.count( "me" ) ) {
00202         settings.RangeUnit = SPML::Units::RU_Meter;
00203     }
00204 //    if( vm.count( "outkm" ) ) {
00205 //        settings.RangeUnitOut = SPML::Units::RU_Kilometer;
00206 //    }
00207 //    if( vm.count( "outmeter" ) ) {
00208 //        settings.RangeUnitOut = SPML::Units::RU_Meter;
00209 //    }
00210     //-------------------------------------------------------------------------------------------------------
00211     // Названия единиц расстояния/углов для вывода на печать
00212     std::string outrange;
00213     if( settings.RangeUnit == SPML::Units::RU_Kilometer ) {
00214         outrange = "km";
00215     } else if( settings.RangeUnit == SPML::Units::RU_Meter ) {
00216         outrange = "m";
00217     } else {
00218         assert( false );
00219     }
00220     std::string outangle;
00221     if( settings.AngleUnit == SPML::Units::AU_Degree ) {
00222         outangle = "deg";
00223     } else if( settings.AngleUnit == SPML::Units::AU_Radian ) {
00224         outangle = "rad";
00225     } else {
00226         assert( false );
00227     }
00228     //-------------------------------------------------------------------------------------------------------
00229     // Эллипсоид
00230     if( vm.count( "el" ) ) {
00231         std::string elName = vm["el"].as<std::string>();
00232         if( elName == "wgs84" ) {
00233             settings.EllipsoidNumber = 0;
00234         } else if( elName == "grs80" ) {
00235             settings.EllipsoidNumber = 1;
00236         } else if( elName == "pz90" ) {
00237             settings.EllipsoidNumber = 2;
00238         } else if( elName == "krassowsky1940" ) {
00239             settings.EllipsoidNumber = 3;
00240         } else if( elName == "sphere6371" ) {
00241             settings.EllipsoidNumber = 4;
00242         } else if( elName == "sphere6378" ) {
00243             settings.EllipsoidNumber = 5;
00244         } else if( elName == "spherekrassowsky1940" ) {
00245             settings.EllipsoidNumber = 6;
00246         } else {
00247             std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00248             return EXIT_FAILURE;
00249         }
00250 //        settings.EllipsoidNumber = vm["el"].as<int>();
00251 //        if( settings.EllipsoidNumber < 0 || settings.EllipsoidNumber > ( ellipsoids.size() - 1 ) ) {
00252 //            std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00253 //            return EXIT_FAILURE;
00254 //        }
00255     }
00256     if( vm.count( "els" ) ) {
00257         std::string ellipsoidsString;
00258         for( int i = 0; i < ellipsoids.size(); i++ ) {
00259             ellipsoidsString += ( ellipsoids.at( i ) ).Name() +
00260                 " a=" + std::to_string( ( ellipsoids.at( i ) ).A() )+
00261                 " invf=" + std::to_string( ( ellipsoids.at( i ) ).Invf() );
00262             if( i != ellipsoids.size() - 1 ) {
00263                 ellipsoidsString += "\n";
00264             }
00265         }
00266         std::cout « ellipsoidsString « std::endl;
00267         return EXIT_SUCCESS;
00268     }
00269     //-------------------------------------------------------------------------------------------------------
00270     // Задачи:
00271     //-------------------------------------------------------------------------------------------------------
00272     if( vm.count( "geo2rad" ) ) {
00273         settings.Input = vm["geo2rad"].as<std::vector<double»();
00274         if( settings.Input.size() != 4 ) {
00275             std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00276             return EXIT_FAILURE;
00277         }
00278
00279         double r, az, azend;
00280         SPML::Geodesy::GEOtoRAD( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00281             settings.Input[0], settings.Input[1], settings.Input[2], settings.Input[3], r, az, azend );
00282
00283         std::string result = "R[" + outrange + "] Az[" + outangle + "] AzEnd[" + outangle + "]:\n" +
```

```
00284           to_string_with_precision( r, settings.Precision ) + " " +
00285           to_string_with_precision( az, settings.Precision ) + " " +
00286           to_string_with_precision( azend, settings.Precision );
00287       std::cout « result « std::endl;
00288
00289       if( vm.count( "check" ) ) {
00290           std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
00291           double lat2, lon2, azend2;
00292           SPML::Geodesy::RADtoGEO( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00293               settings.Input[0], settings.Input[1], r, az, lat2, lon2, azend2 );
00294           std::string result2 = "Lat[" + outangle + "] Lon" + outangle + "] AzEnd[" + outangle + "]:\n" +
00295               to_string_with_precision( lat2, settings.Precision ) + " " +
00296               to_string_with_precision( lon2, settings.Precision ) + " " +
00297               to_string_with_precision( azend2, settings.Precision );
00298           std::cout « result2 « std::endl;
00299           std::cout « "\nDelta:" « std::endl;
00300           std::string resultDelta = "Lat[" + outangle + "] Lon" + outangle + "] AzEnd[" + outangle + "]:\n" +
00301               to_string_with_precision( settings.Input[2] - lat2, settings.Precision ) + " " +
00302               to_string_with_precision( settings.Input[3] - lon2, settings.Precision ) + " " +
00303               to_string_with_precision( azend - azend2, settings.Precision );
00304           std::cout « resultDelta « std::endl;
00305       }
00306       return EXIT_SUCCESS;
00307   }
00308   //-----------------------------------------------------------------------------------------------------
00309   if( vm.count( "rad2geo" ) ) {
00310       settings.Input = vm["rad2geo"].as<std::vector<double»();
00311       if( settings.Input.size() != 4 ) {
00312           std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00313           return EXIT_FAILURE;
00314       }
00315
00316       double lat, lon, azend;
00317       SPML::Geodesy::RADtoGEO( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00318           settings.Input[0], settings.Input[1], settings.Input[2], settings.Input[3], lat, lon, azend );
00319
00320       std::string result = "Lat[" + outangle + "] Lon[" + outangle + "] AzEnd[" + outangle + "]:\n" +
00321           to_string_with_precision( lat, settings.Precision ) + " " +
00322           to_string_with_precision( lon, settings.Precision ) + " " +
00323           to_string_with_precision( azend, settings.Precision );
00324       std::cout « result « std::endl;
00325
00326       if( vm.count( "check" ) ) {
00327           std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
00328           double r, az, azend2;
00329           SPML::Geodesy::GEOtoRAD( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00330               settings.Input[0], settings.Input[1], lat, lon, r, az, azend2 );
00331           std::string result2 = "R[" + outrange + "] Az[" + outangle + "] AzEnd[" + outangle + "]:\n" +
00332               to_string_with_precision( r, settings.Precision ) + " " +
00333               to_string_with_precision( az, settings.Precision ) + " " +
00334               to_string_with_precision( azend2, settings.Precision );
00335           std::cout « result2 « std::endl;
00336           std::cout « "\nDelta:" « std::endl;
00337           std::string resultDelta = "R[" + outrange + "] Az[" + outangle + "] AzEnd[" + outangle + "]:\n" +
00338               to_string_with_precision( settings.Input[2] - r, settings.Precision ) + " " +
00339               to_string_with_precision( settings.Input[3] - az, settings.Precision ) + " " +
00340               to_string_with_precision( azend - azend2, settings.Precision );
00341           std::cout « resultDelta « std::endl;
00342       }
00343       return EXIT_SUCCESS;
00344   }
00345   //-----------------------------------------------------------------------------------------------------
00346   if( vm.count( "geo2ecef" ) ) {
00347       settings.Input = vm["geo2ecef"].as<std::vector<double»();
00348       if( settings.Input.size() != 3 ) {
00349           std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00350           return EXIT_FAILURE;
00351       }
00352
00353       double x, y, z;
00354       SPML::Geodesy::GEOtoECEF( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00355           settings.Input[0], settings.Input[1], settings.Input[2], x, y, z );
00356
00357       std::string result = "X[" + outangle + "] Y[" + outangle + "] Z[" + outangle + "]:\n" +
00358           to_string_with_precision( x, settings.Precision ) + " " +
00359           to_string_with_precision( y, settings.Precision ) + " " +
00360           to_string_with_precision( z, settings.Precision );
00361       std::cout « result « std::endl;
00362
00363       if( vm.count( "check" ) ) {
00364           std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
00365           double lat, lon, h;
00366           SPML::Geodesy::ECEFtoGEO( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00367               x, y, z, lat, lon, h );
00368           std::string result2 = "Lat[" + outangle + "] Lon[" + outangle + "] Height[" + outrange + "]:\n" +
00369               to_string_with_precision( lat, settings.Precision ) + " " +
00370               to_string_with_precision( lon, settings.Precision ) + " " +
```

```
00371             to_string_with_precision( h, settings.Precision );
00372         std::cout « result2 « std::endl;
00373         std::cout « "\nDelta:" « std::endl;
00374         std::string resultDelta = "Lat[" + outangle + "] Lon[" + outangle + "] Height[" + outrange + "]:\n" +
00375             to_string_with_precision( settings.Input[0] - lat, settings.Precision ) + " " +
00376             to_string_with_precision( settings.Input[1] - lon, settings.Precision ) + " " +
00377             to_string_with_precision( settings.Input[2] - h, settings.Precision );
00378         std::cout « resultDelta « std::endl;
00379     }
00380 }
00381 //-------------------------------------------------------------------------------------------------
00382 if( vm.count( "ecef2geo" ) ) {
00383     settings.Input = vm["ecef2geo"].as<std::vector<double»();
00384     if( settings.Input.size() != 3 ) {
00385         std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00386         return EXIT_FAILURE;
00387     }
00388
00389     double lat, lon, h;
00390     SPML::Geodesy::ECEFtoGEO( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00391         settings.Input[0], settings.Input[1], settings.Input[2], lat, lon, h );
00392     std::string result = "Lat[" + outangle + "] Lon[" + outangle + "] Height[" + outrange + "]:\n" +
00393         to_string_with_precision( lat, settings.Precision ) + " " +
00394         to_string_with_precision( lon, settings.Precision ) + " " +
00395         to_string_with_precision( h, settings.Precision );
00396     std::cout « result « std::endl;
00397
00398     if( vm.count( "check" ) ) {
00399         std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
00400         double x, y, z;
00401         SPML::Geodesy::GEOtoECEF( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00402             lat, lon, h, x, y, z );
00403         std::string result2 = "X[" + outrange + "] Y[" + outrange + "] Z[" + outrange + "]:\n" +
00404             to_string_with_precision( x, settings.Precision ) + " " +
00405             to_string_with_precision( y, settings.Precision ) + " " +
00406             to_string_with_precision( z, settings.Precision );
00407         std::cout « result2 « std::endl;
00408         std::cout « "\nDelta:" « std::endl;
00409         std::string resultDelta = "X[" + outrange + "] Y[" + outrange + "] Z[" + outrange + "]:\n" +
00410             to_string_with_precision( settings.Input[0] - x, settings.Precision ) + " " +
00411             to_string_with_precision( settings.Input[1] - y, settings.Precision ) + " " +
00412             to_string_with_precision( settings.Input[2] - z, settings.Precision );
00413         std::cout « resultDelta « std::endl;
00414     }
00415 }
00416 //-------------------------------------------------------------------------------------------------
00417 if( vm.count( "ecefdist" ) ) {
00418     settings.Input = vm["ecefdist"].as<std::vector<double»();
00419     if( settings.Input.size() != 6 ) {
00420         std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00421         return EXIT_FAILURE;
00422     }
00423
00424     double d = SPML::Geodesy::XYZtoDistance( settings.Input[0], settings.Input[1], settings.Input[2],
00425         settings.Input[3], settings.Input[4], settings.Input[5] );
00426     if( settings.RangeUnit == SPML::Units::RU_Kilometer ) {
00427         d *= 0.001;
00428     }
00429     std::string result = "Distance[" + outrange + "]:\n" +
00430         to_string_with_precision( d, settings.Precision );
00431     std::cout « result « std::endl;
00432
00433     if( vm.count( "check" ) ) {
00434         std::cout « "\nNo check provided for this operation!" « std::endl;
00435     }
00436 }
00437 //-------------------------------------------------------------------------------------------------
00438 if( vm.count( "ecefoffset" ) ) {
00439     settings.Input = vm["ecefoffset"].as<std::vector<double»();
00440     if( settings.Input.size() != 6 ) {
00441         std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00442         return EXIT_FAILURE;
00443     }
00444
00445     double dx, dy, dz;
00446     SPML::Geodesy::ECEF_offset( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00447         settings.Input[0], settings.Input[1], settings.Input[2],
00448         settings.Input[3], settings.Input[4], settings.Input[5], dx, dy, dz );
00449     if( settings.RangeUnit == SPML::Units::RU_Kilometer ) {
00450         dx *= 0.001;
00451         dy *= 0.001;
00452         dz *= 0.001;
00453     }
00454     std::string result = "dX[" + outrange + "] dY[" + outrange + "] dZ[" + outrange + "]:\n" +
00455         to_string_with_precision( dx, settings.Precision ) + " " +
00456         to_string_with_precision( dy, settings.Precision ) + " " +
00457         to_string_with_precision( dz, settings.Precision );
```

```
00458       std::cout « result « std::endl;
00459
00460       if( vm.count( "check" ) ) {
00461           std::cout « "\nNo check provided for this operation!" « std::endl;
00462       }
00463   }
00464   //-----------------------------------------------------------------------------------------------------
00465   if( vm.count( "ecef2enu" ) ) {
00466       settings.Input = vm["ecef2enu"].as<std::vector<double»();
00467       if( settings.Input.size() != 6 ) {
00468           std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00469           return EXIT_FAILURE;
00470       }
00471
00472       double e, n, u;
00473       SPML::Geodesy::ECEFtoENU( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00474           settings.Input[0], settings.Input[1], settings.Input[2], settings.Input[3], settings.Input[4], settings.Input[5],
00475           e, n, u );
00476       std::string result = "East[" + outrange + "] North[" + outrange + "] Up[" + outrange + "]:\n" +
00477           to_string_with_precision( e, settings.Precision ) + " " +
00478           to_string_with_precision( n, settings.Precision ) + " " +
00479           to_string_with_precision( u, settings.Precision );
00480       std::cout « result « std::endl;
00481
00482       if( vm.count( "check" ) ) {
00483           std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
00484           double x, y, z;
00485           SPML::Geodesy::ENUtoECEF( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00486               e, n, u, settings.Input[3], settings.Input[4], settings.Input[5], x, y, z );
00487           std::string result2 = "X[" + outrange + "] Y[" + outrange + "] Z[" + outrange + "]:\n" +
00488               to_string_with_precision( x, settings.Precision ) + " " +
00489               to_string_with_precision( y, settings.Precision ) + " " +
00490               to_string_with_precision( z, settings.Precision );
00491           std::cout « result2 « std::endl;
00492           std::cout « "\nDelta:" « std::endl;
00493           std::string resultDelta = "X[" + outrange + "] Y[" + outrange + "] Z[" + outrange + "]:\n" +
00494               to_string_with_precision( settings.Input[0] - x, settings.Precision ) + " " +
00495               to_string_with_precision( settings.Input[1] - y, settings.Precision ) + " " +
00496               to_string_with_precision( settings.Input[2] - z, settings.Precision );
00497           std::cout « resultDelta « std::endl;
00498       }
00499   }
00500   //-----------------------------------------------------------------------------------------------------
00501   if( vm.count( "enu2ecef" ) ) {
00502       settings.Input = vm["enu2ecef"].as<std::vector<double»();
00503       if( settings.Input.size() != 6 ) {
00504           std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00505           return EXIT_FAILURE;
00506       }
00507
00508       double x, y, z;
00509       SPML::Geodesy::ENUtoECEF( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00510           settings.Input[0], settings.Input[1], settings.Input[2],
00511           settings.Input[3], settings.Input[4], settings.Input[5], x, y, z );
00512       std::string result = "X[" + outrange + "] Y[" + outrange + "] Z[" + outrange + "]:\n" +
00513           to_string_with_precision( x, settings.Precision ) + " " +
00514           to_string_with_precision( y, settings.Precision ) + " " +
00515           to_string_with_precision( z, settings.Precision );
00516       std::cout « result « std::endl;
00517
00518       if( vm.count( "check" ) ) {
00519           std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
00520           double e, n, u;
00521           SPML::Geodesy::ECEFtoENU( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00522               x, y, z, settings.Input[3], settings.Input[4], settings.Input[5], e, n, u );
00523           std::string result2 = "East[" + outrange + "] North[" + outrange + "] Up[" + outrange + "]:\n" +
00524               to_string_with_precision( e, settings.Precision ) + " " +
00525               to_string_with_precision( n, settings.Precision ) + " " +
00526               to_string_with_precision( u, settings.Precision );
00527           std::cout « result2 « std::endl;
00528           std::cout « "\nDelta:" « std::endl;
00529           std::string resultDelta = "East[" + outrange + "] North[" + outrange + "] Up[" + outrange + "]:\n" +
00530               to_string_with_precision( settings.Input[0] - e, settings.Precision ) + " " +
00531               to_string_with_precision( settings.Input[1] - n, settings.Precision ) + " " +
00532               to_string_with_precision( settings.Input[2] - u, settings.Precision );
00533           std::cout « resultDelta « std::endl;
00534       }
00535   }
00536   //-----------------------------------------------------------------------------------------------------
00537   if( vm.count( "enu2aer" ) ) {
00538       settings.Input = vm["enu2aer"].as<std::vector<double»();
00539       if( settings.Input.size() != 3 ) {
00540           std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00541           return EXIT_FAILURE;
00542       }
00543
00544       double a, e, r;
```

```
00545       SPML::Geodesy::ENUtoAER( settings.RangeUnit, settings.AngleUnit,
00546           settings.Input[0], settings.Input[1], settings.Input[2], a, e, r );
00547       std::string result = "Azimuth[" + outangle + "] Elevation[" + outangle + "] slantRange[" + outrange + "]:\n" +
00548           to_string_with_precision( a, settings.Precision ) + " " +
00549           to_string_with_precision( e, settings.Precision ) + " " +
00550           to_string_with_precision( r, settings.Precision );
00551       std::cout « result « std::endl;
00552
00553       if( vm.count( "check" ) ) {
00554           std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
00555           double e_, n_, u_;
00556           SPML::Geodesy::AERtoENU( settings.RangeUnit, settings.AngleUnit, a, e, r, e_, n_, u_ );
00557           std::string result2 = "East[" + outrange + "] North[" + outrange + "] Up[" + outrange + "]:\n" +
00558               to_string_with_precision( e_, settings.Precision ) + " " +
00559               to_string_with_precision( n_, settings.Precision ) + " " +
00560               to_string_with_precision( u_, settings.Precision );
00561           std::cout « result2 « std::endl;
00562           std::cout « "\nDelta:" « std::endl;
00563           std::string resultDelta = "East[" + outrange + "] North[" + outrange + "] Up[" + outrange + "]:\n" +
00564               to_string_with_precision( settings.Input[0] - e_, settings.Precision ) + " " +
00565               to_string_with_precision( settings.Input[1] - n_, settings.Precision ) + " " +
00566               to_string_with_precision( settings.Input[2] - u_, settings.Precision );
00567           std::cout « resultDelta « std::endl;
00568       }
00569   }
00570   //-------------------------------------------------------------------------------------------------------
00571   if( vm.count( "aer2enu" ) ) {
00572       settings.Input = vm["aer2enu"].as<std::vector<double»();
00573       if( settings.Input.size() != 3 ) {
00574           std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00575           return EXIT_FAILURE;
00576       }
00577
00578       double e, n, u;
00579       SPML::Geodesy::AERtoENU( settings.RangeUnit, settings.AngleUnit,
00580           settings.Input[0], settings.Input[1], settings.Input[2], e, n, u );
00581       std::string result = "East[" + outrange + "] North[" + outrange + "] Up[" + outrange + "]:\n" +
00582           to_string_with_precision( e, settings.Precision ) + " " +
00583           to_string_with_precision( n, settings.Precision ) + " " +
00584           to_string_with_precision( u, settings.Precision );
00585       std::cout « result « std::endl;
00586
00587       if( vm.count( "check" ) ) {
00588           std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
00589           double a_, e_, r_;
00590           SPML::Geodesy::ENUtoAER( settings.RangeUnit, settings.AngleUnit, e, n, u, a_, e_, r_ );
00591           std::string result2 = "Azimuth[" + outangle + "] Elevation[" + outangle + "] slantRange[" + outrange + "]:\n"
    +
00592               to_string_with_precision( a_, settings.Precision ) + " " +
00593               to_string_with_precision( e_, settings.Precision ) + " " +
00594               to_string_with_precision( r_, settings.Precision );
00595           std::cout « result2 « std::endl;
00596           std::cout « "\nDelta:" « std::endl;
00597           std::string resultDelta = "Azimuth[" + outangle + "] Elevation[" + outangle + "] slantRange[" + outrange +
    "]:\n" +
00598               to_string_with_precision( settings.Input[0] - a_, settings.Precision ) + " " +
00599               to_string_with_precision( settings.Input[1] - e_, settings.Precision ) + " " +
00600               to_string_with_precision( settings.Input[2] - r_, settings.Precision );
00601           std::cout « resultDelta « std::endl;
00602       }
00603   }
00604   //-------------------------------------------------------------------------------------------------------
00605   if( vm.count( "geo2enu" ) ) {
00606       settings.Input = vm["geo2enu"].as<std::vector<double»();
00607       if( settings.Input.size() != 6 ) {
00608           std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00609           return EXIT_FAILURE;
00610       }
00611
00612       double e, n, u;
00613       SPML::Geodesy::GEOtoENU( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00614           settings.Input[0], settings.Input[1], settings.Input[2],
00615           settings.Input[3], settings.Input[4], settings.Input[5], e, n, u );
00616       std::string result = "East[" + outrange + "] North[" + outrange + "] Up[" + outrange + "]:\n" +
00617           to_string_with_precision( e, settings.Precision ) + " " +
00618           to_string_with_precision( n, settings.Precision ) + " " +
00619           to_string_with_precision( u, settings.Precision );
00620       std::cout « result « std::endl;
00621
00622       if( vm.count( "check" ) ) {
00623           std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
00624           double lat, lon, h;
00625           SPML::Geodesy::ENUtoGEO( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00626               e, n, u, settings.Input[3], settings.Input[4], settings.Input[5], lat, lon, h );
00627           std::string result2 = "Lat[" + outangle + "] Lon[" + outangle + "] Height[" + outrange + "]:\n" +
00628               to_string_with_precision( lat, settings.Precision ) + " " +
00629               to_string_with_precision( lon, settings.Precision ) + " " +
```

