

C-Coil (Circular Coil Object-oriented Interaction Library)

1.0.0

Generated by Doxygen 1.9.1

1 Namespace Index	1
1.1 Namespace List	1
2 Class Index	3
2.1 Class List	3
3 Namespace Documentation	5
3.1 Benchmark Namespace Reference	5
3.1.1 Detailed Description	7
3.2 Compare Namespace Reference	7
3.2.1 Detailed Description	8
3.3 Legendre Namespace Reference	8
3.3.1 Detailed Description	8
3.4 Test Namespace Reference	8
3.4.1 Detailed Description	9
3.5 threadPool Namespace Reference	9
3.5.1 Detailed Description	10
3.6 vec3 Namespace Reference	10
3.6.1 Detailed Description	10
4 Class Documentation	11
4.1 Coil Class Reference	11
4.1.1 Detailed Description	15
4.1.2 Member Function Documentation	15
4.1.2.1 computeAllAPotentialVectors() [1/2]	16
4.1.2.2 computeAllAPotentialVectors() [2/2]	17
4.1.2.3 computeAllBFieldVectors() [1/2]	17
4.1.2.4 computeAllBFieldVectors() [2/2]	18
4.1.2.5 computeAllBGradientMatrices() [1/2]	18
4.1.2.6 computeAllBGradientMatrices() [2/2]	19
4.1.2.7 computeAllEFieldVectors() [1/2]	19
4.1.2.8 computeAllEFieldVectors() [2/2]	19
4.1.2.9 computeAllForceTorqueArrangements()	20
4.1.2.10 computeAllMutualInductanceArrangements()	21
4.1.2.11 computeAndSetSelfInductance()	21
4.1.2.12 computeAPotentialVector() [1/2]	22
4.1.2.13 computeAPotentialVector() [2/2]	22
4.1.2.14 computeBFieldVector() [1/2]	23
4.1.2.15 computeBFieldVector() [2/2]	23
4.1.2.16 computeBGradientMatrix() [1/2]	23
4.1.2.17 computeBGradientMatrix() [2/2]	24
4.1.2.18 computeEFieldVector() [1/2]	24
4.1.2.19 computeEFieldVector() [2/2]	25

4.1.2.20 computeForceOnDipoleMoment() [1/2]	25
4.1.2.21 computeForceOnDipoleMoment() [2/2]	25
4.1.2.22 computeForceTorque() [1/2]	26
4.1.2.23 computeForceTorque() [2/2]	27
4.1.2.24 computeMutualInductance() [1/2]	27
4.1.2.25 computeMutualInductance() [2/2]	28
4.1.2.26 computeSecondaryInducedVoltage() [1/2]	28
4.1.2.27 computeSecondaryInducedVoltage() [2/2]	29
4.1.2.28 setPositionAndOrientation()	29
4.2 CoilGroup Class Reference	30
4.2.1 Detailed Description	31
4.2.2 Member Function Documentation	32
4.2.2.1 computeAllAPotentialVectors()	32
4.2.2.2 computeAllBFieldVectors()	32
4.2.2.3 computeAllBGradientMatrices()	33
4.2.2.4 computeAllEFieldVectors()	33
4.2.2.5 computeAllForceTorqueArrangements()	34
4.2.2.6 computeAllMutualInductanceArrangements()	34
4.2.2.7 computeAPotentialVector()	35
4.2.2.8 computeBFieldVector()	35
4.2.2.9 computeBGradientMatrix()	36
4.2.2.10 computeEFieldVector()	36
4.2.2.11 computeForceOnDipoleMoment()	36
4.2.2.12 computeForceTorque()	37
4.2.2.13 computeMutualInductance()	37
4.2.2.14 removeCoil()	38
4.3 CoilPairArguments Struct Reference	39
4.4 vec3::Matrix3 Class Reference	39
4.4.1 Detailed Description	40
4.5 vec3::Matrix3Array Class Reference	40
4.5.1 Detailed Description	41
4.6 PrecisionArguments Struct Reference	41
4.6.1 Detailed Description	42
4.6.2 Constructor & Destructor Documentation	42
4.6.2.1 PrecisionArguments()	42
4.6.3 Member Function Documentation	42
4.6.3.1 getCoilPrecisionArgumentsCPU()	43
4.6.3.2 getCoilPrecisionArgumentsGPU()	43
4.6.3.3 getSecondaryCoilPrecisionArgumentsGPU()	43
4.7 PrecisionFactor Struct Reference	44
4.7.1 Detailed Description	44
4.8 vec3::Triplet Class Reference	44

4.8.1 Detailed Description	45
4.9 vec3::Vector3