```
00630             to_string_with_precision( h, settings.Precision );
00631         std::cout « result2 « std::endl;
00632         std::cout « "\nDelta:" « std::endl;
00633         std::string resultDelta = "Lat[" + outangle + "] Lon[" + outangle + "] Height[" + outrange + "]:\n" +
00634             to_string_with_precision( settings.Input[0] - lat, settings.Precision ) + " " +
00635             to_string_with_precision( settings.Input[1] - lon, settings.Precision ) + " " +
00636             to_string_with_precision( settings.Input[2] - h, settings.Precision );
00637         std::cout « resultDelta « std::endl;
00638     }
00639 }
00640 //-----------------------------------------------------------------------------------------------------------
00641 if( vm.count( "enu2geo" ) ) {
00642     settings.Input = vm["enu2geo"].as<std::vector<double»();
00643     if( settings.Input.size() != 6 ) {
00644         std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00645         return EXIT_FAILURE;
00646     }
00647
00648     double lat, lon, h;
00649     SPML::Geodesy::ENUtoGEO( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00650         settings.Input[0], settings.Input[1], settings.Input[2],
00651         settings.Input[3], settings.Input[4], settings.Input[5], lat, lon, h );
00652     std::string result = "Lat[" + outangle + "] Lon[" + outangle + "] Height[" + outrange + "]:\n" +
00653         to_string_with_precision( lat, settings.Precision ) + " " +
00654         to_string_with_precision( lon, settings.Precision ) + " " +
00655         to_string_with_precision( h, settings.Precision );
00656     std::cout « result « std::endl;
00657
00658     if( vm.count( "check" ) ) {
00659         std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
00660         double e, n, u;
00661         SPML::Geodesy::GEOtoENU( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00662             lat, lon, h, settings.Input[3], settings.Input[4], settings.Input[5], e, n, u );
00663         std::string result2 = "East[" + outrange + "] North[" + outrange + "] Up[" + outrange + "]:\n" +
00664             to_string_with_precision( e, settings.Precision ) + " " +
00665             to_string_with_precision( n, settings.Precision ) + " " +
00666             to_string_with_precision( u, settings.Precision );
00667         std::cout « result2 « std::endl;
00668         std::cout « "\nDelta:" « std::endl;
00669         std::string resultDelta = "East[" + outrange + "] North[" + outrange + "] Up[" + outrange + "]:\n" +
00670             to_string_with_precision( settings.Input[0] - e, settings.Precision ) + " " +
00671             to_string_with_precision( settings.Input[1] - n, settings.Precision ) + " " +
00672             to_string_with_precision( settings.Input[2] - u, settings.Precision );
00673         std::cout « resultDelta « std::endl;
00674     }
00675 }
00676 //-----------------------------------------------------------------------------------------------------------
00677 if( vm.count( "geo2aer" ) ) {
00678     settings.Input = vm["geo2aer"].as<std::vector<double»();
00679     if( settings.Input.size() != 6 ) {
00680         std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00681         return EXIT_FAILURE;
00682     }
00683
00684     double a, e, r;
00685     SPML::Geodesy::GEOtoAER( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00686         settings.Input[0], settings.Input[1], settings.Input[2],
00687         settings.Input[3], settings.Input[4], settings.Input[5], a, e, r );
00688     std::string result = "Azimuth[" + outangle + "] Elevation[" + outangle + "] slantRange[" + outrange + "]:\n" +
00689         to_string_with_precision( a, settings.Precision ) + " " +
00690         to_string_with_precision( e, settings.Precision ) + " " +
00691         to_string_with_precision( r, settings.Precision );
00692     std::cout « result « std::endl;
00693
00694     if( vm.count( "check" ) ) {
00695         std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
00696         double lat, lon, h;
00697         SPML::Geodesy::AERtoGEO( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00698             a, e, r, settings.Input[3], settings.Input[4], settings.Input[5], lat, lon, h );
00699         std::string result2 = "Lat[" + outangle + "] Lon[" + outangle + "] Height[" + outrange + "]:\n" +
00700             to_string_with_precision( lat, settings.Precision ) + " " +
00701             to_string_with_precision( lon, settings.Precision ) + " " +
00702             to_string_with_precision( h, settings.Precision );
00703         std::cout « result2 « std::endl;
00704         std::cout « "\nDelta:" « std::endl;
00705         std::string resultDelta = "Lat[" + outangle + "] Lon[" + outangle + "] Height[" + outrange + "]:\n" +
00706             to_string_with_precision( settings.Input[0] - lat, settings.Precision ) + " " +
00707             to_string_with_precision( settings.Input[1] - lon, settings.Precision ) + " " +
00708             to_string_with_precision( settings.Input[2] - h, settings.Precision );
00709         std::cout « resultDelta « std::endl;
00710     }
00711 }
00712 //-----------------------------------------------------------------------------------------------------------
00713 if( vm.count( "aer2geo" ) ) {
00714     settings.Input = vm["aer2geo"].as<std::vector<double»();
00715     if( settings.Input.size() != 6 ) {
00716         std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
```

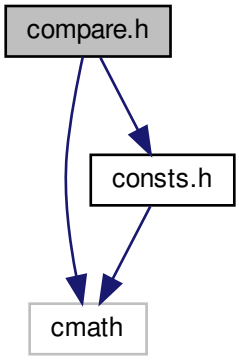
```
00717           return EXIT_FAILURE;
00718       }
00719
00720       double lat, lon, h;
00721       SPML::Geodesy::ENUtoGEO( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00722           settings.Input[0], settings.Input[1], settings.Input[2],
00723           settings.Input[3], settings.Input[4], settings.Input[5], lat, lon, h );
00724       std::string result = "Lat[" + outangle + "] Lon[" + outangle + "] Height[" + outrange + "]:\n" +
00725           to_string_with_precision( lat, settings.Precision ) + " " +
00726           to_string_with_precision( lon, settings.Precision ) + " " +
00727           to_string_with_precision( h, settings.Precision );
00728       std::cout « result « std::endl;
00729
00730       if( vm.count( "check" ) ) {
00731           std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
00732           double e, n, u;
00733           SPML::Geodesy::GEOtoENU( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00734               lat, lon, h, settings.Input[3], settings.Input[4], settings.Input[5], e, n, u );
00735           std::string result2 = "East[" + outrange + "] North[" + outrange + "] Up[" + outrange + "]:\n" +
00736               to_string_with_precision( e, settings.Precision ) + " " +
00737               to_string_with_precision( n, settings.Precision ) + " " +
00738               to_string_with_precision( u, settings.Precision );
00739           std::cout « result2 « std::endl;
00740           std::cout « "\nDelta:" « std::endl;
00741           std::string resultDelta = "East[" + outrange + "] North[" + outrange + "] Up[" + outrange + "]:\n" +
00742               to_string_with_precision( settings.Input[0] - e, settings.Precision ) + " " +
00743               to_string_with_precision( settings.Input[1] - n, settings.Precision ) + " " +
00744               to_string_with_precision( settings.Input[2] - u, settings.Precision );
00745           std::cout « resultDelta « std::endl;
00746       }
00747   }
00748   //-------------------------------------------------------------------------------------------------------
00749   if( vm.count( "ecef2aer" ) ) {
00750       settings.Input = vm["ecef2aer"].as<std::vector<double»();
00751       if( settings.Input.size() != 6 ) {
00752           std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00753           return EXIT_FAILURE;
00754       }
00755
00756       double a, e, r;
00757       SPML::Geodesy::ECEFtoAER( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00758           settings.Input[0], settings.Input[1], settings.Input[2], settings.Input[3], settings.Input[4], settings.Input[5],
00759           a, e, r );
00760       std::string result = "Azimuth[" + outangle + "] Elevation[" + outangle + "] slantRange[" + outrange + "]:\n" +
00761           to_string_with_precision( a, settings.Precision ) + " " +
00762           to_string_with_precision( e, settings.Precision ) + " " +
00763           to_string_with_precision( r, settings.Precision );
00764       std::cout « result « std::endl;
00765
00766       if( vm.count( "check" ) ) {
00767           std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
00768           double x, y, z;
00769           SPML::Geodesy::AERtoECEF( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00770               a, e, r, settings.Input[3], settings.Input[4], settings.Input[5], x, y, z );
00771           std::string result2 = "X[" + outrange + "] Y[" + outrange + "] Z[" + outrange + "]:\n" +
00772               to_string_with_precision( x, settings.Precision ) + " " +
00773               to_string_with_precision( y, settings.Precision ) + " " +
00774               to_string_with_precision( z, settings.Precision );
00775           std::cout « result2 « std::endl;
00776           std::cout « "\nDelta:" « std::endl;
00777           std::string resultDelta = "X[" + outrange + "] Y[" + outrange + "] Z[" + outrange + "]:\n" +
00778               to_string_with_precision( settings.Input[0] - x, settings.Precision ) + " " +
00779               to_string_with_precision( settings.Input[1] - y, settings.Precision ) + " " +
00780               to_string_with_precision( settings.Input[2] - z, settings.Precision );
00781           std::cout « resultDelta « std::endl;
00782       }
00783   }
00784   //-------------------------------------------------------------------------------------------------------
00785   if( vm.count( "aer2ecef" ) ) {
00786       settings.Input = vm["aer2ecef"].as<std::vector<double»();
00787       if( settings.Input.size() != 6 ) {
00788           std::cout « "Неверный ввод, смотри --help/Wrong input, read --help" « std::endl;
00789           return EXIT_FAILURE;
00790       }
00791
00792       double x, y, z;
00793       SPML::Geodesy::AERtoECEF( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00794           settings.Input[0], settings.Input[1], settings.Input[2],
00795           settings.Input[3], settings.Input[4], settings.Input[5], x, y, z );
00796       std::string result = "X[" + outrange + "] Y[" + outrange + "] Z[" + outrange + "]:\n" +
00797           to_string_with_precision( x, settings.Precision ) + " " +
00798           to_string_with_precision( y, settings.Precision ) + " " +
00799           to_string_with_precision( z, settings.Precision );
00800       std::cout « result « std::endl;
00801
00802       if( vm.count( "check" ) ) {
00803           std::cout « "\nCheck by solving inverse task and calc delta:" « std::endl;
```

```
00804           double a, e, r;
00805           SPML::Geodesy::ECEFtoAER( ellipsoids.at( settings.EllipsoidNumber ), settings.RangeUnit, settings.AngleUnit,
00806               x, y, z, settings.Input[3], settings.Input[4], settings.Input[5], a, e, r );
00807           std::string result2 = "Azimuth[" + outangle + "] Elevation[" + outangle + "] slantRange[" + outrange + "]:\n"
    +
00808               to_string_with_precision( a, settings.Precision ) + " " +
00809               to_string_with_precision( e, settings.Precision ) + " " +
00810               to_string_with_precision( r, settings.Precision );
00811           std::cout « result2 « std::endl;
00812           std::cout « "\nDelta:" « std::endl;
00813           std::string resultDelta = "Azimuth[" + outangle + "] Elevation[" + outangle + "] slantRange[" + outrange +
    "]:\n" +
00814               to_string_with_precision( settings.Input[0] - a, settings.Precision ) + " " +
00815               to_string_with_precision( settings.Input[1] - e, settings.Precision ) + " " +
00816               to_string_with_precision( settings.Input[2] - r, settings.Precision );
00817           std::cout « resultDelta « std::endl;
00818       }
00819   }
00820   //-------------------------------------------------------------------------------------------------------
00821   return EXIT_SUCCESS;
00822 }// end main
```

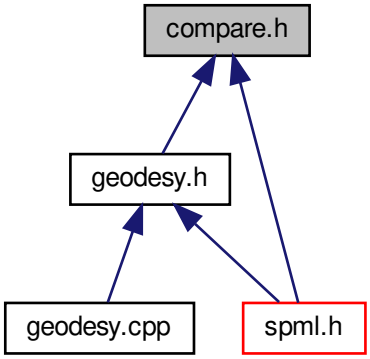
10.3 Файл compare.h

Функции сравнения чисел, массивов

#include <cmath>
#include <consts.h>
Граф включаемых заголовочных файлов для compare.h:



Граф файлов, в которые включается этот файл:



Пространства имен

- namespace SPML

  Специальная библиотека программных модулей (СБ ПМ)
- namespace SPML::Compare

  Сравнение чисел

Функции

- bool SPML::Compare::AreEqualAbs (float first, float second, const float &eps=EPS_F)

  Сравнение двух действительных чисел (по абсолютной разнице)
- bool SPML::Compare::AreEqualAbs (double first, double second, const double &eps=EPS_D)

  Сравнение двух действительных чисел (по абсолютной разнице)
- bool SPML::Compare::AreEqualRel (float first, float second, const float &eps=EPS_REL)

  Сравнение двух действительных чисел (по относительной разнице)
- bool SPML::Compare::AreEqualRel (double first, double second, const double &eps=EPS_REL)

  Сравнение двух действительных чисел (по относительной разнице)
- bool SPML::Compare::IsZeroAbs (float value, const float &eps=EPS_F)

  Проверка действительного числа на равенство нулю (по абсолютной разнице)
- bool SPML::Compare::IsZeroAbs (double value, const double &eps=EPS_D)

  Проверка действительного числа на равенство нулю (по абсолютной разнице)

Переменные

- static const float SPML::Compare::EPS_F = 1.0e-4f

  Абсолютная точность по умолчанию при сравнениях чисел типа float (1.0e-4)
- static const double SPML::Compare::EPS_D = 1.0e-8

  Абсолютная точность по умолчанию при сравнениях чисел типа double (1.0e-8)
- static const float SPML::Compare::EPS_REL = 0.01

  Относительная точность по умолчанию

10.3.1 Подробное описание

Функции сравнения чисел, массивов

Дата

27.07.20 - создан

Автор

Соболев А.А.

См. определение в файле compare.h

10.4 compare.h

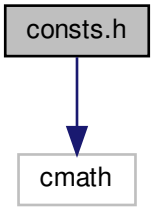
См. документацию.

```
00001 //-----------------------------------------------------------------------------------------------------------
00010
00011 #ifndef SPML_COMPARE_H
00012 #define SPML_COMPARE_H
00013
00014 // System includes:
00015 #include <cmath>
00016
00017 // SPML includes:
00018 #include <consts.h>
00019
00020 namespace SPML
00021 {
00022 namespace Compare
00023 {
00024 //-----------------------------------------------------------------------------------------------------------
00025 static const float EPS_F = 1.0e-4f;
00026 static const double EPS_D = 1.0e-8;
00027 static const float EPS_REL = 0.01;
00028
00029 //-----------------------------------------------------------------------------------------------------------
00038 inline bool AreEqualAbs( float first, float second, const float &eps = EPS_F )
00039 {
00040     return ( std::abs( first - second ) <= eps );
00041 }
00042
00051 inline bool AreEqualAbs( double first, double second, const double &eps = EPS_D )
00052 {
00053     return ( std::abs( first - second ) <= eps );
00054 }
00055
00056 //-----------------------------------------------------------------------------------------------------------
00065 inline bool AreEqualRel( float first, float second, const float &eps = EPS_REL )
00066 {
00067     return ( ( std::abs( first - second ) <= ( eps * std::abs( first ) ) ) &&
00068         ( std::abs( first - second ) <= ( eps * std::abs( second ) ) ) );
00069 }
00070
00079 inline bool AreEqualRel( double first, double second, const double &eps = EPS_REL )
00080 {
00081     return ( ( std::abs( first - second ) <= ( eps * std::abs( first ) ) ) &&
00082         ( std::abs( first - second ) <= ( eps * std::abs( second ) ) ) );
00083 }
00084
00085 //-----------------------------------------------------------------------------------------------------------
00093 inline bool IsZeroAbs( float value, const float &eps = EPS_F )
00094 {
00095     return ( std::abs( value ) <= eps );
00096 }
00097
00105 inline bool IsZeroAbs( double value, const double &eps = EPS_D )
00106 {
00107     return ( std::abs( value ) <= eps );
00108 }
00109
00110 } // end namespace Compare
00111 } // end namespace SPML
00112 #endif // SPML_COMPARE_H
```

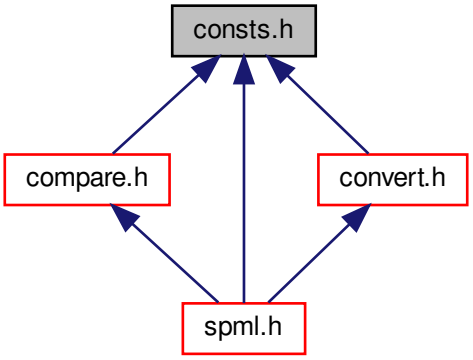
10.5 Файл consts.h

Константы библиотеки СБПМ

#include <cmath>
Граф включаемых заголовочных файлов для consts.h:



Граф файлов, в которые включается этот файл:



Пространства имен

- namespace SPML

  Специальная библиотека программных модулей (СБ ПМ)
- namespace SPML::Consts

  Константы

Переменные

- const float SPML::Consts::C_F = 3.0e8f

  Скорость света, [м/с] в одинарной точности (float)
- const double SPML::Consts::C_D = 3.0e8

  Скорость света, [м/с] в двойной точности (double)
- const double SPML::Consts::PI_D = std::acos(-1.0)

  Число PI = 3.14... в радианах в двойной точности (double)
- const float SPML::Consts::PI_F = static_cast<float>(std::acos(-1.0))

  Число PI = 3.14... в радианах в одинарной точности (float)
- const double SPML::Consts::PI_2_D = 2.0 * std::acos(-1.0)

  Число 2*PI = 6.28... в радианах в двойной точности (double)
- const float SPML::Consts::PI_2_F = static_cast<float>(2.0 * std::acos(-1.0))

  Число 2*PI = 6.28... в радианах в одинарной точности (float)
- const double SPML::Consts::PI_05_D = std::acos(0.0)

  Число PI/2 = 1.57... в радианах в двойной точности (double)
- const float SPML::Consts::PI_05_F = static_cast<float>(std::acos(0.0))

  Число PI/2 = 1.57... в радианах в одинарной точности (float)
- const double SPML::Consts::PI_025_D = std::acos(-1.0) * 0.25

  Число PI/4 = 0.785... в радианах в двойной точности (double)
- const float SPML::Consts::PI_025_F = static_cast<float>(std::acos(-1.0) * 0.25)

  Число PI/4 = 0.785... в радианах в одинарной точности (float)

10.5.1 Подробное описание

Константы библиотеки СБПМ

Дата

27.07.20 - создан

Автор

Соболев А.А.

См. определение в файле consts.h

10.6 consts.h

См. документацию.

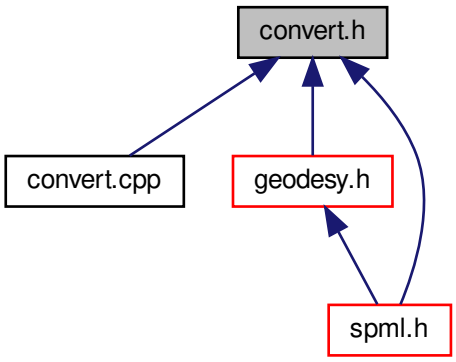
```
00001 //-----------------------------------------------------------------------------------------------------------
00010
00011 #ifndef SPML_CONSTS_H
00012 #define SPML_CONSTS_H
00013
00014 // System includes:
00015 #include <cmath>
00016
00017 namespace SPML
00018 {
00019 namespace Consts
00020 {
00021 //-----------------------------------------------------------------------------------------------------------
00022 // Скорость света
```

```
00023 const float C_F = 3.0e8f;
00024 const double C_D = 3.0e8;
00025
00026 //-------------------------------------------------------------------------------------------------------------
00027 // Число ПИ и его части
00028 const double PI_D = std::acos( -1.0 );
00029 const float PI_F = static_cast<float>( std::acos( -1.0 ) );
00030
00031 const double PI_2_D = 2.0 * std::acos( -1.0 );
00032 const float PI_2_F = static_cast<float>( 2.0 * std::acos( -1.0 ) );
00033
00034 const double PI_05_D = std::acos( 0.0 );
00035 const float PI_05_F = static_cast<float>( std::acos( 0.0 ) );
00036
00037 const double PI_025_D = std::acos( -1.0 ) * 0.25;
00038 const float PI_025_F = static_cast<float>( std::acos( -1.0 ) * 0.25 );
00039
00040 } // end namespace Consts
00041 } // end namespace SPML
00042 #endif // SPML_CONSTS_H
```

10.7 Файл convert.h

Переводы единиц библиотеки СБПМ

#include <cmath>
#include <ctime>
#include <string>
#include <cassert>
#include <type_traits>
#include <consts.h>
#include <units.h>
Граф файлов, в которые включается этот файл:



Пространства имен

- namespace SPML

  Специальная библиотека программных модулей (СБ ПМ)
- namespace SPML::Convert

  Переводы единиц

Функции

- float SPML::Convert::AngleTo360 (float angle, const Units::TAngleUnit &au)

  Приведение угла в [0,360) градусов или [0,2PI) радиан
- double SPML::Convert::AngleTo360 (double angle, const Units::TAngleUnit &au)

  Приведение угла в [0,360) градусов или [0,2PI) радиан
- float SPML::Convert::EpsToMP90 (float angle, const Units::TAngleUnit &au)

  Приведение угла места в [-90,90] градусов или [-PI/2, PI/2] радиан
- double SPML::Convert::EpsToMP90 (double angle, const Units::TAngleUnit &au)

  Приведение угла места в [-90,90] градусов или [-PI/2, PI/2] радиан
- template<class T >
  T SPML::Convert::AbsAzToRelAz (T absAz, T origin, const Units::TAngleUnit &au)

  Перевод абсолютного азимута относительно севера в азимут относительно указанного направления
- template<class T >
  T SPML::Convert::RelAzToAbsAz (T relAz, T origin, const Units::TAngleUnit &au)

  Перевод относительного азимута в абсолютный азимут относительно севера
- template<class T >
  T SPML::Convert::dBtoTimesByP (T dB)

  Перевод [дБ] в разы по мощности
- template<class T >
  T SPML::Convert::dBtoTimesByU (T dB)

  Перевод [дБ] в разы по напряжению
- void SPML::Convert::UnixTimeToHourMinSec (int rawtime, int &hour, int &min, int &sec, int &day=dummy_int, int &mon=dummy_int, int &year=dummy_int)

  Перевод целого числа секунд с 00:00:00 01.01.1970 в часы/минуты/секунды/день/месяц/год
- const std::string SPML::Convert::CurrentDateTimeToString ()

  Получение текущей даты и времени
- double SPML::Convert::CheckDeltaAngle (double deltaAngle, const SPML::Units::TAngleUnit &au)

  Проверка разницы в углах

Переменные

- const float SPML::Convert::DgToRdF = static_cast<float>(std::asin(1.0) / 90.0)

  Перевод градусов в радианы (float) путем умножения на данную константу
- const float SPML::Convert::RdToDgF = static_cast<float>(90.0 / std::asin(1.0))

  Перевод радианов в градусы (float) путем умножения на данную константу
- const double SPML::Convert::DgToRdD = std::asin(1.0) / 90.0

  Перевод градусов в радианы (double) путем умножения на данную константу
- const double SPML::Convert::RdToDgD = 90.0 / std::asin(1.0)

  Перевод радианов в градусы (double) путем умножения на данную константу
- const double SPML::Convert::MsToKmD_half = Consts::C_D * 0.5 * 1.0e-6

  Перевод задержки [мс] в дальность [км] путем умножения на данную константу (по формуле R = C * Tau / 2)
- const double SPML::Convert::KmToMsD_half = 1.0 / MsToKmD_half

  Перевод дальности [км] в задержку [мс] путем умножения на данную константу (по формуле Tau = 2 * R / C)
- const double SPML::Convert::McsToKmD_half = Consts::C_D * 0.5 * 1.0e-9

  Перевод задержки [мкс] в дальность [км] путем умножения на данную константу (по формуле R = C * Tau / 2)
- const double SPML::Convert::KmToMcsD_half = 1.0 / McsToKmD_half

Перевод дальности [км] в задержку [мкс] путем умножения на данную константу (по формуле Tau = 2 * R / C)

- const double SPML::Convert::MsToMetersD_full = Consts::C_D * 1.0e-3

  Перевод задержки [мс] в дальность [м] путем умножения на данную константу (по формуле R = C * Tau / 2)
- const double SPML::Convert::MetersToMsD_full = 1.0 / MsToMetersD_full

  Перевод дальности [м] в задержку [мс] путем умножения на данную константу (по формуле Tau = 2 * R / C)
- const double SPML::Convert::MsToKmD_full = Consts::C_D * 1.0e-6

  Перевод задержки [мс] в дальность [км] путем умножения на данную константу (по формуле R = C * Tau / 2)
- const double SPML::Convert::KmToMsD_full = 1.0 / MsToKmD_full

  Перевод дальности [км] в задержку [мс] путем умножения на данную константу (по формуле Tau = 2 * R / C)
- static int SPML::Convert::dummy_int

10.7.1 Подробное описание

Переводы единиц библиотеки СБПМ

Дата

27.07.20 - создан

Автор

Соболев А.А.

См. определение в файле convert.h

10.8 convert.h

См. документацию.

```
00001 //-----------------------------------------------------------------------------------------------------------
00010
00011 #ifndef SPML_CONVERT_H
00012 #define SPML_CONVERT_H
00013
00014 // System includes:
00015 #include <cmath>
00016 #include <ctime>
00017 #include <string>
00018 #include <cassert>
00019 #include <type_traits>
00020
00021 // SPML includes:
00022 #include <consts.h>
00023 #include <units.h>
00024
00025 namespace SPML
00026 {
00027 namespace Convert
00028 {
00029 //-----------------------------------------------------------------------------------------------------------
00030 // Константы перевода радианов в градусы и наоборот
00031 const float DgToRdF = static_cast<float>( std::asin( 1.0 ) / 90.0 );
00032 const float RdToDgF = static_cast<float>( 90.0 / std::asin( 1.0 ) );
00033 const double DgToRdD = std::asin( 1.0 ) / 90.0;
00034 const double RdToDgD = 90.0 / std::asin( 1.0 );
00035
00036 //-----------------------------------------------------------------------------------------------------------
```

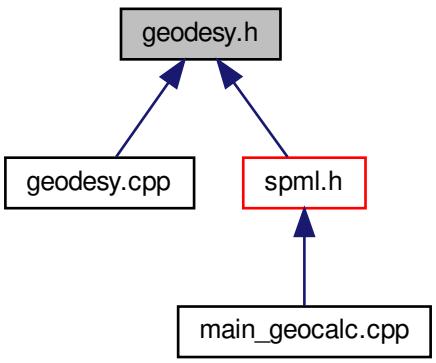
```
00037 // Перевод дальности в задержку и наоборот исходя из формулы R = C*tau/2 путем умножения на данную
      константу
00038 //const double MsToMetersD_half = Consts::C_D * 0.5 * 1.0e-3; ///< Перевод задержки [мс] в дальность [м] путем
      умножения на данную константу (по формуле R = C * Tau / 2 )
00039 //const double MetersToMsD_half = 1.0 / MsToMetersD_half; ///< Перевод дальности [м] в задержку [мс] путем
      умножения на данную константу (по формуле Tau = 2 * R / C )
00040
00041 const double MsToKmD_half = Consts::C_D * 0.5 * 1.0e-6;
00042 const double KmToMsD_half = 1.0 / MsToKmD_half;
00043
00044 //const double McsToMetersD_half = Consts::C_D * 0.5 * 1.0e-6; ///< Перевод задержки [мкс] в дальность [м] путем
      умножения на данную константу (по формуле R = C * Tau / 2 )
00045 //const double MetersToMcsD_half = 1.0 / McsToMetersD_half; ///< Перевод дальности [м] в задержку [мкс] путем
      умножения на данную константу (по формуле Tau = 2 * R / C )
00046
00047 const double McsToKmD_half = Consts::C_D * 0.5 * 1.0e-9;
00048 const double KmToMcsD_half = 1.0 / McsToKmD_half;
00049
00050 //const double SecToMetersD_half = Consts::C_D * 0.5; ///< Перевод задержки [с] в дальность [м] путем умножения
      на данную константу (по формуле R = C * Tau / 2 )
00051 //const double MetersToSecD_half = 1.0 / SecToMetersD_half; ///< Перевод дальности [м] в задержку [с] путем
      умножения на данную константу (по формуле Tau = 2 * R / C )
00052
00053 //const double SecToKmD_half = Consts::C_D * 0.5 * 1.0e-3; ///< Перевод задержки [с] в дальность [км] путем
      умножения на данную константу (по формуле R = C * Tau / 2 )
00054 //const double KmToSecD_half = 1.0 / SecToKmD_half; ///< Перевод дальности к[м] в задержку [с] путем
      умножения на данную константу (по формуле Tau = 2 * R / C )
00055
00058 const double MsToMetersD_full = Consts::C_D * 1.0e-3;
00059 const double MetersToMsD_full = 1.0 / MsToMetersD_full;
00060
00061 const double MsToKmD_full = Consts::C_D * 1.0e-6;
00062 const double KmToMsD_full = 1.0 / MsToKmD_full;
00063
00064 //const double McsToMetersD_full = Consts::C_D * 1.0e-6; ///< Перевод задержки [мкс] в дальность [м] путем
      умножения на данную константу (по формуле R = C * Tau / 2 )
00065 //const double MetersToMcsD_full = 1.0 / McsToMetersD_full; ///< Перевод дальности [м] в задержку [мкс] путем
      умножения на данную константу (по формуле Tau = 2 * R / C )
00066
00067 //const double McsToKmD_full = Consts::C_D * 1.0e-9; ///< Перевод задержки [мкс] в дальность [км] путем
      умножения на данную константу (по формуле R = C * Tau / 2 )
00068 //const double KmToMcsD_full = 1.0 / McsToKmD_full; ///< Перевод дальности [км] в задержку [мкс] путем
      умножения на данную константу (по формуле Tau = 2 * R / C )
00069
00070 //const double SecToMetersD_full = Consts::C_D; ///< Перевод задержки [с] в дальность [м] путем умножения на
      данную константу (по формуле R = C * Tau / 2 )
00071 //const double MetersToSecD_full = 1.0 / SecToMetersD_full; ///< Перевод дальности [м] в задержку [с] путем
      умножения на данную константу (по формуле Tau = 2 * R / C )
00072
00073 //const double SecToKmD_full = Consts::C_D * 1.0e-3; ///< Перевод задержки [с] в дальность [км] путем
      умножения на данную константу (по формуле R = C * Tau / 2 )
00074 //const double KmToSecD_full = 1.0 / SecToKmD_full; ///< Перевод дальности к[м] в задержку [с] путем
      умножения на данную константу (по формуле Tau = 2 * R / C )
00075
00076 //-----------------------------------------------------------------------------------------------------------
00083 float AngleTo360( float angle, const Units::TAngleUnit &au );
00084
00091 double AngleTo360( double angle, const Units::TAngleUnit &au );
00092
00093 //-----------------------------------------------------------------------------------------------------------
00100 float EpsToMP90( float angle, const Units::TAngleUnit &au );
00101
00108 double EpsToMP90( double angle, const Units::TAngleUnit &au );
00109
00110 //-----------------------------------------------------------------------------------------------------------
00118 template <class T>
00119 inline T AbsAzToRelAz( T absAz, T origin, const Units::TAngleUnit &au )
00120 {
00121     static_assert( std::is_same<T, float>::value || std::is_same<T, double>::value, "wrong template class!" );
00122     T result = AngleTo360( absAz, au ) - origin ;
00123     return result;
00124 }
00125
00134 template <class T>
00135 inline T RelAzToAbsAz( T relAz, T origin, const Units::TAngleUnit &au )
00136 {
00137     static_assert( ( std::is_same<T, float>::value ) || ( std::is_same<T, double>::value ), "wrong template class!" );
00138     T result = AngleTo360( ( relAz + origin ), au );
00139     return result;
00140 }
00141
00142 //-----------------------------------------------------------------------------------------------------------
00148 template <class T>
00149 inline T dBtoTimesByP( T dB )
00150 {
00151     static_assert( ( std::is_same<T, float>::value ) || ( std::is_same<T, double>::value ), "wrong template class!" );
00152     return ( std::pow( 10.0, ( dB * 0.1 ) ) ); // 10 ^ ( dB / 10 )
```

```
00153 }
00154
00160 template <class T>
00161 inline T dBtoTimesByU( T dB )
00162 {
00163     static_assert( ( std::is_same<T, float>::value ) || ( std::is_same<T, double>::value ), "wrong template class!" );
00164     return ( std::pow( 10.0, ( dB * 0.05 ) ) ); // 10 ^ ( dB / 20 )
00165 }
00166
00167 static int dummy_int;
00168 //-----------------------------------------------------------------------------------------------------------
00179 void UnixTimeToHourMinSec( int rawtime, int &hour, int &min, int &sec, int &day = dummy_int, int &mon =
       dummy_int, int &year = dummy_int );
00180
00181 //-----------------------------------------------------------------------------------------------------------
00186 const std::string CurrentDateTimeToString();
00187
00188 //-----------------------------------------------------------------------------------------------------------
00196 double CheckDeltaAngle( double deltaAngle, const SPML::Units::TAngleUnit &au );
00197
00198 } // end namespace Convert
00199 } // end namespace SPML
00200 #endif // SPML_CONVERT_H
```

10.9 Файл geodesy.h

Земные эллипсоиды, геодезические задачи (персчеты координат)

#include <cassert>
#include <string>
#include <vector>
#include <compare.h>
#include <convert.h>
#include <units.h>
Граф файлов, в которые включается этот файл:



Классы

- class SPML::Geodesy::CEllipsoid

  Земной эллипсоид
- struct SPML::Geodesy::Geographic

  Географические координаты (широта, долгота)

- struct SPML::Geodesy::Geodetic

  Геодезические координаты (широта, долгота, высота)
- struct SPML::Geodesy::RAD

  Радиолокационные координаты (расстояние по ортодроме, азимут, конечный азимут)
- struct SPML::Geodesy::XYZ

  3D декартовы ортогональные координаты (X, Y, Z)
- struct SPML::Geodesy::ENU

  Координаты ENU (East-North-Up)
- struct SPML::Geodesy::UVW

  Координаты UVW.
- struct SPML::Geodesy::AER

  Локальные сферические координаты AER (Azimuth-Elevation-Range, Азимут-Угол места-Дальность)

Пространства имен

- namespace SPML

  Специальная библиотека программных модулей (СБ ПМ)
- namespace SPML::Geodesy

  Геодезические функции и функции перевода координат
- namespace SPML::Geodesy::Ellipsoids

  Земные эллипсоиды

Функции

- static CEllipsoid SPML::Geodesy::Ellipsoids::WGS84 ()

  Эллипсоид WGS84 (EPSG:7030)
- static CEllipsoid SPML::Geodesy::Ellipsoids::GRS80 ()

  Эллипсоид GRS80 (EPSG:7019)
- static CEllipsoid SPML::Geodesy::Ellipsoids::PZ90 ()

  Эллипсоид ПЗ-90 (EPSG:7054)
- static CEllipsoid SPML::Geodesy::Ellipsoids::Krassowsky1940 ()

  Эллипсоид Красовского 1940 (EPSG:7024)
- static CEllipsoid SPML::Geodesy::Ellipsoids::Sphere6371 ()

  Сфера радиусом 6371000.0 [м] (EPSG:7035)
- static CEllipsoid SPML::Geodesy::Ellipsoids::Sphere6378 ()

  Сфера радиусом 6378000.0 [м].
- static CEllipsoid SPML::Geodesy::Ellipsoids::SphereKrassowsky1940 ()

  Сфера радиусом большой полуоси эллипсоида Красовского 1940 (EPSG:7024)
- static const SPML::Geodesy::Ellipsoids::__attribute__ ((unused)) std

  Возвращает доступные предопределенные эллипсоидоы
- void SPML::Geodesy::GEOtoRAD (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double latStart, double lonStart, double latEnd, double lonEnd, double &d, double &az, double &azEnd=dummy_double)

  Пересчет географических координат в радиолокационные (Обратная геодезическая задача)
- RAD SPML::Geodesy::GEOtoRAD (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geographic &start, const Geographic &end)

  Пересчет географических координат в радиолокационные (Обратная геодезическая задача)

- void SPML::Geodesy::RADtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double latStart, double lonStart, double d, double az, double &latEnd, double &lonEnd, double &azEnd=dummy_double)

  Пересчет радиолокационных координат в географические (Прямая геодезическая задача)
- Geographic SPML::Geodesy::RADtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geographic &start, const RAD &rad, double &azEnd=dummy_double)

  Пересчет радиолокационных координат в географические (Прямая геодезическая задача)
- void SPML::Geodesy::GEOtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double lat, double lon, double h, double &x, double &y, double &z)

  Пересчет широты, долготы, высоты в декартовые геоцентрические координаты
- XYZ SPML::Geodesy::GEOtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geodetic point)

  Пересчет широты, долготы, высоты в декартовые геоцентрические координаты
- void SPML::Geodesy::ECEFtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double x, double y, double z, double &lat, double &lon, double &h)

  Пересчет декартовых геоцентрических координат в широту, долготу, высоту
- Geodetic SPML::Geodesy::ECEFtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, XYZ &point)

  Пересчет декартовых геоцентрических координат в широту, долготу, высоту
- double SPML::Geodesy::XYZtoDistance (double x1, double y1, double z1, double x2, double y2, double z2)

  Вычисление расстояния между точками в декартовых координатах
- double SPML::Geodesy::XYZtoDistance (const XYZ &point1, const XYZ &point2)

  Вычисление расстояния между точками в декартовых координатах
- void SPML::Geodesy::ECEF_offset (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double lat1, double lon1, double h1, double lat2, double lon2, double h2, double &dX, double &dY, double &dZ)

  ECEF смещение (разница в декартовых ECEF координатах двух точек)
- XYZ SPML::Geodesy::ECEF_offset (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geodetic &point1, const Geodetic &point2)

  ECEF смещение (разница в декартовых ECEF координатах двух точек)
- void SPML::Geodesy::ECEFtoENU (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double x, double y, double z, double lat, double lon, double h, double &xEast, double &yNorth, double &zUp)

  Перевод ECEF координат точки в ENU относительно географических координат опорной точки (lat, lon)
- ENU SPML::Geodesy::ECEFtoENU (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const XYZ &ecef, const Geodetic &point)

  Перевод ECEF координат точки в ENU относительно географических координат опорной точки point.
- void SPML::Geodesy::ECEFtoENUV (const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double dX, double dY, double dZ, double lat, double lon, double &xEast, double &yNorth, double &zUp)

  Перевод ECEF координат точки в ENU относительно географических координат (lat, lon)
- ENU SPML::Geodesy::ECEFtoENUV (const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const XYZ &shift, const Geographic &point)

  Перевод ECEF координат точки в ENU относительно географических координат point.
- void SPML::Geodesy::ENUtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double e, double n, double u, double lat, double lon, double h, double &x, double &y, double &z)

Перевод ENU координат точки в ECEF относительно географических координат опорной точки (lat, lon)

- XYZ SPML::Geodesy::ENUtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const ENU &enu, const Geodetic &point)

  Перевод ENU координат точки в ECEF относительно географических координат точки point.
- void SPML::Geodesy::ENUtoAER (const Units::TRangeUnit &rangeUnit, const Units::TAngle↩ Unit &angleUnit, double xEast, double yNorth, double zUp, double &az, double &elev, double &slantRange)

  Перевод ENU координат точки в AER координаты
- AER SPML::Geodesy::ENUtoAER (const Units::TRangeUnit &rangeUnit, const Units::TAngle↩ Unit &angleUnit, const ENU &point)

  Перевод ENU координат точки в AER координаты
- void SPML::Geodesy::AERtoENU (const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double az, double elev, double slantRange, double &xEast, double &yNorth, double &zUp)

  Перевод AER координат точки в ENU координаты
- ENU SPML::Geodesy::AERtoENU (const Units::TRangeUnit &rangeUnit, const Units::TAngle↩ Unit &angleUnit, const AER &aer)

  Перевод ENU координат точки в AER координаты
- void SPML::Geodesy::GEOtoENU (const CEllipsoid &ellipsoid, const Units::TRangeUnit &range↩ Unit, const Units::TAngleUnit &angleUnit, double lat, double lon, double h, double lat0, double lon0, double h0, double &xEast, double &yNorth, double &zUp)

  Перевод геодезических координат GEO точки point в координаты ENU относительно опорной точки
- ENU SPML::Geodesy::GEOtoENU (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geodetic &point, const Geodetic &anchor)

  Перевод геодезических координат GEO точки point в координаты ENU относительно опорной точки
- void SPML::Geodesy::ENUtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &range↩ Unit, const Units::TAngleUnit &angleUnit, double xEast, double yNorth, double zUp, double lat0, double lon0, double h0, double &lat, double &lon, double &h)

  Перевод координат ENU в геодезические координаты GEO относительно опорной точки
- Geodetic SPML::Geodesy::ENUtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const ENU &point, const Geodetic &anchor)

  Перевод координат ENU в геодезические координаты GEO относительно опорной точки
- void SPML::Geodesy::GEOtoAER (const CEllipsoid &ellipsoid, const Units::TRangeUnit &range↩ Unit, const Units::TAngleUnit &angleUnit, double lat1, double lon1, double h1, double lat2, double lon2, double h2, double &az, double &elev, double &slantRange)

  Вычисление AER координат между двумя геодезическими точками
- AER SPML::Geodesy::GEOtoAER (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geodetic &point1, const Geodetic &point2)

  Вычисление AER координат между двумя геодезическими точками
- void SPML::Geodesy::AERtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &range↩ Unit, const Units::TAngleUnit &angleUnit, double az, double elev, double slantRange, double lat0, double lon0, double h0, double &lat, double &lon, double &h)

  Перевод AER координат в геодезические относительно опорной точки
- Geodetic SPML::Geodesy::AERtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const AER &aer, const Geodetic &anchor)

  Перевод AER координат в геодезические относительно опорной точки
- void SPML::Geodesy::AERtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double az, double elev, double slantRange, double lat0, double lon0, double h0, double &x, double &y, double &z)

  Перевод AER координат относительно опорной точки в глобальные декартовые

- XYZ SPML::Geodesy::AERtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const AER &aer, const Geodetic &anchor)

  Перевод AER координат относительно опорной точки в глобальные декартовые
- void SPML::Geodesy::ECEFtoAER (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double x, double y, double z, double lat0, double lon0, double h0, double &az, double &elev, double &slantRange)

  Перевод AER координат относительно опорной точки в глобальные декартовые
- AER SPML::Geodesy::ECEFtoAER (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const XYZ &ecef, const Geodetic &anchor)

  Перевод AER координат относительно опорной точки в глобальные декартовые
- void SPML::Geodesy::ENUtoUVW (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double xEast, double yNorth, double zUp, double lat0, double lon0, double &u, double &v, double &w)

  Перевод ENU координат точки в UVW координаты
- UVW SPML::Geodesy::ENUtoUVW (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const ENU &enu, const Geographic &point)

  Перевод ENU координат точки в UVW координаты
- double SPML::Geodesy::CosAngleBetweenVectors (double x1, double y1, double z1, double x2, double y2, double z2)

  Косинус угла между векторами в евклидовом пространстве
- double SPML::Geodesy::CosAngleBetweenVectors (const XYZ &point1, const XYZ &point2)

  Косинус угла между векторами в евклидовом пространстве
- double SPML::Geodesy::AngleBetweenVectors (double x1, double y1, double z1, double x2, double y2, double z2)

  Угол между векторами в евклидовом пространстве
- double SPML::Geodesy::AngleBetweenVectors (const XYZ &vec1, const XYZ &vec2)

  Угол между векторами в евклидовом пространстве
- void SPML::Geodesy::VectorFromTwoPoints (double x1, double y1, double z1, double x2, double y2, double z2, double &xV, double &yV, double &zV)

  Вектор из координат двух точек
- XYZ SPML::Geodesy::VectorFromTwoPoints (const XYZ &point1, const XYZ &point2)

  Вектор, полученный из координат двух точек

Переменные

- static double SPML::Geodesy::dummy_double

10.9.1 Подробное описание

Земные эллипсоиды, геодезические задачи (персчеты координат)

http://wiki.gis.com/wiki/index.php/Geodetic_system

Дата

06.11.19 - создан

Автор

Соболев А.А.

См. определение в файле geodesy.h

10.10 geodesy.h

См. документацию.

```
00001 //-----------------------------------------------------------------------------------------------------------------
00011
00012 #ifndef SPML_GEODESY_H
00013 #define SPML_GEODESY_H
00014
00015 // System includes:
00016 #include <cassert>
00017 #include <string>
00018 #include <vector>
00019
00020 // SPML includes:
00021 #include <compare.h>
00022 #include <convert.h>
00023 #include <units.h>
00024
00025 namespace SPML
00026 {
00027 namespace Geodesy
00028 {
00029 //-----------------------------------------------------------------------------------------------------------------
00033 class CEllipsoid
00034 {
00035 public:
00040     std::string Name() const
00041     {
00042         return name;
00043     }
00044
00049     double A() const
00050     {
00051         return a;
00052     }
00053
00058     double B() const
00059     {
00060         return b;
00061     }
00062
00067     double F() const
00068     {
00069         return f;
00070     }
00071
00076     double Invf() const
00077     {
00078         return invf;
00079     }
00080
00085     double EccentricityFirst() const
00086     {
00087         return ( std::sqrt( ( a * a ) - ( b * b ) ) / a );
00088     }
00089
00094     double EccentricityFirstSquared() const
00095     {
00096         return ( 1.0 - ( ( b * b ) / ( a * a ) ) );
00097     }
00098
00103     double EccentricitySecond() const
00104     {
00105         return ( std::sqrt( ( a * a ) - ( b * b ) ) / b );
00106     }
00107
00112     double EccentricitySecondSquared() const
00113     {
00114         return ( ( ( a * a ) / ( b * b ) ) - 1.0 );
00115     }
00116
00120     CEllipsoid();
00121
00130     CEllipsoid( std::string ellipsoidName, double semiMajorAxis, double semiMinorAxis, double inverseFlattening, bool
      isInvfDef );
00131
00132 private: // Доступ к параметрам эллипсоида после его создания не предполагается, поэтому private
00133     std::string name;
00134     double a;
00135     double b;
00136     double invf;
00137     double f;
00138 };
00139
00140 namespace Ellipsoids
```

```
00141 {
00142 //-------------------------------------------------------------------------------------------------------------
00143 //
00144 //                                                Земные эллипсоиды:
00145 //
00146 //  1) Эллипсоид WGS84, https://epsg.io/7030-ellipsoid
00147 //  2) Эллипсоид GRS80, https://epsg.io/7019-ellipsoid
00148 //  3) Эллипсоид ПЗ-90, https://epsg.io/7054-ellipsoid
00149 //  4) Эллипсоид Красовского, https://epsg.io/7024-ellipsoid
00150 //  5) Сфера радиусом 6371000.0 [м], https://epsg.io/7035-ellipsoid
00151 //  6) Сфера радиусом 6378000.0 [м]
00152 //  7) Сфера радиусом большой полуоси эллипсоида Красовского 1940 (6378245.0 [м])
00153 //
00154
00159 static CEllipsoid WGS84()
00160 {
00161     return CEllipsoid( "WGS84 (EPSG:7030)", 6378137.0, 0.0, 298.257223563, true );
00162 }
00163
00168 static CEllipsoid GRS80()
00169 {
00170     return CEllipsoid( "GRS80 (EPSG:7019)", 6378137.0, 0.0, 298.257222101, true );
00171 }
00172
00177 static CEllipsoid PZ90()
00178 {
00179     return CEllipsoid( "PZ90 (EPSG:7054)", 6378136.0, 0.0, 298.257839303, true );
00180 }
00181
00186 static CEllipsoid Krassowsky1940()
00187 {
00188     return CEllipsoid( "Krasovsky1940 (EPSG:7024)", 6378245.0, 0.0, 298.3, true );
00189 }
00190
00191
00196 static CEllipsoid Sphere6371()
00197 {
00198     return CEllipsoid( "Sphere 6371000.0 [м] (EPSG:7035)", 6371000.0, 6371000.0, 0.0, false );
00199 }
00200
00205
00206 static CEllipsoid Sphere6378()
00207 {
00208     return CEllipsoid( "Sphere 6378000.0 [м]", 6378000.0, 6378000.0, 0.0, false );
00209 }
00210
00215 static CEllipsoid SphereKrassowsky1940()
00216 {
00217     return CEllipsoid( "SphereRadiusKrasovsky1940 (EPSG:7024)", 6378245.0, 6378245.0, 0.0, false );
00218 }
00219
00224 [[maybe_unused]]
00225 static const __attribute__ ((unused)) std::vector<CEllipsoid> GetPredefinedEllipsoids()
00226 {
00227     return std::vector<CEllipsoid>{
00228         WGS84(),
00229         GRS80(),
00230         PZ90(),
00231         Krassowsky1940(),
00232         Sphere6371(),
00233         Sphere6378(),
00234         SphereKrassowsky1940()
00235     };
00236 }
00237
00238 } // end namespace Ellipsoids
00239
00240 //-------------------------------------------------------------------------------------------------------------
00244 struct Geographic
00245 {
00246     double Lat;
00247     double Lon;
00248
00252     Geographic() : Lat( 0.0 ), Lon( 0.0 )
00253     {}
00254
00260     Geographic( double lat, double lon ) : Lat( lat ), Lon( lon )
00261     {}
00262 };
00263
00264 //-------------------------------------------------------------------------------------------------------------
00268 struct Geodetic : public Geographic
00269 {
00270     double Height;
00271
00275     Geodetic() : Geographic( 0.0, 0.0 ), Height( 0.0 )
00276     {}
```

```
00277
00284     Geodetic( double lat, double lon, double h ) : Geographic( lat, lon ), Height( h )
00285     {}
00286 };
00287
00288 //---------------------------------------------------------------------------------------------------------------
00293 struct RAD
00294 {
00295     double R;
00296     double Az;
00297     double AzEnd;
00298
00302     RAD() : R( 0.0 ), Az( 0.0 ), AzEnd( 0.0 )
00303     {}
00304
00311     RAD( double r, double az, double azEnd ) : R( r ), Az( az ), AzEnd( azEnd )
00312     {}
00313 };
00314
00315 //---------------------------------------------------------------------------------------------------------------
00319 struct XYZ
00320 {
00321     double X;
00322     double Y;
00323     double Z;
00324
00328     XYZ() : X( 0.0 ), Y( 0.0 ), Z( 0.0 )
00329     {}
00330
00337     XYZ( double x, double y, double z ) : X( x ), Y( y ), Z( z )
00338     {}
00339 };
00340
00341 //---------------------------------------------------------------------------------------------------------------
00345 struct ENU
00346 {
00347     double E;
00348     double N;
00349     double U;
00350
00354     ENU() : E( 0.0 ), N( 0.0 ), U( 0.0 )
00355     {}
00356
00363     ENU( double e, double n, double u ) : E( e ), N( n ), U( u )
00364     {}
00365 };
00366
00367 //---------------------------------------------------------------------------------------------------------------
00371 struct UVW
00372 {
00373     double U;
00374     double V;
00375     double W;
00376
00380     UVW() : U( 0.0 ), V( 0.0 ), W( 0.0 )
00381     {}
00382
00389     UVW( double u, double v, double w ) : U( u ), V( v ), W( w )
00390     {}
00391 };
00392
00393 //---------------------------------------------------------------------------------------------------------------
00397 struct AER
00398 {
00399     double A;
00400     double E;
00401     double R;
00402
00406     AER() : A( 0.0 ), E( 0.0 ), R( 0.0 )
00407     {}
00408
00415     AER( double a, double e, double r ) : A( a ), E( e ), R( r )
00416     {}
00417 };
00418
00419 //---------------------------------------------------------------------------------------------------------------
00420 //
00421 //                                    Функции пересчета координат
00422 //
00423 [[maybe_unused]]
00424 static double dummy_double; // Заглушка для списка параметров функций без перегрузки
00425
00426 //---------------------------------------------------------------------------------------------------------------
00445 void GEOtoRAD( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00446     double latStart, double lonStart, double latEnd, double lonEnd, double &d, double &az, double &azEnd =
      dummy_double );
00447
```

```
00463 RAD GEOtoRAD( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00464     const Geographic &start, const Geographic &end );
00465
00466 //---------------------------------------------------------------------------
00485 void RADtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00486     double latStart, double lonStart, double d, double az, double &latEnd, double &lonEnd, double &azEnd =
      dummy_double );
00487
00504 Geographic RADtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00505     const Geographic &start, const RAD &rad, double &azEnd = dummy_double );
00506
00507 //---------------------------------------------------------------------------
00526 void GEOtoECEF( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00527     double lat, double lon, double h, double &x, double &y, double &z );
00528
00543 XYZ GEOtoECEF( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00544     const Geodetic point );
00545
00546 //---------------------------------------------------------------------------
00566 void ECEFtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00567     double x, double y, double z, double &lat, double &lon, double &h );
00568
00584 Geodetic ECEFtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00585     XYZ &point );
00586
00587 //---------------------------------------------------------------------------
00600 double XYZtoDistance( double x1, double y1, double z1, double x2, double y2, double z2 );
00601
00609 double XYZtoDistance( const XYZ &point1, const XYZ &point2 );
00610
00611 //---------------------------------------------------------------------------
00627 void ECEF_offset( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00628     double lat1, double lon1, double h1, double lat2, double lon2, double h2, double &dX, double &dY, double &dZ );
00629
00639 XYZ ECEF_offset( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00640     const Geodetic &point1, const Geodetic &point2 );
00641
00642 //---------------------------------------------------------------------------
00659 void ECEFtoENU( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00660     double x, double y, double z, double lat, double lon, double h, double &xEast, double &yNorth, double &zUp );
00661
00672 ENU ECEFtoENU( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00673     const XYZ &ecef, const Geodetic &point );
00674
00675 //---------------------------------------------------------------------------
00689 void ECEFtoENUV( const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00690     double dX, double dY, double dZ, double lat, double lon, double &xEast, double &yNorth, double &zUp );
00691
00700 ENU ECEFtoENUV( const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00701     const XYZ &shift, const Geographic &point );
00702
00703 //---------------------------------------------------------------------------
00720 void ENUtoECEF( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00721     double e, double n, double u, double lat, double lon, double h, double &x, double &y, double &z );
00722
00733 XYZ ENUtoECEF( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00734     const ENU &enu, const Geodetic &point );
00735
00736 //---------------------------------------------------------------------------
00748 void ENUtoAER( const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00749     double xEast, double yNorth, double zUp, double &az, double &elev, double &slantRange );
00750
00758 AER ENUtoAER( const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const ENU &point );
00759
00760 //---------------------------------------------------------------------------
00772 void AERtoENU( const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00773     double az, double elev, double slantRange, double &xEast, double &yNorth, double &zUp );
00774
00782 ENU AERtoENU( const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const AER &aer );
00783
00784 //---------------------------------------------------------------------------
00800 void GEOtoENU( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00801     double lat, double lon, double h, double lat0, double lon0, double h0, double &xEast, double &yNorth, double &zUp );
00802
00812 ENU GEOtoENU( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
```

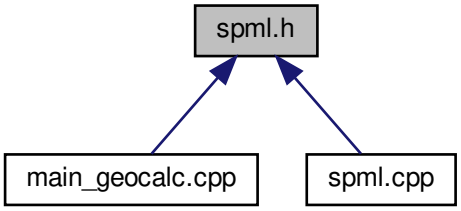
```
        &angleUnit,
00813     const Geodetic &point, const Geodetic &anchor );
00814
00815 //---------------------------------------------------------------------------------------------------------
00830
00832 void ENUtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00833     double xEast, double yNorth, double zUp, double lat0, double lon0, double h0, double &lat, double &lon, double &h );
00834
00844 Geodetic ENUtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00845     const ENU &point, const Geodetic &anchor );
00846
00847 //---------------------------------------------------------------------------------------------------------
00863 void GEOtoAER( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00864     double lat1, double lon1, double h1, double lat2, double lon2, double h2, double &az, double &elev, double
        &slantRange );
00865
00875 AER GEOtoAER( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00876     const Geodetic &point1, const Geodetic &point2 );
00877
00878 //---------------------------------------------------------------------------------------------------------
00894 void AERtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00895     double az, double elev, double slantRange, double lat0, double lon0, double h0, double &lat, double &lon, double &h );
00896
00906 Geodetic AERtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00907     const AER &aer, const Geodetic &anchor );
00908
00909 //---------------------------------------------------------------------------------------------------------
00925 void AERtoECEF( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00926     double az, double elev, double slantRange, double lat0, double lon0, double h0, double &x, double &y, double &z );
00927
00937 XYZ AERtoECEF( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00938     const AER &aer, const Geodetic &anchor );
00939 //---------------------------------------------------------------------------------------------------------
00955 void ECEFtoAER( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00956     double x, double y, double z, double lat0, double lon0, double h0, double &az, double &elev, double &slantRange );
00957
00967 AER ECEFtoAER(const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00968     const XYZ &ecef, const Geodetic &anchor );
00969 //---------------------------------------------------------------------------------------------------------
00984 void ENUtoUVW( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00985     double xEast, double yNorth, double zUp, double lat0, double lon0, double &u, double &v, double &w );
00986
00997 UVW ENUtoUVW( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00998     const ENU &enu, const Geographic &point );
00999 //---------------------------------------------------------------------------------------------------------
01011 double CosAngleBetweenVectors( double x1, double y1, double z1, double x2, double y2, double z2 );
01012
01020 double CosAngleBetweenVectors( const XYZ &point1, const XYZ &point2 );
01021
01022 //---------------------------------------------------------------------------------------------------------
01034 double AngleBetweenVectors( double x1, double y1, double z1, double x2, double y2, double z2 );
01035
01043 double AngleBetweenVectors( const XYZ &vec1, const XYZ &vec2 );
01044
01045 //---------------------------------------------------------------------------------------------------------
01059 void VectorFromTwoPoints( double x1, double y1, double z1, double x2, double y2, double z2, double &xV, double &yV,
        double &zV );
01060
01068 XYZ VectorFromTwoPoints( const XYZ &point1, const XYZ &point2 );
01069
01070 } // end namespace SPML
01071 } // end namespace Geodesy
01072 #endif // SPML_GEODESY_H
```

10.11 Файл spml.h

SPML (Special Program Modules Library) - СБ ПМ (Специальная Библиотека Программных Модулей)

#include <string>
#include <compare.h>

#include <consts.h>
#include <convert.h>
#include <geodesy.h>
#include <units.h>
Граф файлов, в которые включается этот файл:



Пространства имен

- namespace SPML

  Специальная библиотека программных модулей (СБ ПМ)

Функции

- std::string SPML::GetVersion ()

  Возвращает строку, содержащую информацию о версии
- void SPML::ClearConsole ()

  Очистка консоли (терминала) в *nix.

10.11.1 Подробное описание

SPML (Special Program Modules Library) - СБ ПМ (Специальная Библиотека Программных Модулей)

Единый заголовочный файл библиотеки SPML (его подключение включает полностью всю библиотеку).

Дата

14.07.20 - создан

Автор

Соболев А.А.

См. определение в файле spml.h

10.12 spml.h

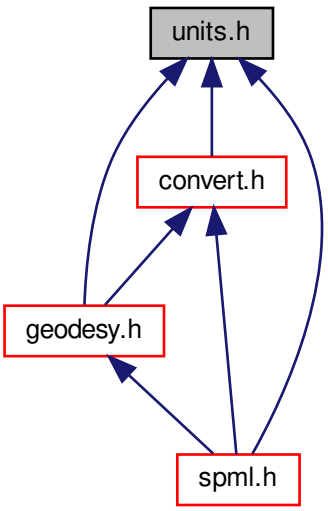
См. документацию.

```
00001 //-----------------------------------------------------------------------------------------------------------
00013
00014 #ifndef SPML_H
00015 #define SPML_H
00016
00017 // System includes:
00018 #include <string>
00019
00020 // SPML includes:
00021 #include <compare.h>
00022 #include <consts.h>
00023 #include <convert.h>
00024 #include <geodesy.h>
00025 #include <units.h>
00026
00027 namespace SPML
00028 {
00029 //-----------------------------------------------------------------------------------------------------------
00034 std::string GetVersion();
00035
00036 //-----------------------------------------------------------------------------------------------------------
00040 void ClearConsole();
00041
00042 } // end namespace SPML
00043 #endif // SPML_H
```

10.13 Файл units.h

Единицы измерения физических величин, форматы чисел

Граф файлов, в которые включается этот файл:



Пространства имен

- namespace SPML

  Специальная библиотека программных модулей (СБ ПМ)
- namespace SPML::Units

  Единицы измерения физических величин, форматы чисел

Перечисления

- enum SPML::Units::TNumberFormat : int { SPML::Units::NF_Fixed = 0 , SPML::Units::NF_Scientific = 1 }

  Формат числа
- enum SPML::Units::TAngleUnit : int { SPML::Units::AU_Radian = 0 , SPML::Units::AU_Degree = 1 }

  Размерность угловых единиц
- enum SPML::Units::TRangeUnit : int { SPML::Units::RU_Meter = 0 , SPML::Units::RU_Kilometer = 1 }

  Размерность единиц дальности

10.13.1 Подробное описание

Единицы измерения физических величин, форматы чисел

Дата

17.02.20 - создан

Автор

Соболев А.А.

См. определение в файле units.h

10.14 units.h

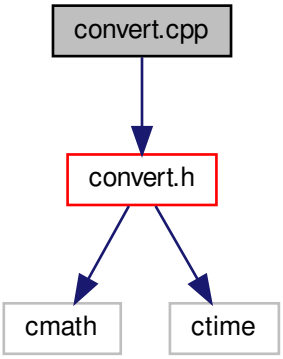
См. документацию.

```
00001 //-----------------------------------------------------------------------------------------------------------
00010
00011 #ifndef SPML_UNITS_H
00012 #define SPML_UNITS_H
00013
00014 namespace SPML
00015 {
00016 namespace Units
00017 {
00018 //-----------------------------------------------------------------------------------------------------------
00022 enum TNumberFormat : int
00023 {
00024     NF_Fixed = 0,
00025     NF_Scientific = 1
00026 };
00027
00031 enum TAngleUnit : int
00032 {
00033     AU_Radian = 0,
00034     AU_Degree = 1
00035 };
00036
00040 enum TRangeUnit : int
00041 {
00042     RU_Meter = 0,
00043     RU_Kilometer = 1
00044 };
00045
00046 }
00047 }
00048 #endif // SPML_UNITS_H
```

10.15 Файл convert.cpp

Переводы единиц библиотеки СБПМ

#include <convert.h>
Граф включаемых заголовочных файлов для convert.cpp:



Пространства имен

- namespace SPML

  Специальная библиотека программных модулей (СБ ПМ)
- namespace SPML::Convert

  Переводы единиц

Функции

- float SPML::Convert::AngleTo360 (float angle, const Units::TAngleUnit &au)

  Приведение угла в [0,360) градусов или [0,2PI) радиан
- double SPML::Convert::AngleTo360 (double angle, const Units::TAngleUnit &au)

  Приведение угла в [0,360) градусов или [0,2PI) радиан
- float SPML::Convert::EpsToMP90 (float angle, const Units::TAngleUnit &au)

  Приведение угла места в [-90,90] градусов или [-PI/2, PI/2] радиан
- double SPML::Convert::EpsToMP90 (double angle, const Units::TAngleUnit &au)

  Приведение угла места в [-90,90] градусов или [-PI/2, PI/2] радиан
- void SPML::Convert::UnixTimeToHourMinSec (int rawtime, int &hour, int &min, int &sec, int &day=dummy_int, int &mon=dummy_int, int &year=dummy_int)

  Перевод целого числа секунд с 00:00:00 01.01.1970 в часы/минуты/секунды/день/месяц/год
- const std::string SPML::Convert::CurrentDateTimeToString ()

  Получение текущей даты и времени
- double SPML::Convert::CheckDeltaAngle (double deltaAngle, const SPML::Units::TAngleUnit &au)

  Проверка разницы в углах

10.15.1 Подробное описание

Переводы единиц библиотеки СБПМ

Дата

14.07.20 - создан

Автор

Соболев А.А.

См. определение в файле convert.cpp

10.16 convert.cpp

См. документацию.

```
00001 //-----------------------------------------------------------------------------------------------------------
00010
00011 #include <convert.h>
00012
00013 namespace SPML
00014 {
00015 namespace Convert
00016 {
00017 //-----------------------------------------------------------------------------------------------------------
00018 float AngleTo360( float angle, const Units::TAngleUnit &au )
00019 {
00020     float _angle = angle;
00021     switch ( au ) {
00022         case Units::TAngleUnit::AU_Degree:
00023         {
00024             // old
00025 //            while( _angle >= 360.0f ) {
00026 //                _angle -= 360.0f;
00027 //            }
00028 //            while( _angle < 0.0f ) {
00029 //                _angle += 360.0f;
00030 //            }
00031
00032 //            // new 1
00033 //            float n = std::floor( _angle / 360.0 );
00034 //            if( _angle >= 360.0 ) {
00035 //                _angle -= ( 360.0 * n );
00036 //            } else if( _angle < 0.0 ) {
00037 //                _angle += ( 360.0 * n );
00038 //            }
00039
00040             // new 2
00041             float n = std::floor( _angle / 360.0 );
00042             if( ( _angle >= 360.0 ) || ( _angle < 0.0 ) ) {
00043                 _angle -= ( 360.0 * n );
00044             }
00045             break;
00046         }
00047         case Units::TAngleUnit::AU_Radian:
00048         {
00049             // old
00050 //            while( _angle >= Consts::PI_2_F ) {
00051 //                _angle -= Consts::PI_2_F;
00052 //            }
00053 //            while( _angle < 0.0f ) {
00054 //                _angle += Consts::PI_2_F;
00055 //            }
00056
00057             // new 1
00058 //            float n = std::floor( _angle / Consts::PI_2_F );
00059 //            if( _angle >= Consts::PI_2_F ) {
00060 //                _angle -= ( Consts::PI_2_F * n );
00061 //            } else if( _angle < 0.0 ) {
00062 //                _angle += ( Consts::PI_2_F * n );
00063 //            }
```

```
00064
00065             // new 2
00066             float n = std::floor( _angle / Consts::PI_2_F );
00067             if( ( _angle >= Consts::PI_2_F ) || ( _angle < 0.0 ) ) {
00068                 _angle -= ( Consts::PI_2_F * n );
00069             }
00070             break;
00071         }
00072         default:
00073             assert( false );
00074     }
00075     return _angle;
00076 }
00077
00078 double AngleTo360( double angle, const Units::TAngleUnit &au )
00079 {
00080     double _angle = angle;
00081     switch ( au ) {
00082         case Units::TAngleUnit::AU_Degree:
00083         {
00084             // old
00085 //            while( _angle >= 360.0 ) {
00086 //                _angle -= 360.0;
00087 //            }
00088 //            while( _angle < 0.0 ) {
00089 //                _angle += 360.0;
00090 //            }
00091
00092             // new 1
00093 //            double n = std::floor( _angle / 360.0 );
00094 //            if( _angle >= 360.0 ) {
00095 //                _angle -= ( 360.0 * n );
00096 //            } else if( _angle < 0.0 ) {
00097 //                _angle += ( 360.0 * n );
00098 //            }
00099
00100             // new 2
00101             double n = std::floor( _angle / 360.0 );
00102             if( ( _angle >= 360.0 ) || ( _angle < 0.0 ) ) {
00103                 _angle -= ( 360.0 * n );
00104             }
00105             break;
00106         }
00107         case Units::TAngleUnit::AU_Radian:
00108         {
00109             // old
00110 //            while( _angle >= Consts::PI_2_F ) {
00111 //                _angle -= Consts::PI_2_F;
00112 //            }
00113 //            while( _angle < 0.0 ) {
00114 //                _angle += Consts::PI_2_F;
00115 //            }
00116
00117             // new 1
00118 //            double n = std::floor( _angle / Consts::PI_2_D );
00119 //            if( _angle >= Consts::PI_2_D ) {
00120 //                _angle -= ( Consts::PI_2_D * n );
00121 //            } else if( _angle < 0.0 ) {
00122 //                _angle += ( Consts::PI_2_D * n );
00123 //            }
00124
00125             // new 2
00126             double n = std::floor( _angle / Consts::PI_2_D );
00127             if( ( _angle >= Consts::PI_2_F ) || ( _angle < 0.0 ) ) {
00128                 _angle -= ( Consts::PI_2_F * n );
00129             }
00130             break;
00131         }
00132         default:
00133             assert( false );
00134     }
00135     return _angle;
00136 }
00137 //---------------------------------------------------------------------------------------------------------------
00138 float EpsToMP90( float angle, const Units::TAngleUnit &au )
00139 {
00140     float _angle = angle;
00141     switch ( au ) {
00142         case Units::TAngleUnit::AU_Degree:
00143         {
00144             while( _angle > 90.0f ) {
00145                 _angle -= 180.0f;
00146             }
00147             while( _angle <= -90.0f ) {
00148                 _angle += 180.0f;
00149             }
00150             if( ( AngleTo360( std::abs( angle ), Units::TAngleUnit::AU_Degree ) > 90.0f ) &&
```

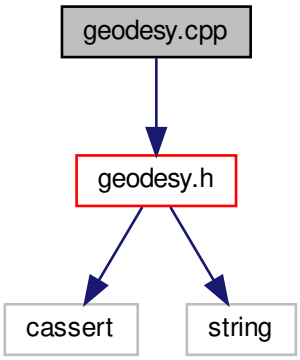
```
00151             ( AngleTo360( std::abs( angle ), Units::TAngleUnit::AU_Degree ) <= 270.0f ) )
00152         {
00153             _angle *= -1.0f;
00154         }
00155         break;
00156     }
00157     case Units::TAngleUnit::AU_Radian:
00158     {
00159         while( _angle > Consts::PI_05_F ) {
00160             _angle -= Consts::PI_F;
00161         }
00162         while( _angle <= -Consts::PI_05_F ) {
00163             _angle += Consts::PI_F;
00164         }
00165         if( ( AngleTo360( std::abs( angle ), Units::TAngleUnit::AU_Radian ) > Consts::PI_05_F ) &&
00166             ( AngleTo360( std::abs( angle ), Units::TAngleUnit::AU_Radian ) <= ( 3.0f * Consts::PI_05_F ) ) )
00167         {
00168             _angle *= -1.0f;
00169         }
00170         break;
00171     }
00172     default:
00173         assert( false );
00174 }
00175 return _angle;
00176 }
00177
00178 double EpsToMP90( double angle, const Units::TAngleUnit &au )
00179 {
00180     double _angle = angle;
00181     switch ( au ) {
00182         case Units::TAngleUnit::AU_Degree:
00183         {
00184             // old
00185 //          while( _angle > 90.0 ) {
00186 //              _angle -= 180.0;
00187 //          }
00188 //          while( _angle <= -90.0 ) {
00189 //              _angle += 180.0;
00190 //          }
00191 //          if( ( AngleTo360( std::abs( angle ), Units::TAngleUnit::AU_Degree ) > 90.0 ) &&
00192 //              ( AngleTo360( std::abs( angle ), Units::TAngleUnit::AU_Degree ) <= 270.0 ) )
00193 //          {
00194 //              _angle *= -1.0;
00195 //          }
00196
00197             // new
00198             _angle = std::asin( std::sin( _angle * DgToRdD ) ) * RdToDgD;
00199             break;
00200         }
00201         case Units::TAngleUnit::AU_Radian:
00202         {
00203 //          while( _angle > Consts::PI_05_D ) {
00204 //              _angle -= Consts::PI_D;
00205 //          }
00206 //          while( _angle <= -Consts::PI_05_D ) {
00207 //              _angle += Consts::PI_D;
00208 //          }
00209 //          if( ( AngleTo360( std::abs( angle ), Units::TAngleUnit::AU_Radian ) > Consts::PI_05_D ) &&
00210 //              ( AngleTo360( std::abs( angle ), Units::TAngleUnit::AU_Radian ) <= ( 3.0 * Consts::PI_05_D ) ) )
00211 //          {
00212 //              _angle *= -1.0;
00213 //          }
00214             _angle = std::asin( std::sin( _angle ) );
00215             break;
00216         }
00217         default:
00218             assert( false );
00219     }
00220     return _angle;
00221 }
00222 //-----------------------------------------------------------------------------------------------------------------
00223 void UnixTimeToHourMinSec( int rawtime, int &hour, int &min, int &sec, int &day, int &mon, int &year )
00224 {
00225     std::time_t temp = rawtime;
00226     std::tm res;
00227     gmtime_r( &temp, &res );
00228     hour = ( res.tm_hour ) % 24;
00229     min = ( res.tm_min ) % 60;
00230     sec = ( res.tm_sec ) % 60;
00231     day = res.tm_mday;
00232     mon = ( res.tm_mon + 1 );
00233     year = ( res.tm_year + 1900 );
00234 }
00235 //-----------------------------------------------------------------------------------------------------------------
00236 const std::string CurrentDateTimeToString() {
00237     time_t now = time( nullptr );
```

```
00238     struct tm tstruct;
00239     char buf[80];
00240     tstruct = *localtime(&now);
00241     // Visit http://en.cppreference.com/w/cpp/chrono/c/strftime
00242     // for more information about date/time format
00243     //strftime( buf, sizeof( buf ), "%Y-%m-%d.%X", &tstruct ); // original
00244     strftime( buf, sizeof( buf ), "%X %d-%m-%Y UTC%z", &tstruct ); // my
00245     return buf;
00246 }
00247 //-----------------------------------------------------------------------------------------------------------
00248 double CheckDeltaAngle( double deltaAngle, const SPML::Units::TAngleUnit &au )
00249 {
00250     double _deltaAngle = deltaAngle;
00251     switch ( au ) {
00252         case SPML::Units::TAngleUnit::AU_Degree:
00253         {
00254 //             if( std::abs( _deltaAngle ) > std::abs( _deltaAngle + 360.0 ) ) {
00255 //                 _deltaAngle += 360.0;
00256 //             }
00257 //             if( std::abs( _deltaAngle ) > std::abs( _deltaAngle - 360.0 ) ) {
00258 //                 _deltaAngle -= 360.0;
00259 //             }
00260             _deltaAngle = std::fmod( std::abs( deltaAngle ) + 180.0, 360.0 ) - 180.0;
00261             if( deltaAngle < 0.0 ) {
00262                 _deltaAngle *= ( -1.0 );
00263             }
00264             break;
00265         }
00266         case SPML::Units::TAngleUnit::AU_Radian:
00267         {
00268 //             if( std::abs( _deltaAngle ) > std::abs( _deltaAngle + ( SPML::Consts::PI_2_D ) ) ) {
00269 //                 _deltaAngle += ( SPML::Consts::PI_2_D );
00270 //             }
00271 //             if( std::abs( _deltaAngle ) > std::abs( _deltaAngle - ( SPML::Consts::PI_2_D ) ) ) {
00272 //                 _deltaAngle -= ( SPML::Consts::PI_2_D );
00273 //             }
00274             _deltaAngle = std::fmod( std::abs( deltaAngle ) + SPML::Consts::PI_D, SPML::Consts::PI_2_D ) -
      SPML::Consts::PI_D;
00275             if( deltaAngle < 0.0 ) {
00276                 _deltaAngle *= ( -1.0 );
00277             }
00278             break;
00279         }
00280         default:
00281             assert( false );
00282     }
00283     return _deltaAngle;
00284 }
00285
00286 }
00287 }
```

10.17 Файл geodesy.cpp

Земные эллипсоиды, геодезические задачи (персчеты координат)

#include <geodesy.h>
Граф включаемых заголовочных файлов для geodesy.cpp:



Пространства имен

- namespace SPML

  Специальная библиотека программных модулей (СБ ПМ)
- namespace SPML::Geodesy

  Геодезические функции и функции перевода координат

Функции

- void SPML::Geodesy::GEOtoRAD (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double latStart, double lonStart, double latEnd, double lonEnd, double &d, double &az, double &azEnd=dummy_double)

  Пересчет географических координат в радиолокационные (Обратная геодезическая задача)
- RAD SPML::Geodesy::GEOtoRAD (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geographic &start, const Geographic &end)

  Пересчет географических координат в радиолокационные (Обратная геодезическая задача)
- void SPML::Geodesy::RADtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double latStart, double lonStart, double d, double az, double &latEnd, double &lonEnd, double &azEnd=dummy_double)

  Пересчет радиолокационных координат в географические (Прямая геодезическая задача)
- Geographic SPML::Geodesy::RADtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geographic &start, const RAD &rad, double &azEnd=dummy_double)

  Пересчет радиолокационных координат в географические (Прямая геодезическая задача)
- void SPML::Geodesy::GEOtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double lat, double lon, double h, double &x, double &y, double &z)

  Пересчет широты, долготы, высоты в декартовые геоцентрические координаты
- XYZ SPML::Geodesy::GEOtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geodetic point)

Пересчет широты, долготы, высоты в декартовые геоцентрические координаты

- void SPML::Geodesy::ECEFtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double x, double y, double z, double &lat, double &lon, double &h)

  Пересчет декартовых геоцентрических координат в широту, долготу, высоту
- Geodetic SPML::Geodesy::ECEFtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, XYZ &point)

  Пересчет декартовых геоцентрических координат в широту, долготу, высоту
- double SPML::Geodesy::XYZtoDistance (double x1, double y1, double z1, double x2, double y2, double z2)

  Вычисление расстояния между точками в декартовых координатах
- double SPML::Geodesy::XYZtoDistance (const XYZ &point1, const XYZ &point2)

  Вычисление расстояния между точками в декартовых координатах
- void SPML::Geodesy::ECEF_offset (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double lat1, double lon1, double h1, double lat2, double lon2, double h2, double &dX, double &dY, double &dZ)

  ECEF смещение (разница в декартовых ECEF координатах двух точек)
- XYZ SPML::Geodesy::ECEF_offset (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geodetic &point1, const Geodetic &point2)

  ECEF смещение (разница в декартовых ECEF координатах двух точек)
- void SPML::Geodesy::ECEFtoENU (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double x, double y, double z, double lat, double lon, double h, double &xEast, double &yNorth, double &zUp)

  Перевод ECEF координат точки в ENU относительно географических координат опорной точки (lat, lon)
- ENU SPML::Geodesy::ECEFtoENU (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const XYZ &ecef, const Geodetic &point)

  Перевод ECEF координат точки в ENU относительно географических координат опорной точки point.
- void SPML::Geodesy::ECEFtoENUV (const Units::TRangeUnit &rangeUnit, const Units::TAngle↩ Unit &angleUnit, double dX, double dY, double dZ, double lat, double lon, double &xEast, double &yNorth, double &zUp)

  Перевод ECEF координат точки в ENU относительно географических координат (lat, lon)
- ENU SPML::Geodesy::ECEFtoENUV (const Units::TRangeUnit &rangeUnit, const Units::↩ TAngleUnit &angleUnit, const XYZ &shift, const Geographic &point)

  Перевод ECEF координат точки в ENU относительно географических координат point.
- void SPML::Geodesy::ENUtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double e, double n, double u, double lat, double lon, double h, double &x, double &y, double &z)

  Перевод ENU координат точки в ECEF относительно географических координат опорной точки (lat, lon)
- XYZ SPML::Geodesy::ENUtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const ENU &enu, const Geodetic &point)

  Перевод ENU координат точки в ECEF относительно географических координат точки point.
- void SPML::Geodesy::ENUtoAER (const Units::TRangeUnit &rangeUnit, const Units::TAngle↩ Unit &angleUnit, double xEast, double yNorth, double zUp, double &az, double &elev, double &slantRange)

  Перевод ENU координат точки в AER координаты
- AER SPML::Geodesy::ENUtoAER (const Units::TRangeUnit &rangeUnit, const Units::TAngle↩ Unit &angleUnit, const ENU &point)

  Перевод ENU координат точки в AER координаты
- void SPML::Geodesy::AERtoENU (const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double az, double elev, double slantRange, double &xEast, double &yNorth, double &zUp)

Перевод AER координат точки в ENU координаты

- ENU SPML::Geodesy::AERtoENU (const Units::TRangeUnit &rangeUnit, const Units::TAngle↩ Unit &angleUnit, const AER &aer)

  Перевод ENU координат точки в AER координаты
- void SPML::Geodesy::GEOtoENU (const CEllipsoid &ellipsoid, const Units::TRangeUnit &range↩ Unit, const Units::TAngleUnit &angleUnit, double lat, double lon, double h, double lat0, double lon0, double h0, double &xEast, double &yNorth, double &zUp)

  Перевод геодезических координат GEO точки point в координаты ENU относительно опорной точки
- ENU SPML::Geodesy::GEOtoENU (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geodetic &point, const Geodetic &anchor)

  Перевод геодезических координат GEO точки point в координаты ENU относительно опорной точки
- void SPML::Geodesy::ENUtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &range↩ Unit, const Units::TAngleUnit &angleUnit, double xEast, double yNorth, double zUp, double lat0, double lon0, double h0, double &lat, double &lon, double &h)

  Перевод координат ENU в геодезические координаты GEO относительно опорной точки
- Geodetic SPML::Geodesy::ENUtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const ENU &point, const Geodetic &anchor)

  Перевод координат ENU в геодезические координаты GEO относительно опорной точки
- void SPML::Geodesy::GEOtoAER (const CEllipsoid &ellipsoid, const Units::TRangeUnit &range↩ Unit, const Units::TAngleUnit &angleUnit, double lat1, double lon1, double h1, double lat2, double lon2, double h2, double &az, double &elev, double &slantRange)

  Вычисление AER координат между двумя геодезическими точками
- AER SPML::Geodesy::GEOtoAER (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const Geodetic &point1, const Geodetic &point2)

  Вычисление AER координат между двумя геодезическими точками
- void SPML::Geodesy::AERtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &range↩ Unit, const Units::TAngleUnit &angleUnit, double az, double elev, double slantRange, double lat0, double lon0, double h0, double &lat, double &lon, double &h)

  Перевод AER координат в геодезические относительно опорной точки
- Geodetic SPML::Geodesy::AERtoGEO (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const AER &aer, const Geodetic &anchor)

  Перевод AER координат в геодезические относительно опорной точки
- void SPML::Geodesy::AERtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double az, double elev, double slantRange, double lat0, double lon0, double h0, double &x, double &y, double &z)

  Перевод AER координат относительно опорной точки в глобальные декартовые
- XYZ SPML::Geodesy::AERtoECEF (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const AER &aer, const Geodetic &anchor)

  Перевод AER координат относительно опорной точки в глобальные декартовые
- void SPML::Geodesy::ECEFtoAER (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double x, double y, double z, double lat0, double lon0, double h0, double &az, double &elev, double &slantRange)

  Перевод AER координат относительно опорной точки в глобальные декартовые
- AER SPML::Geodesy::ECEFtoAER (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const XYZ &ecef, const Geodetic &anchor)

  Перевод AER координат относительно опорной точки в глобальные декартовые
- void SPML::Geodesy::ENUtoUVW (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, double xEast, double yNorth, double zUp, double lat0, double lon0, double &u, double &v, double &w)

  Перевод ENU координат точки в UVW координаты

- UVW SPML::Geodesy::ENUtoUVW (const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const ENU &enu, const Geographic &point)

  Перевод ENU координат точки в UVW координаты
- double SPML::Geodesy::CosAngleBetweenVectors (double x1, double y1, double z1, double x2, double y2, double z2)

  Косинус угла между векторами в евклидовом пространстве
- double SPML::Geodesy::CosAngleBetweenVectors (const XYZ &point1, const XYZ &point2)

  Косинус угла между векторами в евклидовом пространстве
- double SPML::Geodesy::AngleBetweenVectors (double x1, double y1, double z1, double x2, double y2, double z2)

  Угол между векторами в евклидовом пространстве
- double SPML::Geodesy::AngleBetweenVectors (const XYZ &vec1, const XYZ &vec2)

  Угол между векторами в евклидовом пространстве
- void SPML::Geodesy::VectorFromTwoPoints (double x1, double y1, double z1, double x2, double y2, double z2, double &xV, double &yV, double &zV)

  Вектор из координат двух точек
- XYZ SPML::Geodesy::VectorFromTwoPoints (const XYZ &point1, const XYZ &point2)

  Вектор, полученный из координат двух точек

10.17.1 Подробное описание

Земные эллипсоиды, геодезические задачи (персчеты координат)

Дата

06.11.19 - создан

Автор

Соболев А.А.

См. определение в файле geodesy.cpp

10.18 geodesy.cpp

См. документацию.

```
00001 //-----------------------------------------------------------------------------------------------------------
00010
00011 #include <geodesy.h>
00012
00013 namespace SPML
00014 {
00015 namespace Geodesy
00016 {
00017
00018 //-----------------------------------------------------------------------------------------------------------
00019 CEllipsoid::CEllipsoid()
00020 {
00021     a = 0.0;
00022     b = 0.0;
00023     f = 0.0;
00024     invf = 0.0;
00025 }
00026
00027 CEllipsoid::CEllipsoid( std::string ellipsoidName, double semiMajorAxis, double semiMinorAxis, double inverseFlattening,
        bool isInvfDef )
00028 {
```

```
00029     name = ellipsoidName;
00030     a = semiMajorAxis;
00031     invf = inverseFlattening;
00032     if( isInvfDef && ( Compare::IsZeroAbs( inverseFlattening ) || std::isinf( inverseFlattening ) ) ) {
00033         b = semiMajorAxis;
00034         f = 0.0;
00035     } else if ( isInvfDef ) {
00036         b = ( 1.0 - ( 1.0 / inverseFlattening ) ) * semiMajorAxis;
00037         f = 1.0 / inverseFlattening;
00038     } else {
00039         b = semiMinorAxis;
00040         f = 1.0 / inverseFlattening;
00041     }
00042 }
00043
00044 //CEllipsoid::CEllipsoid( std::string ellipsoidName, double semiMajorAxis, double semiMinorAxis, double
      inverseFlattening )
00045 //{
00046 //    name = ellipsoidName;
00047 //    a = semiMajorAxis;
00048 //    invf = inverseFlattening;
00049 //    if( Compare::IsZero( semiMinorAxis ) && !Compare::IsZero( inverseFlattening ) ) { // b = 0, invf != 0
00050 //        b = ( 1.0 - ( 1.0 / inverseFlattening ) ) * semiMajorAxis;
00051 //        f = 1.0 / inverseFlattening;
00052 //    } else if( !Compare::IsZero( semiMinorAxis ) && Compare::IsZero( inverseFlattening ) ) {
00053 //        b = semiMinorAxis;
00054 //        f = 0.0;
00055 //    } else {
00056 //        b = semiMinorAxis;
00057 //        f = 1.0 / inverseFlattening;
00058 //    }
00059 //}
00060 //-----------------------------------------------------------------------------------------------------------------
00061 void GEOtoRAD( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00062     double latStart, double lonStart, double latEnd, double lonEnd, double &d, double &az, double &azEnd )
00063 {
00064     // Параметры эллипсоида:
00065     double a = ellipsoid.A();
00066     double b = ellipsoid.B();
00067     double f = ellipsoid.F();
00068
00069     // По умолчанию Радианы:
00070     double _latStart = latStart;
00071     double _lonStart = lonStart;
00072     double _latEnd = latEnd;
00073     double _lonEnd = lonEnd;
00074
00075     // При необходимости переведем входные данные в Радианы:
00076     switch( angleUnit ) {
00077         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
00078         case( Units::TAngleUnit::AU_Degree ):
00079         {
00080             _latStart *= Convert::DgToRdD;
00081             _lonStart *= Convert::DgToRdD;
00082             _latEnd *= Convert::DgToRdD;
00083             _lonEnd *= Convert::DgToRdD;
00084             break;
00085         }
00086         default:
00087             assert( false );
00088     }
00089     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
00090
00091     if( Compare::AreEqualAbs( a, b ) ) { // При расчете на сфере используем упрощенные формулы
00092         // Azimuth
00093         double fact1, fact2, fact3;
00094         fact1 = std::cos( _latEnd ) * std::sin( _lonEnd - _lonStart );
00095         fact2 = std::cos( _latStart ) * std::sin( _latEnd );
00096         fact3 = std::sin( _latStart ) * std::cos( _latEnd ) * std::cos( _lonEnd - _lonStart );
00097         az = Convert::AngleTo360( std::atan2( fact1, fact2 - fact3 ), Units::AU_Radian ); // [рад] - Прямой азимут в
      начальной точке
00098
00099         // ReverseAzimuth
00100         fact1 = std::cos( _latStart ) * std::sin( _lonEnd - _lonStart );
00101         fact2 = std::cos( _latStart ) * std::sin( _latEnd ) * std::cos( _lonEnd - _lonStart );
00102         fact3 = std::sin( _latStart ) * std::cos( _latEnd );
00103         azEnd = Convert::AngleTo360( ( std::atan2( fact1, fact2 - fact3 ) ), Units::AU_Radian ); // [рад] - Прямой азимут
      в конечной точке
00104
00105         // Distance
00106         double temp1, temp2, temp3;
00107         temp1 = std::sin( _latStart ) * std::sin( _latEnd );
00108         temp2 = std::cos( _latStart ) * std::cos( _latEnd ) * std::cos( _lonEnd - _lonStart );
00109         temp3 = temp1 + temp2;
00110         d = std::acos( temp3 ) * a ; // [м]
00111     } else { // Для эллипсоида используем формулы Винсента
00112         double L = _lonEnd - _lonStart;
```

```
00113
00114       double U1 = std::atan( ( 1.0 - f ) * std::tan( _latStart ) );
00115       double U2 = std::atan( ( 1.0 - f ) * std::tan( _latEnd ) );
00116
00117       double sinU1 = std::sin( U1 );
00118       double cosU1 = std::cos( U1 );
00119       double sinU2 = std::sin( U2 );
00120       double cosU2 = std::cos( U2 );
00121
00122       // eq. 13
00123       double lambda = L;
00124       double lambda_new = 0.0;
00125       int iterLimit = 100;
00126
00127       double sinSigma = 0.0;
00128       double cosSigma = 0.0;
00129       double sigma = 0.0;
00130       double sinAlpha = 0.0;
00131       double cosSqAlpha = 0.0;
00132       double cos2SigmaM = 0.0;
00133       double c = 0.0;
00134       double sinLambda = 0.0;
00135       double cosLambda = 0.0;
00136
00137       do {
00138           sinLambda = std::sin( lambda );
00139           cosLambda = std::cos( lambda );
00140
00141           // eq. 14
00142           sinSigma = std::sqrt( ( ( cosU2 * sinLambda ) * ( cosU2 * sinLambda ) +
00143               ( cosU1 * sinU2 - sinU1 * cosU2 * cosLambda ) * ( cosU1 * sinU2 - sinU1 * cosU2 * cosLambda ) ) );
00144           if( Compare::IsZeroAbs( sinSigma ) ) { // co-incident points
00145               d = 0.0;
00146               az = 0.0;
00147               azEnd = 0.0;
00148               return;
00149           }
00150
00151           // eq. 15
00152           cosSigma = sinU1 * sinU2 + cosU1 * cosU2 * cosLambda;
00153
00154           // eq. 16
00155           sigma = std::atan2( sinSigma, cosSigma );
00156
00157           // eq. 17    Careful!  sin2sigma might be almost 0!
00158           sinAlpha = cosU1 * cosU2 * sinLambda / sinSigma;
00159           cosSqAlpha = 1 - sinAlpha * sinAlpha;
00160
00161           // eq. 18    Careful!  cos2alpha might be almost 0!
00162           cos2SigmaM = cosSigma - 2.0 * sinU1 * sinU2 / cosSqAlpha;
00163
00164           if( std::isnan( cos2SigmaM ) ) {
00165               cos2SigmaM = 0; // equatorial line: cosSqAlpha = 0
00166           }
00167
00168           // eq. 10
00169           c = ( f / 16.0 ) * cosSqAlpha * ( 4.0 + f * ( 4.0 - 3.0 * cosSqAlpha ) );
00170
00171           lambda_new = lambda;
00172
00173           // eq. 11 (modified)
00174           lambda = L + ( 1.0 - c ) * f * sinAlpha *
00175               ( sigma + c * sinSigma * ( cos2SigmaM + c * cosSigma * ( -1.0 + 2.0 * cos2SigmaM * cos2SigmaM ) ) );
00176
00177       } while( std::abs( ( lambda - lambda_new ) / lambda ) > 1.0e-15 && --iterLimit > 0 ); // see how much
      improvement we got
00178
00179       double uSq = cosSqAlpha * ( a * a - b * b ) / ( b * b );
00180
00181       // eq. 3
00182       double A = 1 + uSq / 16384.0 * ( 4096.0 + uSq * ( -768.0 + uSq * ( 320.0 - 175.0 * uSq ) ) );
00183
00184       // eq. 4
00185       double B = uSq / 1024.0 * ( 256.0 + uSq * ( -128.0 + uSq * ( 74.0 - 47.0 * uSq ) ) );
00186
00187       // eq. 6
00188       double deltaSigma = B * sinSigma *
00189           ( cos2SigmaM + ( B / 4.0 ) * ( cosSigma * ( -1.0 + 2.0 * cos2SigmaM * cos2SigmaM ) -
00190           ( B / 6.0 ) * cos2SigmaM * ( -3.0 + 4.0 * sinSigma * sinSigma ) * ( -3.0 + 4.0 * cos2SigmaM * cos2SigmaM ) )
      );
00191
00192       // eq. 19
00193       d = b * A * ( sigma - deltaSigma ); // [m]
00194
00195       // eq. 20
00196       az = Convert::AngleTo360( std::atan2( ( cosU2 * sinLambda ),
00197           ( cosU1 * sinU2 - sinU1 * cosU2 * cosLambda ) ), Units::TAngleUnit::AU_Radian ); // Прямой азимут в
```

```
    начальной точке, [рад]
00198
00199         // eq. 21
00200         azEnd = Convert::AngleTo360( std::atan2( ( cosU1 * sinLambda ), ( -sinU1 * cosU2 + cosU1 * sinU2 * cosLambda
    ) ), Units::TAngleUnit::AU_Radian ); // Прямой азимут в конечной точке, [рад]
00201     }
00202     // az, azEnd, d сейчас в радианах и метрах соответственно
00203
00204     // Проверим, нужен ли перевод:
00205     switch( angleUnit ) {
00206         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
00207         case( Units::TAngleUnit::AU_Degree ):
00208         {
00209             az *= Convert::RdToDgD;
00210             azEnd *= Convert::RdToDgD;
00211             break;
00212         }
00213         default:
00214             assert( false );
00215     }
00216     switch( rangeUnit ) {
00217         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00218         case( Units::TRangeUnit::RU_Kilometer):
00219         {
00220             d *= 0.001;
00221             break;
00222         }
00223         default:
00224             assert( false );
00225     }
00226     return;
00227 }
00228
00229 RAD GEOtoRAD(const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00230     const Geographic &start, const Geographic &end )
00231 {
00232     double d, az, azEnd;
00233     GEOtoRAD( ellipsoid, rangeUnit, angleUnit, start.Lat, start.Lon, end.Lat, end.Lon, d, az, azEnd );
00234     return RAD( d, az, azEnd );
00235 }
00236
00237 //-----------------------------------------------------------------------------------------------------------
00238 void RADtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00239     double latStart, double lonStart, double d, double az, double &latEnd, double &lonEnd, double &azEnd )
00240 {
00241     // Параметры эллипсоида:
00242     double a = ellipsoid.A();
00243     double b = ellipsoid.B();
00244     double f = ellipsoid.F();
00245
00246     // по умолчанию Метры-Радианы:
00247     double _latStart = latStart;    // [рад]
00248     double _lonStart = lonStart;    // [рад]
00249     double _d = d;                  // [м]
00250     double _az = az;                // [рад]
00251
00252     // При необходимости переведем в Радианы-Метры:
00253     switch( angleUnit ) {
00254         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
00255         case( Units::TAngleUnit::AU_Degree ):
00256         {
00257             _latStart *= Convert::DgToRdD;
00258             _lonStart *= Convert::DgToRdD;
00259             _az *= Convert::DgToRdD;
00260             break;
00261         }
00262         default:
00263             assert( false );
00264     }
00265     switch( rangeUnit ) {
00266         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00267         case( Units::TRangeUnit::RU_Kilometer):
00268         {
00269             _d *= 1000.0;
00270             break;
00271         }
00272         default:
00273             assert( false );
00274     }
00275     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
00276
00277     if( Compare::AreEqualAbs(a, b) ) { // При расчете на сфере используем упрощенные формулы
00278         _d = _d / a; // Нормирование
00279         // latitude
00280         double temp1, temp2, temp3;
00281         temp1 = std::sin( _latStart ) * std::cos( _d );
00282         temp2 = std::cos( _latStart ) * std::sin( _d ) * std::cos( _az );
```

```
00283         latEnd = std::asin( temp1 + temp2 ); // [рад]
00284
00285         // longitude
00286         temp1 = std::sin( _d ) * std::sin( _az );
00287         temp2 = std::cos( _latStart ) * std::cos( _d );
00288         temp3 = std::sin( _latStart ) * std::sin( _d ) * std::cos( _az );
00289         lonEnd = _lonStart + std::atan2( temp1, temp2 - temp3 ); // [рад]
00290
00291         // final bearing
00292         temp1 = std::cos( _latStart ) * std::sin( _az );
00293         temp2 = std::cos( _latStart ) * std::cos( _d ) * std::cos( _az );
00294         temp3 = std::sin( _latStart ) * std::sin( _d );
00295         azEnd = Convert::AngleTo360( std::atan2( temp1, temp2 - temp3 ), Units::TAngleUnit::AU_Radian ); // [рад] -
    Прямой азимут в конечной точке
00296     } else { // Для эллипсоида используем формулы Винсента
00297         double cosAlpha1 = std::cos( _az );
00298         double sinAlpha1 = std::sin( _az );
00299         double s = _d; // distance [m]
00300         double tanU1 = ( 1.0 - f ) * std::tan( _latStart );
00301         double cosU1 = 1.0 / std::sqrt( ( 1.0 + tanU1 * tanU1 ) );
00302         double sinU1 = tanU1 * cosU1;
00303
00304         // eq. 1
00305         double sigma1 = std::atan2( tanU1, cosAlpha1 );
00306
00307         // eq. 2
00308         double sinAlpha = cosU1 * sinAlpha1;
00309         double cosSqAlpha = 1 - sinAlpha * sinAlpha;
00310         double uSq = cosSqAlpha * ( a * a - b * b ) / ( b * b );
00311
00312         // eq. 3
00313         double A = 1.0 + ( uSq / 16384.0 ) * ( 4096.0 + uSq * ( -768.0 + uSq * ( 320.0 - 175.0 * uSq ) ) );
00314
00315         // eq. 4
00316         double B = ( uSq / 1024.0 ) * ( 256.0 + uSq * ( -128.0 + uSq * ( 74.0 - 47.0 * uSq ) ) );
00317
00318         // iterate until there is a negligible change in sigma
00319         double sOverbA = s / ( b * A );
00320         double sigma = sOverbA;
00321         double prevSigma = sOverbA;
00322         double cos2SigmaM = 0.0;
00323         double sinSigma = 0.0;
00324         double cosSigma = 0.0;
00325         double deltaSigma = 0.0;
00326
00327         int iterations = 0;
00328
00329         while( true ) {
00330             // eq. 5
00331             cos2SigmaM = std::cos( 2.0 * sigma1 + sigma );
00332             sinSigma = std::sin( sigma );
00333             cosSigma = std::cos( sigma );
00334
00335             // eq. 6
00336             deltaSigma = B * sinSigma * ( cos2SigmaM +
00337                 ( B / 4.0 ) * ( cosSigma * ( -1.0 + 2.0 * cos2SigmaM * cos2SigmaM ) -
00338                 ( B / 6.0 ) * cos2SigmaM * ( -3.0 + 4.0 * sinSigma * sinSigma ) * ( -3.0 + 4.0 * cos2SigmaM * cos2SigmaM )
    ) );
00339
00340             // eq. 7
00341             sigma = sOverbA + deltaSigma;
00342
00343             // break after converging to tolerance
00344             if( std::abs( sigma - prevSigma ) < 1.0e-15 || std::isnan( std::abs( sigma - prevSigma ) ) ) {
00345                 break;
00346             }
00347             prevSigma = sigma;
00348
00349             iterations++;
00350             if( iterations > 1000 ) {
00351                 break;
00352             }
00353         }
00354         cos2SigmaM = std::cos( 2.0 * sigma1 + sigma );
00355         sinSigma = std::sin( sigma );
00356         cosSigma = std::cos( sigma );
00357
00358         double tmp = sinU1 * sinSigma - cosU1 * cosSigma * cosAlpha1;
00359
00360         // eq. 8
00361         latEnd = std::atan2( sinU1 * cosSigma + cosU1 * sinSigma * cosAlpha1,
00362             ( 1.0 - f ) * std::sqrt( ( sinAlpha * sinAlpha + tmp * tmp) ) ); // [рад]
00363
00364         // eq. 9
00365         double lambda = std::atan2( ( sinSigma * sinAlpha1 ), ( cosU1 * cosSigma - sinU1 * sinSigma * cosAlpha1 ) );
00366
00367         // eq. 10
```

```
00368         double c = ( f / 16.0 ) * cosSqAlpha * ( 4.0 + f * ( 4.0 - 3.0 * cosSqAlpha ) );
00369
00370         // eq. 11
00371         double L = lambda - ( 1.0 - c ) * f * sinAlpha * ( sigma + c * sinSigma *
00372             ( cos2SigmaM + c * cosSigma * ( -1.0 + 2.0 * cos2SigmaM * cos2SigmaM ) ) );
00373
00374         //double phi = ( _lonStart + L + 3 * PI ) % ( 2 * PI ) - PI;    // to -180.. 180 original!
00375         //lonEnd = ( _lonStart + L ) * RdToDgD;    // [град] my
00376         lonEnd = _lonStart + L;    // [рад] my
00377
00378         // eq. 12
00379         double alpha2 = std::atan2( sinAlpha, -tmp ); // final bearing, if required
00380         azEnd = Convert::AngleTo360( alpha2, Units::TAngleUnit::AU_Radian ); // Прямой азимут в конечной точке,
    [рад]
00381     }
00382     // latEnd, lonEnd, azEnd сейчас в радианах
00383
00384     // Проверим, нужен ли перевод:
00385     switch( angleUnit ) {
00386         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
00387         case( Units::TAngleUnit::AU_Degree ):
00388         {
00389             latEnd *= Convert::RdToDgD;
00390             lonEnd *= Convert::RdToDgD;
00391             azEnd *= Convert::RdToDgD;
00392             break;
00393         }
00394         default:
00395             assert( false );
00396     }
00397     return;
00398 }
00399
00400 Geographic RADtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
    &angleUnit,
00401     const Geographic &start, const RAD &rad, double &azEnd )
00402 {
00403     double latEnd, lonEnd;
00404     RADtoGEO( ellipsoid, rangeUnit, angleUnit,
00405         start.Lat, start.Lon, rad.R, rad.Az, latEnd, lonEnd, azEnd );
00406     return Geographic( latEnd, lonEnd );
00407 }
00408 //-----------------------------------------------------------------------------------------------------------------
00409 void GEOtoECEF( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
    &angleUnit,
00410     double lat, double lon, double h, double &x, double &y, double &z )
00411 {
00412     // Параметры эллипсоида:
00413     double a = ellipsoid.A();
00414 //    double b = ellipsoid.B();
00415
00416     // по умолчанию Метры-Радианы:
00417     double _lat = lat; // [рад]
00418     double _lon = lon; // [рад]
00419     double _h = h;     // [м]
00420
00421     // При необходимости переведем в Радианы-Метры:
00422     switch( angleUnit ) {
00423         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
00424         case( Units::TAngleUnit::AU_Degree ):
00425         {
00426             _lat *= Convert::DgToRdD;
00427             _lon *= Convert::DgToRdD;
00428             break;
00429         }
00430         default:
00431             assert( false );
00432     }
00433     switch( rangeUnit ) {
00434         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00435         case( Units::TRangeUnit::RU_Kilometer):
00436         {
00437             _h *= 1000.0;
00438             break;
00439         }
00440         default:
00441             assert( false );
00442     }
00443     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
00444     assert( !Compare::IsZeroAbs( a * a ) );
00445 //    double es = 1.0 - ( ( b * b ) / ( a * a ) ); // e^2
00446     double es = ellipsoid.EccentricityFirstSquared();
00447
00448     double sinLat = std::sin( _lat );
00449     double cosLat = std::cos( _lat );
00450     double sinLon = std::sin( _lon );
00451     double cosLon = std::cos( _lon );
```

```
00452
00453     double arg = 1.0 - ( es * ( sinLat * sinLat ) );
00454     assert( arg > 0 );
00455     double v = a / std::sqrt( arg );
00456
00457     x = ( v + _h ) * cosLat * cosLon; // [м]
00458     y = ( v + _h ) * cosLat * sinLon; // [м]
00459     z = ( v * ( 1.0 - es ) + _h ) * std::sin( _lat );    // [м]
00460
00461     // x, y, x сейчас в метрах
00462
00463     // Проверим, нужен ли перевод:
00464     switch( rangeUnit ) {
00465         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00466         case( Units::TRangeUnit::RU_Kilometer ):
00467         {
00468             x /= 1000.0;
00469             y /= 1000.0;
00470             z /= 1000.0;
00471             break;
00472         }
00473         default:
00474             assert( false );
00475     }
00476 }
00477
00478 XYZ GEOtoECEF( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00479     const Geodetic point )
00480 {
00481     double x, y, z;
00482     GEOtoECEF( ellipsoid, rangeUnit, angleUnit, point.Lat, point.Lon, point.Height, x, y, z );
00483     return XYZ( x, y, z );
00484 }
00485 //-------------------------------------------------------------------------------------------------------------
00486 /*
00487 // OLD METHOD
00488 void ECEFtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00489     double x, double y, double z, double &lat, double &lon, double &h )
00490 {
00491     // An Improved Algorithm for Geocentric to Geodetic Coordinate Conversion, Ralph Toms, Feb 1996. UCRL-JC-
      123138
00492
00493     static const double AD_C = 1.0026000;   // Toms region 1 constant. ( h_min = -1e5 [m], h_max = 2e6 [m] ) - MOST
       CASES!
00494 //    static const double AD_C = 1.00092592;  // Toms region 2 constant. ( h_min = 2e6 [m], h_max = 6e6 [m] )
00495 //    static const double AD_C = 0.999250297; // Toms region 3 constant. ( h_min = 6e6 [m], h_max = 18e6 [m] )
00496 //    static const double AD_C = 0.997523508; // Toms region 4 constant. ( h_min = 18e5 [m], h_max = 1e9 [m] )
00497     static const double COS_67P5 = 0.38268343236508977; // Cosine of 67.5 degrees
00498
00499     // Параметры эллипсоида:
00500     double a = ellipsoid.A();
00501     double b = ellipsoid.B();
00502
00503 //    double es = 1.0 - ( b * b ) / ( a * a );    // Eccentricity squared : (a^2 - b^2)/a^2
00504 //    double ses = ( a * a ) / ( b * b ) - 1.0;   // Second eccentricity squared : (a^2 - b^2)/b^2
00505     double es = ellipsoid.EccentricityFirstSquared();   // Eccentricity squared : (a^2 - b^2)/a^2
00506     double ses = ellipsoid.EccentricitySecondSquared(); // Second eccentricity squared : (a^2 - b^2)/b^2
00507
00508     bool At_Pole = false; // indicates whether location is in polar region
00509
00510     double _x = x; // [м]
00511     double _y = y; // [м]
00512     double _z = z; // [м]
00513
00514     // При необходимости переведем в Метры:
00515     switch( rangeUnit ) {
00516         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00517         case( Units::TRangeUnit::RU_Kilometer):
00518         {
00519             _x *= 1000.0;
00520             _y *= 1000.0;
00521             _z *= 1000.0;
00522             break;
00523         }
00524         default:
00525             assert( false );
00526     }
00527     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
00528     double _lon = 0;
00529     double _lat = 0;
00530     double _h = 0;
00531
00532     if( !Compare::IsZeroAbs( x ) ) { //if (x != 0.0)
00533         _lon = std::atan2( _y, _x );
00534     } else {
```

```
00535         if( _y > 0 ) {
00536             _lon = Consts::PI_D / 2;
00537         } else if ( _y < 0 ) {
00538             _lon = -Consts::PI_D * 0.5;
00539         } else {
00540             At_Pole = true;
00541             _lon = 0.0;
00542             if( _z > 0.0 ) {            // north pole
00543                 _lat = Consts::PI_D * 0.5;
00544             } else if ( _z < 0.0 ) {    // south pole
00545                 _lat = -Consts::PI_D * 0.5;
00546             } else {                    // center of earth
00547                 //lat = PI_D * 0.5 * RdToDgD; // [град]
00548                 lat = Consts::PI_D * 0.5; // [рад] TODO: Как тут улучшить?
00549                 lon = 0;
00550                 h = -b;
00551                 return;
00552             }
00553         }
00554     }
00555     double W2 = _x * _x + _y * _y; // Square of distance from Z axis
00556     assert( W2 > 0 );
00557     double W = std::sqrt( W2 ); // distance from Z axis
00558     double T0 = _z * AD_C; // initial estimate of vertical component
00559     assert( ( T0 * T0 + W2 ) > 0 );
00560     double S0 = std::sqrt( T0 * T0 + W2 ); //initial estimate of horizontal component
00561     double Sin_B0 = T0 / S0; //std::sin(B0), B0 is estimate of Bowring aux variable
00562     double Cos_B0 = W / S0; //std::cos(B0)
00563     double Sin3_B0 = Sin_B0 * Sin_B0 * Sin_B0;//Math.Pow(Sin_B0, 3);
00564     double T1 = _z + b * ses * Sin3_B0; //corrected estimate of vertical component
00565     double Sum = W - a * es * Cos_B0 * Cos_B0 * Cos_B0; //numerator of std::cos(phi1)
00566     assert( ( T1 * T1 + Sum * Sum ) > 0 );
00567     double S1 = std::sqrt( T1 * T1 + Sum * Sum ); //corrected estimate of horizontal component
00568     double Sin_p1 = T1 / S1; //std::sin(phi1), phi1 is estimated latitude
00569     double Cos_p1 = Sum / S1; //std::cos(phi1)
00570     assert( ( 1.0 - es * Sin_p1 * Sin_p1 ) > 0 );
00571     double Rn = a / std::sqrt( 1.0 - es * Sin_p1 * Sin_p1 ); //Earth radius at location
00572     if( Cos_p1 >= COS_67P5 ) {
00573         _h = W / Cos_p1 - Rn;
00574     } else if ( Cos_p1 <= -COS_67P5 ) {
00575         assert( !Compare::IsZeroAbs( Cos_p1 ) );
00576         _h = W / ( -Cos_p1 ) - Rn;
00577     } else {
00578         assert( !Compare::IsZeroAbs( Sin_p1 ) );
00579         _h = ( _z / Sin_p1 ) + Rn * ( es - 1.0 );
00580     }
00581     if( !At_Pole ) {
00582         assert( !Compare::IsZeroAbs( Cos_p1 ) );
00583         _lat = std::atan2( Sin_p1, Cos_p1 );
00584     }
00585     lat = _lat; // [рад]
00586     lon = _lon; // [рад]
00587     h = _h;     // [м]
00588
00589     // LLH сейчас в радианах и метрах соответственно
00590
00591     // Проверим, нужен ли перевод:
00592     switch( angleUnit ) {
00593         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
00594         case( Units::TAngleUnit::AU_Degree ):
00595         {
00596             lat *= Convert::RdToDgD;
00597             lon *= Convert::RdToDgD;
00598             break;
00599         }
00600         default:
00601             assert( false );
00602     }
00603     switch( rangeUnit ) {
00604         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00605         case( Units::TRangeUnit::RU_Kilometer ):
00606         {
00607             h /= 1000.0;
00608             break;
00609         }
00610         default:
00611             assert( false );
00612     }
00613 }
00614 */
00615
00616 void ECEFtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00617     double x, double y, double z, double &lat, double &lon, double &h )
00618 {
00619     // Olson, D. K. (1996). Converting Earth-Centered, Earth-Fixed Coordinates to Geodetic Coordinates. IEEE
        Transactions on Aerospace and Electronic Systems, 32(1), 473–476. https://doi.org/10.1109/7.481290
```

```
00620
00621     // Параметры эллипсоида:
00622     double _a = ellipsoid.A();
00623     double _es = ellipsoid.EccentricityFirstSquared();   // Eccentricity squared : (a^2 - b^2)/a^2
00624     const double _a1 = _a * _es;
00625     const double _a2 = _a1 * _a1;
00626     const double _a3 = _a1 * _es / 2.0;
00627     const double _a4 = 2.5 * _a2;
00628     const double _a5 = _a1 + _a3;
00629     const double _a6 = 1.0 - _es;
00630
00631     // wgs-84
00632 //    double _a = 6378137.0;
00633 //    double _es = 6.6943799901377997e-3;
00634 //    double _a1 = 4.2697672707157535e+4;
00635 //    double _a2 = 1.8230912546075455e+9;
00636 //    double _a3 = 1.4291722289812413e+2;
00637 //    double _a4 = 4.5577281365188637e+9;
00638 //    double _a5 = 4.2840589930055659e+4;
00639 //    double _a6 = 9.9330562000986220e-1;
00640
00641     double _x = x;
00642     double _y = y;
00643     double _z = z;
00644
00645     // При необходимости переведем в Метры:
00646     switch( rangeUnit ) {
00647         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00648         case( Units::TRangeUnit::RU_Kilometer):
00649         {
00650             _x *= 1000.0;
00651             _y *= 1000.0;
00652             _z *= 1000.0;
00653             break;
00654         }
00655         default:
00656             assert( false );
00657     }
00658     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
00659
00660     double _lon = 0;
00661     double _lat = 0;
00662     double _h = 0;
00663
00664     double _zp, _w2, _w, _z2, _r2, _r, _s2, _c2, _s, _c, _ss, _g, _rg, _rf, _u, _v, _m, _f, _p;// _rm;
00665
00666     _zp = std::abs( _z );
00667     _w2 = ( _x * _x ) + ( _y * _y );
00668     _w = std::sqrt( _w2 );
00669     _z2 = _z * _z;
00670     _r2 = _w2 + _z2;
00671     _r = std::sqrt( _r2 );
00672 //    if( _r < 100000.0 ) {
00673 //        lat = 0.;
00674 //        lon = 0.;
00675 //        h = -1.e7;
00676 //        return;
00677 //    }
00678     _lon = std::atan2( _y, _x );
00679     _s2 = _z2 / _r2;
00680     _c2 = _w2 / _r2;
00681     _u = _a2 / _r;
00682     _v = _a3 - _a4 / _r;
00683     if( _c2 > 0.3 ) {
00684         _s = ( _zp / _r ) * ( 1.0 + _c2 * ( _a1 + _u + _s2 * _v ) / _r );
00685         _lat = std::asin( _s );//Lat
00686         _ss = _s * _s;
00687         _c = std::sqrt( 1.0 - _ss );
00688     } else {
00689         _c = ( _w / _r ) * ( 1.0 - _s2 * ( _a5 - _u - _c2 * _v ) / _r );
00690         _lat = std::acos( _c );      //Lat
00691         _ss = 1.0 - ( _c * _c );
00692         _s = std::sqrt( _ss );
00693     }
00694     _g = 1.0 - ( _es * _ss );
00695     _rg = _a / std::sqrt( _g );
00696     _rf = _a6 * _rg;
00697     _u = _w - _rg * _c;
00698     _v = _zp - _rf * _s;
00699     _f = _c * _u + _s * _v;
00700     _m = _c * _v - _s * _u;
00701     _p = _m / ( _rf / _g + _f );
00702     _lat += _p; // Lat
00703     _h = _f + _m * _p / 2; // Height
00704     if( _z < 0.0 ){
00705         _lat = -_lat; //Lat
00706     }
```

```
00707     lat = _lat;
00708     lon = _lon;
00709     h = _h;
00710
00711     // Проверим, нужен ли перевод:
00712     switch( angleUnit ) {
00713         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
00714         case( Units::TAngleUnit::AU_Degree ):
00715         {
00716             lat *= Convert::RdToDgD;
00717             lon *= Convert::RdToDgD;
00718             break;
00719         }
00720         default:
00721             assert( false );
00722     }
00723     switch( rangeUnit ) {
00724         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00725         case( Units::TRangeUnit::RU_Kilometer ):
00726         {
00727             h /= 1000.0;
00728             break;
00729         }
00730         default:
00731             assert( false );
00732     }
00733 }
00734
00735 Geodetic ECEFtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00736     XYZ &point )
00737 {
00738     double lat, lon, h;
00739     ECEFtoGEO( ellipsoid, rangeUnit, angleUnit, point.X, point.Y, point.Z, lat, lon, h );
00740     return Geodetic( lat, lon, h );
00741 }
00742 //---------------------------------------------------------------------------------------------------------------
00743 double XYZtoDistance( double x1, double y1, double z1, double x2, double y2, double z2 )
00744 {
00745     double res = ( ( ( x1 - x2 ) * ( x1 - x2 ) ) +
00746                  ( ( y1 - y2 ) * ( y1 - y2 ) ) +
00747                  ( ( z1 - z2 ) * ( z1 - z2 ) ) );
00748     assert( res >= 0 );
00749     res = std::sqrt( res );
00750     return res;
00751 }
00752
00753 double XYZtoDistance( const XYZ &point1, const XYZ &point2 )
00754 {
00755     return XYZtoDistance( point1.X, point1.Y, point1.Z, point2.X, point2.Y, point2.Z );
00756 }
00757 //---------------------------------------------------------------------------------------------------------------
00758 void ECEF_offset( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00759     double lat1, double lon1, double h1, double lat2, double lon2, double h2, double &dX, double &dY, double &dZ )
00760 {
00761     // Параметры эллипсоида:
00762     double a = ellipsoid.A();
00763     double b = ellipsoid.B();
00764     //double f = el.F();
00765
00766     // по умолчанию Метры-Радианы:
00767     double _lat1 = lat1;
00768     double _lon1 = lon1;
00769     double _h1 = h1;
00770     double _lat2 = lat2;
00771     double _lon2 = lon2;
00772     double _h2 = h2;
00773
00774     // При необходимости переведем в Радианы-Метры:
00775     switch( angleUnit ) {
00776         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
00777         case( Units::TAngleUnit::AU_Degree ):
00778         {
00779             _lat1 *= Convert::DgToRdD;
00780             _lon1 *= Convert::DgToRdD;
00781             _lat2 *= Convert::DgToRdD;
00782             _lon2 *= Convert::DgToRdD;
00783             break;
00784         }
00785         default:
00786             assert( false );
00787     }
00788     switch( rangeUnit ) {
00789         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00790         case( Units::TRangeUnit::RU_Kilometer):
00791         {
```

```
00792             _h1 *= 1000.0;
00793             _h2 *= 1000.0;
00794             break;
00795         }
00796         default:
00797             assert( false );
00798     }
00799     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
00800
00801     double s1 = std::sin( lat1 );
00802     double c1 = std::cos( lat1 );
00803
00804     double s2 = std::sin( lat2 );
00805     double c2 = std::cos( lat2 );
00806
00807     double p1 = c1 * std::cos( lon1 );
00808     double p2 = c2 * std::cos( lon2 );
00809
00810     double q1 = c1 * std::sin( lon1 );
00811     double q2 = c2 * std::sin( lon2 );
00812
00813     if( Compare::AreEqualAbs( a, b ) ) { // Сфера
00814         dX = a * ( p2 - p1 ) + ( h2 * p2 - h1 * p1 );
00815         dY = a * ( q2 - q1 ) + ( h2 * q2 - h1 * q1 );
00816         dZ = a * ( s2 - s1 ) + ( h2 * s2 - h1 * s1 );
00817     } else { // Эллипсоид
00818         double e2 = std::pow( ellipsoid.EccentricityFirst(), 2 ); // Квадрат 1-го эксцентриситета эллипсоида
00819
00820         double w1 = 1.0 / std::sqrt( 1.0 - e2 * s1 * s1 );
00821         double w2 = 1.0 / std::sqrt( 1.0 - e2 * s2 * s2 );
00822
00823         dX = a * ( p2 * w2 - p1 * w1 ) + ( h2 * p2 - h1 * p1 );
00824         dY = a * ( q2 * w2 - q1 * w1 ) + ( h2 * q2 - h1 * q1 );
00825         dZ = ( 1.0 - e2 ) * a * ( s2 * w2 - s1 * w1 ) + ( h2 * s2 - h1 * s1 );
00826     }
00827     // dX dY dZ сейчас в метрах
00828
00829     // Проверим, нужен ли перевод:
00830     switch( rangeUnit ) {
00831         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00832         case( Units::TRangeUnit::RU_Kilometer ):
00833         {
00834             dX *= 0.001;
00835             dY *= 0.001;
00836             dZ *= 0.001;
00837             break;
00838         }
00839         default:
00840             assert( false );
00841     }
00842 }
00843
00844 XYZ ECEF_offset( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00845     const Geodetic &point1, const Geodetic &point2 )
00846 {
00847     double x, y, z;
00848     ECEF_offset( ellipsoid, rangeUnit, angleUnit,
00849         point1.Lat, point1.Lon, point1.Height, point2.Lat, point2.Lon, point2.Height, x, y, z );
00850     return XYZ( x, y, z );
00851 }
00852 //-----------------------------------------------------------------------------------------------------------
00853 void ECEFtoENU( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
00854     double x, double y, double z, double lat, double lon, double h, double &xEast, double &yNorth, double &zUp )
00855 {
00856     // по умолчанию Метры-Радианы:
00857     double _lat = lat;
00858     double _lon = lon;
00859     double _h = h;
00860     double _x = x;
00861     double _y = y;
00862     double _z = z;
00863     double _xr, _yr, _zr; // Reference point
00864
00865     // При необходимости переведем в Радианы-Метры:
00866     switch( angleUnit ) {
00867         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
00868         case( Units::TAngleUnit::AU_Degree ):
00869         {
00870             _lat *= Convert::DgToRdD;
00871             _lon *= Convert::DgToRdD;
00872             break;
00873         }
00874         default:
00875             assert( false );
00876     }
```

```
00877     switch( rangeUnit ) {
00878         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00879         case( Units::TRangeUnit::RU_Kilometer):
00880         {
00881             _h *= 1000.0;
00882             _x *= 1000.0;
00883             _y *= 1000.0;
00884             _z *= 1000.0;
00885             break;
00886         }
00887         default:
00888             assert( false );
00889     }
00890     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
00891
00892     GEOtoECEF( ellipsoid, Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _lat, _lon, _h, _xr, _yr,
      _zr ); // Получены ECEF координаты опорной точки
00893
00894     double cosPhi = std::cos( _lat );
00895     double sinPhi = std::sin( _lat );
00896     double cosLambda = std::cos( _lon );
00897     double sinLambda = std::sin( _lon );
00898
00899     double _dx = _x - _xr;
00900     double _dy = _y - _yr;
00901     double _dz = _z - _zr;
00902
00903     double t = ( cosLambda * _dx ) + ( sinLambda * _dy );
00904     xEast = ( -sinLambda * _dx ) + ( cosLambda * _dy );
00905     yNorth = ( -sinPhi * t ) + ( cosPhi * _dz );
00906     zUp = ( cosPhi * t ) + ( sinPhi * _dz );
00907     // xEast yNorth zUp сейчас в метрах
00908
00909     // Проверим, нужен ли перевод:
00910     switch( rangeUnit ) {
00911         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00912         case( Units::TRangeUnit::RU_Kilometer ):
00913         {
00914             xEast *= 0.001;
00915             yNorth *= 0.001;
00916             zUp *= 0.001;
00917             break;
00918         }
00919         default:
00920             assert( false );
00921     }
00922 }
00923
00924 ENU ECEFtoENU( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
00925     const XYZ &ecef, const Geodetic &point )
00926 {
00927     double e, n, u;
00928     ECEFtoENU( ellipsoid, rangeUnit, angleUnit, ecef.X, ecef.Y, ecef.Z, point.Lat, point.Lon, point.Height, e, n, u );
00929     return ENU( e, n, u );
00930 }
00931 //-----------------------------------------------------------------------------------------------------------
00932 void ECEFtoENUV( const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00933     double dX, double dY, double dZ, double lat, double lon, double &xEast, double &yNorth, double &zUp )
00934 {
00935     // по умолчанию Метры-Радианы:
00936     double _lat = lat;
00937     double _lon = lon;
00938     double _dX = dX;
00939     double _dY = dY;
00940     double _dZ = dZ;
00941
00942     // При необходимости переведем в Радианы-Метры:
00943     switch( angleUnit ) {
00944         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
00945         case( Units::TAngleUnit::AU_Degree ):
00946         {
00947             _lat *= Convert::DgToRdD;
00948             _lon *= Convert::DgToRdD;
00949             break;
00950         }
00951         default:
00952             assert( false );
00953     }
00954     switch( rangeUnit ) {
00955         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00956         case( Units::TRangeUnit::RU_Kilometer):
00957         {
00958             _dX *= 1000.0;
00959             _dY *= 1000.0;
00960             _dZ *= 1000.0;
00961             break;
```

```
00962         }
00963         default:
00964             assert( false );
00965     }
00966     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
00967
00968     double cosPhi = std::cos( _lat );
00969     double sinPhi = std::sin( _lat );
00970     double cosLambda = std::cos( _lon );
00971     double sinLambda = std::sin( _lon );
00972
00973     double t = ( cosLambda * _dX ) + ( sinLambda * _dY );
00974     xEast = ( -sinLambda * _dX ) + ( cosLambda * _dY );
00975
00976     zUp    =  ( cosPhi * t ) + ( sinPhi * _dZ );
00977     yNorth =  ( -sinPhi * t ) + ( cosPhi * _dZ );
00978
00979     // xEast yNorth zUp сейчас в метрах
00980
00981     // Проверим, нужен ли перевод:
00982     switch( rangeUnit ) {
00983         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
00984         case( Units::TRangeUnit::RU_Kilometer ):
00985         {
00986             xEast *= 0.001;
00987             yNorth *= 0.001;
00988             zUp *= 0.001;
00989             break;
00990         }
00991         default:
00992             assert( false );
00993     }
00994 }
00995
00996 ENU ECEFtoENUV( const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
00997     const XYZ &shift, const Geographic &point )
00998 {
00999     double e, n, u;
01000     ECEFtoENUV( rangeUnit, angleUnit, shift.X, shift.Y, shift.Z, point.Lat, point.Lon, e, n, u );
01001     return ENU( e, n, u );
01002 }
01003 //---------------------------------------------------------------------------------------------------------------
01004 void ENUtoECEF( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
01005     double e, double n, double u, double lat, double lon, double h, double &x, double &y, double &z )
01006 {
01007     // по умолчанию Метры-Радианы:
01008     double _lat = lat;
01009     double _lon = lon;
01010     double _h = h;
01011     double _e = e;
01012     double _n = n;
01013     double _u = u;
01014     double _xr, _yr, _zr; // Reference point
01015
01016     // При необходимости переведем в Радианы-Метры:
01017     switch( angleUnit ) {
01018         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01019         case( Units::TAngleUnit::AU_Degree ):
01020         {
01021             _lat *= Convert::DgToRdD;
01022             _lon *= Convert::DgToRdD;
01023             break;
01024         }
01025         default:
01026             assert( false );
01027     }
01028     switch( rangeUnit ) {
01029         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01030         case( Units::TRangeUnit::RU_Kilometer):
01031         {
01032             _h *= 1000.0;
01033             _e *= 1000.0;
01034             _n *= 1000.0;
01035             _u *= 1000.0;
01036             break;
01037         }
01038         default:
01039             assert( false );
01040     }
01041     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
01042
01043     GEOtoECEF( ellipsoid, Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian,
01044         _lat, _lon, _h, _xr, _yr, _zr ); // Получены ECEF координаты опорной точки
01045
01046     double cosPhi = std::cos( _lat );
01047     double sinPhi = std::sin( _lat );
```

```
01048     double cosLambda = std::cos( _lon );
01049     double sinLambda = std::sin( _lon );
01050
01051     x = -sinLambda * _e - sinPhi * cosLambda * _n + cosPhi * cosLambda * _u + _xr;
01052     y = cosLambda * _e - sinPhi * sinLambda * _n + cosPhi * sinLambda * _u + _yr;
01053     z = cosPhi * _n + sinPhi * _u + _zr;
01054
01055     // Проверим, нужен ли перевод:
01056     switch( rangeUnit ) {
01057         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01058         case( Units::TRangeUnit::RU_Kilometer ):
01059         {
01060             x *= 0.001;
01061             y *= 0.001;
01062             z *= 0.001;
01063             break;
01064         }
01065         default:
01066             assert( false );
01067     }
01068 }
01069
01070 XYZ ENUtoECEF( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
01071     const ENU &enu, const Geodetic &point )
01072 {
01073     double x, y, z;
01074     ENUtoECEF( ellipsoid, rangeUnit, angleUnit, enu.E, enu.N, enu.U, point.Lat, point.Lon, point.Height, x, y, z );
01075     return XYZ( x, y, z );
01076 }
01077
01078 //-----------------------------------------------------------------------------------------------------------
01079 void ENUtoAER( const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
01080     double xEast, double yNorth, double zUp, double &az, double &elev, double &slantRange )
01081 {
01082     // по умолчанию Метры-Радианы:
01083     double _xEast = xEast;
01084     double _yNorth = yNorth;
01085     double _zUp = zUp;
01086
01087     // Проверим, нужен ли перевод:
01088     switch( rangeUnit ) {
01089         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01090         case( Units::TRangeUnit::RU_Kilometer ):
01091         {
01092             _xEast *=  1000.0;
01093             _yNorth *= 1000.0;
01094             _zUp *= 1000.0;
01095             break;
01096         }
01097         default:
01098             assert( false );
01099     }
01100     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
01101
01102 //    r = std::sqrt( ( _xEast * _xEast ) + ( _yNorth * _yNorth ) ); // dangerous
01103     double r = std::hypot( _xEast, _yNorth ); // C++11 style
01104
01105 //    slantRange = sqrt( ( r * r ) + ( _zUp * _zUp ) ); // dangerous
01106     slantRange = std::hypot( r, _zUp ); // C++11 style
01107     elev = std::atan2( _zUp, r );
01108     az = Convert::AngleTo360( std::atan2( _xEast, _yNorth ), Units::TAngleUnit::AU_Radian );
01109
01110     // Проверим, нужен ли перевод:
01111     switch( rangeUnit ) {
01112         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01113         case( Units::TRangeUnit::RU_Kilometer ):
01114         {
01115             slantRange *= 0.001;
01116             break;
01117         }
01118         default:
01119             assert( false );
01120     }
01121     switch( angleUnit ) {
01122         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01123         case( Units::TAngleUnit::AU_Degree ):
01124         {
01125             az *= Convert::RdToDgD;
01126             elev *= Convert::RdToDgD;
01127         }
01128     }
01129 }
01130
01131 AER ENUtoAER( const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const ENU &point )
01132 {
01133     double a, e, r;
```

```
01134     ENUtoAER( rangeUnit, angleUnit, point.E, point.N, point.U, a, e, r );
01135     return AER( a, e, r );
01136 }
01137 //-----------------------------------------------------------------------------------------------------------
01138 void AERtoENU( const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
01139     double az, double elev, double slantRange, double &xEast, double &yNorth, double &zUp )
01140 {
01141     double _az = az;
01142     double _elev = elev;
01143     double _slantRange = slantRange;
01144
01145     // При необходимости переведем в Радианы-Метры:
01146     switch( angleUnit ) {
01147         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01148         case( Units::TAngleUnit::AU_Degree ):
01149         {
01150             _az *= Convert::DgToRdD;
01151             _elev *= Convert::DgToRdD;
01152             break;
01153         }
01154         default:
01155             assert( false );
01156     }
01157     switch( rangeUnit ) {
01158         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01159         case( Units::TRangeUnit::RU_Kilometer):
01160         {
01161             _slantRange *= 1000.0;
01162             break;
01163         }
01164         default:
01165             assert( false );
01166     }
01167
01168     zUp = _slantRange * std::sin( _elev );
01169     double _r = _slantRange * std::cos( _elev );
01170     xEast = _r * std::sin( _az );
01171     yNorth = _r * std::cos( _az );
01172     // xEast yNorth zUp сейчас в метрах
01173
01174     // Проверим, нужен ли перевод:
01175     switch( rangeUnit ) {
01176         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01177         case( Units::TRangeUnit::RU_Kilometer ):
01178         {
01179             xEast *= 0.001;
01180             yNorth *= 0.001;
01181             zUp *= 0.001;
01182             break;
01183         }
01184         default:
01185             assert( false );
01186     }
01187 }
01188
01189 ENU AERtoENU( const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit, const AER &aer )
01190 {
01191     double e, n, u;
01192     AERtoENU( rangeUnit, angleUnit, aer.A, aer.E, aer.R, e, n, u );
01193     return ENU( e, n, u );
01194 }
01195
01196 //-----------------------------------------------------------------------------------------------------------
01197 void GEOtoENU( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
01198     double lat, double lon, double h, double lat0, double lon0, double h0, double &xEast, double &yNorth, double &zUp )
01199 {
01200     // по умолчанию Метры-Радианы:
01201     double _lat = lat;
01202     double _lon = lon;
01203     double _h = h;
01204     double _lat0 = lat0;
01205     double _lon0 = lon0;
01206     double _h0 = h0;
01207
01208     // При необходимости переведем в Радианы-Метры:
01209     switch( angleUnit ) {
01210         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01211         case( Units::TAngleUnit::AU_Degree ):
01212         {
01213             _lat *= Convert::DgToRdD;
01214             _lon *= Convert::DgToRdD;
01215             _lat0 *= Convert::DgToRdD;
01216             _lon0 *= Convert::DgToRdD;
01217             break;
01218         }
01219         default:
01220             assert( false );
```

```
01221     }
01222     switch( rangeUnit ) {
01223         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01224         case( Units::TRangeUnit::RU_Kilometer):
01225         {
01226             _h *= 1000.0;
01227             _h0 *= 1000.0;
01228             break;
01229         }
01230         default:
01231             assert( false );
01232     }
01233
01234     double _x, _y, _z, _x0, _y0, _z0;
01235     GEOtoECEF( ellipsoid, Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _lat, _lon, _h, _x, _y, _z );
01236     GEOtoECEF( ellipsoid, Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _lat0, _lon0, _h0, _x0,
      _y0, _z0 );
01237
01238     double _dx = _x - _x0;
01239     double _dy = _y - _y0;
01240     double _dz = _z - _z0;
01241
01242     ECEFtoENUV( Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _dx, _dy, _dz, _lat0, _lon0, xEast,
      yNorth, zUp );
01243
01244     // Проверим, нужен ли перевод:
01245     switch( rangeUnit ) {
01246         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01247         case( Units::TRangeUnit::RU_Kilometer ):
01248         {
01249             xEast *= 0.001;
01250             yNorth *= 0.001;
01251             zUp *= 0.001;
01252             break;
01253         }
01254         default:
01255             assert( false );
01256     }
01257 }
01258
01259 ENU GEOtoENU( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
01260     const Geodetic &point, const Geodetic &anchor )
01261 {
01262     double e, n, u;
01263     GEOtoENU( ellipsoid, rangeUnit, angleUnit,
01264         point.Lat, point.Lon, point.Height, anchor.Lat, anchor.Lon, anchor.Height, e, n, u );
01265     return ENU( e, n, u );
01266 }
01267 //-----------------------------------------------------------------------------------------------------------------
01268 void ENUtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
01269     double xEast, double yNorth, double zUp, double lat0, double lon0, double h0, double &lat, double &lon, double &h )
01270 {
01271     // по умолчанию Метры-Радианы:
01272     double _xEast = xEast;
01273     double _yNorth = yNorth;
01274     double _zUp = zUp;
01275     double _lat0 = lat0;
01276     double _lon0 = lon0;
01277     double _h0 = h0;
01278
01279     // При необходимости переведем в Радианы-Метры:
01280     switch( angleUnit ) {
01281         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01282         case( Units::TAngleUnit::AU_Degree ):
01283         {
01284             _lat0 *= Convert::DgToRdD;
01285             _lon0 *= Convert::DgToRdD;
01286             break;
01287         }
01288         default:
01289             assert( false );
01290     }
01291     switch( rangeUnit ) {
01292         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01293         case( Units::TRangeUnit::RU_Kilometer):
01294         {
01295             _xEast *= 1000.0;
01296             _yNorth *= 1000.0;
01297             _zUp *= 1000.0;
01298             _h0 *= 1000.0;
01299             break;
01300         }
01301         default:
01302             assert( false );
01303     }
01304
```

```
01305     double _x, _y, _z;
01306     ENUtoECEF( ellipsoid, Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _xEast, _yNorth, _zUp,
01307         _lat0, _lon0, _h0, _x, _y, _z );
01308     ECEFtoGEO( ellipsoid, Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _x, _y, _z, lat, lon, h );
01309
01310     // Проверим, нужен ли перевод:
01311     switch( angleUnit ) {
01312         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01313         case( Units::TAngleUnit::AU_Degree ):
01314         {
01315             lat *= Convert::RdToDgD;
01316             lon *= Convert::RdToDgD;
01317             break;
01318         }
01319         default:
01320             assert( false );
01321     }
01322     switch( rangeUnit ) {
01323         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01324         case( Units::TRangeUnit::RU_Kilometer):
01325         {
01326             h *= 0.001;
01327             break;
01328         }
01329         default:
01330             assert( false );
01331     }
01332 }
01333
01334 Geodetic ENUtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
01335     const ENU &point, const Geodetic &anchor )
01336 {
01337     double lat, lon, h;
01338     ENUtoGEO( ellipsoid, rangeUnit, angleUnit, point.E, point.N, point.U, anchor.Lat, anchor.Lon, anchor.Height, lat, lon,
        h );
01339     return Geodetic( lat, lon, h );
01340 }
01341
01342 //---------------------------------------------------------------------------------------------------------------
01343 void GEOtoAER( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
01344     double lat1, double lon1, double h1, double lat2, double lon2, double h2, double &az, double &elev, double
        &slantRange )
01345 {
01346     // по умолчанию Метры-Радианы:
01347     double _lat1 = lat1;
01348     double _lon1 = lon1;
01349     double _h1 = h1;
01350     double _lat2 = lat2;
01351     double _lon2 = lon2;
01352     double _h2 = h2;
01353
01354     // При необходимости переведем в Радианы-Метры:
01355     switch( angleUnit ) {
01356         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01357         case( Units::TAngleUnit::AU_Degree ):
01358         {
01359             _lat1 *= Convert::DgToRdD;
01360             _lon1 *= Convert::DgToRdD;
01361             _lat2 *= Convert::DgToRdD;
01362             _lon2 *= Convert::DgToRdD;
01363             break;
01364         }
01365         default:
01366             assert( false );
01367     }
01368     switch( rangeUnit ) {
01369         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01370         case( Units::TRangeUnit::RU_Kilometer):
01371         {
01372             _h1 *= 1000.0;
01373             _h2 *= 1000.0;
01374             break;
01375         }
01376         default:
01377             assert( false );
01378     }
01379     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
01380
01381     double _xEast, _yNorth, _zUp;
01382     GEOtoENU( ellipsoid, Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _lat1, _lon1, _h1, _lat2,
        _lon2, _h2,
01383         _xEast, _yNorth, _zUp );
01384     ENUtoAER( Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _xEast, _yNorth, _zUp, az, elev,
        slantRange );
01385
01386     // Проверим, нужен ли перевод:
```

```
01387     switch( angleUnit ) {
01388         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01389         case( Units::TAngleUnit::AU_Degree ):
01390         {
01391             az *= Convert::RdToDgD;
01392             elev *= Convert::RdToDgD;
01393             break;
01394         }
01395         default:
01396             assert( false );
01397     }
01398     switch( rangeUnit ) {
01399         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01400         case( Units::TRangeUnit::RU_Kilometer):
01401         {
01402             slantRange *= 0.001;
01403             break;
01404         }
01405         default:
01406             assert( false );
01407     }
01408 }
01409
01410 AER GEOtoAER( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
01411     const Geodetic &point1, const Geodetic &point2 )
01412 {
01413     double a, e, r;
01414     GEOtoAER( ellipsoid, rangeUnit, angleUnit,
01415         point1.Lat, point1.Lon, point1.Height, point2.Lat, point2.Lon, point2.Height, a, e, r );
01416     return AER( a, e, r );
01417 }
01418 //---------------------------------------------------------------------------------------------------------------------
01419 void AERtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit &angleUnit,
01420     double az, double elev, double slantRange, double lat0, double lon0, double h0, double &lat, double &lon, double &h )
01421 {
01422     // по умолчанию Метры-Радианы:
01423     double _az = az;
01424     double _elev = elev;
01425     double _slantRange = slantRange;
01426     double _lat0 = lat0;
01427     double _lon0 = lon0;
01428     double _h0 = h0;
01429
01430     // При необходимости переведем в Радианы-Метры:
01431     switch( angleUnit ) {
01432         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01433         case( Units::TAngleUnit::AU_Degree ):
01434         {
01435             _az *= Convert::DgToRdD;
01436             _elev *= Convert::DgToRdD;
01437             _lat0 *= Convert::DgToRdD;
01438             _lon0 *= Convert::DgToRdD;
01439             break;
01440         }
01441         default:
01442             assert( false );
01443     }
01444     switch( rangeUnit ) {
01445         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01446         case( Units::TRangeUnit::RU_Kilometer):
01447         {
01448             _slantRange *= 1000.0;
01449             _h0 *= 1000.0;
01450             break;
01451         }
01452         default:
01453             assert( false );
01454     }
01455     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
01456
01457     double _x, _y, _z;
01458     AERtoECEF( ellipsoid, Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _az, _elev, _slantRange,
01459         _lat0, _lon0, _h0, _x, _y, _z );
01460     ECEFtoGEO( ellipsoid, Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _x, _y, _z, lat, lon, h );
01461
01462     // Проверим, нужен ли перевод:
01463     switch( angleUnit ) {
01464         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01465         case( Units::TAngleUnit::AU_Degree ):
01466         {
01467             lat *= Convert::RdToDgD;
01468             lon *= Convert::RdToDgD;
01469             break;
01470         }
01471         default:
01472             assert( false );
```

```
01473     }
01474     switch( rangeUnit ) {
01475         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01476         case( Units::TRangeUnit::RU_Kilometer):
01477         {
01478             h *= 0.001;
01479             break;
01480         }
01481         default:
01482             assert( false );
01483     }
01484 }
01485
01486 Geodetic AERtoGEO( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
01487     const AER &aer, const Geodetic &anchor )
01488 {
01489     double lat, lon, h;
01490     AERtoGEO( ellipsoid, rangeUnit, angleUnit, aer.A, aer.E, aer.R, anchor.Lat, anchor.Lon, anchor.Height, lat, lon, h );
01491     return Geodetic( lat, lon, h );
01492 }
01493 //-----------------------------------------------------------------------------------------------------------------
01494 void AERtoECEF( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
        &angleUnit,
01495      double az, double elev, double slantRange, double lat0, double lon0, double h0, double &x, double &y, double &z )
01496 {
01497     // по умолчанию Метры-Радианы:
01498     double _az = az;
01499     double _elev = elev;
01500     double _slantRange = slantRange;
01501     double _lat0 = lat0;
01502     double _lon0 = lon0;
01503     double _h0 = h0;
01504
01505     // При необходимости переведем в Радианы-Метры:
01506     switch( angleUnit ) {
01507         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01508         case( Units::TAngleUnit::AU_Degree ):
01509         {
01510             _az *= Convert::DgToRdD;
01511             _elev *= Convert::DgToRdD;
01512             _lat0 *= Convert::DgToRdD;
01513             _lon0 *= Convert::DgToRdD;
01514             break;
01515         }
01516         default:
01517             assert( false );
01518     }
01519     switch( rangeUnit ) {
01520         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01521         case( Units::TRangeUnit::RU_Kilometer):
01522         {
01523             _slantRange *= 1000.0;
01524             _h0 *= 1000.0;
01525             break;
01526         }
01527         default:
01528             assert( false );
01529     }
01530     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
01531
01532     double _x0, _y0, _z0, _e, _n, _u, _dx, _dy, _dz;
01533     GEOtoECEF( ellipsoid, Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _lat0, _lon0, _h0, _x0,
        _y0, _z0 );
01534     AERtoENU( Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _az, _elev, _slantRange, _e, _n, _u );
01535     ENUtoUVW( ellipsoid, Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _e, _n, _u, _lat0, _lon0,
        _dx, _dy, _dz );
01536     // Origin + offset from origin equals position in ECEF
01537     x = _x0 + _dx;
01538     y = _y0 + _dy;
01539     z = _z0 + _dz;
01540
01541     // Проверим, нужен ли перевод:
01542     switch( rangeUnit ) {
01543         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01544         case( Units::TRangeUnit::RU_Kilometer ):
01545         {
01546             x *= 0.001;
01547             y *= 0.001;
01548             z *= 0.001;
01549             break;
01550         }
01551         default:
01552             assert( false );
01553     }
01554 }
01555
```

```
01556 XYZ AERtoECEF( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
01557     const AER &aer, const Geodetic &anchor )
01558 {
01559     double x, y, z;
01560     AERtoECEF( ellipsoid, rangeUnit, angleUnit, aer.A, aer.E, aer.R, anchor.Lat, anchor.Lon, anchor.Height, x, y, z );
01561     return XYZ( x, y, z );
01562 }
01563 //---------------------------------------------------------------------------------------------------------------
01564 void ECEFtoAER( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
01565     double x, double y, double z, double lat0, double lon0, double h0, double &az, double &elev, double &slantRange )
01566 {
01567     // по умолчанию Метры-Радианы:
01568     double _lat0 = lat0;
01569     double _lon0 = lon0;
01570     double _h0 = h0;
01571     double _x = x;
01572     double _y = y;
01573     double _z = z;
01574
01575     // При необходимости переведем в Радианы-Метры:
01576     switch( angleUnit ) {
01577         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01578         case( Units::TAngleUnit::AU_Degree ):
01579         {
01580             _lat0 *= Convert::DgToRdD;
01581             _lon0 *= Convert::DgToRdD;
01582             break;
01583         }
01584         default:
01585             assert( false );
01586     }
01587     switch( rangeUnit ) {
01588         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01589         case( Units::TRangeUnit::RU_Kilometer):
01590         {
01591             _h0 *= 1000.0;
01592             _x *= 1000.0;
01593             _y *= 1000.0;
01594             _z *= 1000.0;
01595             break;
01596         }
01597         default:
01598             assert( false );
01599     }
01600     // Далее в математике используются углы в радианах и дальность в метрах, перевод в нужные единицы у конце
01601
01602     double _e, _n, _u;
01603     ECEFtoENU( ellipsoid, Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _x, _y, _z, _lat0, _lon0,
      _h0,
01604         _e, _n, _u );
01605     ENUtoAER( Units::TRangeUnit::RU_Meter, Units::TAngleUnit::AU_Radian, _e, _n, _u, az, elev, slantRange );
01606
01607     // Проверим, нужен ли перевод:
01608     switch( angleUnit ) {
01609         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01610         case( Units::TAngleUnit::AU_Degree ):
01611         {
01612             az *= Convert::RdToDgD;
01613             elev *= Convert::RdToDgD;
01614             break;
01615         }
01616         default:
01617             assert( false );
01618     }
01619     switch( rangeUnit ) {
01620         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01621         case( Units::TRangeUnit::RU_Kilometer):
01622         {
01623             slantRange *= 0.001;
01624             break;
01625         }
01626         default:
01627             assert( false );
01628     }
01629 }
01630
01631 AER ECEFtoAER(const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
01632     const XYZ &ecef, const Geodetic &anchor )
01633 {
01634     double a, e, r;
01635     ECEFtoAER( ellipsoid, rangeUnit, angleUnit, ecef.X, ecef.Y, ecef.Z, anchor.Lat, anchor.Lon, anchor.Height, a, e, r );
01636     return AER( a, e, r );
01637 }
01638 //---------------------------------------------------------------------------------------------------------------
```

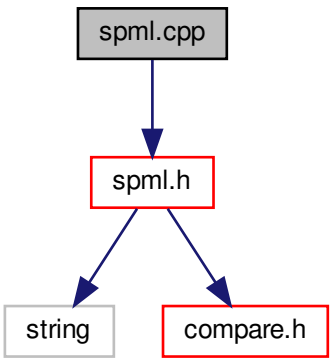
```
01639 void ENUtoUVW( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
01640     double xEast, double yNorth, double zUp, double lat0, double lon0, double &u, double &v, double &w )
01641 {
01642     double _xEast = xEast;
01643     double _yNorth = yNorth;
01644     double _zUp = zUp;
01645     double _lat0 = lat0;
01646     double _lon0 = lon0;
01647
01648     switch( angleUnit ) {
01649         case( Units::TAngleUnit::AU_Radian ): break; // Уже переведено
01650         case( Units::TAngleUnit::AU_Degree ):
01651         {
01652             _lat0 *= Convert::DgToRdD;
01653             _lon0 *= Convert::DgToRdD;
01654             break;
01655         }
01656         default:
01657             assert( false );
01658     }
01659     switch( rangeUnit ) {
01660         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01661         case( Units::TRangeUnit::RU_Kilometer):
01662         {
01663             _xEast *= 1000.0;
01664             _yNorth *= 1000.0;
01665             _zUp *= 1000.0;
01666             break;
01667         }
01668         default:
01669             assert( false );
01670     }
01671
01672     double t = std::cos( _lat0 ) * _zUp - std::sin( _lat0 ) * _yNorth;
01673     w = std::sin( _lat0 ) * _zUp + std::cos( _lat0 ) * _yNorth;
01674     u = std::cos( _lon0 ) * t - std::sin( _lon0 ) * _xEast;
01675     v = std::sin( _lon0 ) * t + std::cos( _lon0 ) * _xEast;
01676
01677     // Проверим, нужен ли перевод:
01678     switch( rangeUnit ) {
01679         case( Units::TRangeUnit::RU_Meter ): break; // Уже переведено
01680         case( Units::TRangeUnit::RU_Kilometer ):
01681         {
01682             w *= 0.001;
01683             u *= 0.001;
01684             v *= 0.001;
01685             break;
01686         }
01687         default:
01688             assert( false );
01689     }
01690 }
01691
01692 UVW ENUtoUVW( const CEllipsoid &ellipsoid, const Units::TRangeUnit &rangeUnit, const Units::TAngleUnit
      &angleUnit,
01693     const ENU &enu, const Geographic &point )
01694 {
01695     double u, v, w;
01696     ENUtoUVW( ellipsoid, rangeUnit, angleUnit, enu.E, enu.N, enu.U, point.Lat, point.Lon, u, v, w );
01697     return UVW( u, v, w );
01698 }
01699 //-----------------------------------------------------------------------------------------------------------
01700 double CosAngleBetweenVectors( double x1, double y1, double z1, double x2, double y2, double z2 )
01701 {
01702     // Исходя из формулы косинуса угла между векторами:
01703     double a1 =  x1 * x2;
01704     double a2 =  y1 * y2;
01705     double a3 =  z1 * z2;
01706     double b1 = std::sqrt( ( x1 * x1 ) + ( y1 * y1 ) + ( z1 * z1 ) );
01707     double b2 = std::sqrt( ( x2 * x2 ) + ( y2 * y2 ) + ( z2 * z2 ) );
01708
01709     double res = ( a1 + a2 + a3 ) / ( b1 * b2 );
01710     return res;
01711 }
01712
01713 double CosAngleBetweenVectors( const XYZ &point1, const XYZ &point2 )
01714 {
01715     return CosAngleBetweenVectors( point1.X, point1.Y, point1.Z, point2.X, point2.Y, point2.Z );
01716 }
01717
01718 //-----------------------------------------------------------------------------------------------------------
01719 double AngleBetweenVectors( double x1, double y1, double z1, double x2, double y2, double z2 )
01720 {
01721     return std::acos( CosAngleBetweenVectors( x1, y1, z1, x2, y2, z2 ) );
01722 }
01723
```

```
01724 double AngleBetweenVectors( const XYZ &vec1, const XYZ &vec2 )
01725 {
01726     return std::acos( CosAngleBetweenVectors( vec1, vec2 ) );
01727 }
01728 //-----------------------------------------------------------------------------------------------------------
01729 void VectorFromTwoPoints( double x1, double y1, double z1, double x2, double y2, double z2, double &xV, double &yV,
       double &zV )
01730 {
01731     xV = x2 - x1;
01732     yV = y2 - y1;
01733     zV = z2 - z1;
01734 }
01735
01736 XYZ VectorFromTwoPoints( const XYZ &point1, const XYZ &point2 )
01737 {
01738     XYZ result;
01739     VectorFromTwoPoints( point1.X, point1.Y, point1.Z, point2.X, point2.Y, point2.Z, result.X, result.Y, result.Z );
01740     return result;
01741 }
01742
01743 }
01744 }
```

10.19 Файл spml.cpp

SPML (Special Program Modules Library) - Специальная Библиотека Программных Модулей (СБПМ)

#include <spml.h>
Граф включаемых заголовочных файлов для spml.cpp:



Пространства имен

- namespace SPML

  Специальная библиотека программных модулей (СБ ПМ)

Функции

- std::string SPML::GetVersion ()

  Возвращает строку, содержащую информацию о версии
- void SPML::ClearConsole ()

  Очистка консоли (терминала) в *nix.

10.19.1 Подробное описание

SPML (Special Program Modules Library) - Специальная Библиотека Программных Модулей (СБПМ)

Дата

14.07.20 - создан

Автор

Соболев А.А.

См. определение в файле spml.cpp

10.20 spml.cpp

См. документацию.

```
00001 //------------------------------------------------------------------------------------------------------------
00010
00011 #include <spml.h>
00012
00013 namespace SPML
00014 {
00015 //------------------------------------------------------------------------------------------------------------
00016 std::string GetVersion()
00017 {
00018     return "SPML_25.11.2021_v01_Develop";
00019 }
00020
00021 //------------------------------------------------------------------------------------------------------------
00022 void ClearConsole()
00023 {
00024     // 1 способ:
00025     // Reset terminal - быстрее, чем вызов std::system
00026     printf("\033c");
00027
00028     // 2 способ:
00029     // CSI[2J clears screen, CSI[H moves the cursor to top-left corner
00030 //    std::cout « "\x1B[2J\x1B[H";
00031
00032     // 3 способ:
00033 //    std::system("clear");
00034 }
00035
00036 }
```

10.21 Файл README.md

Предметный указатель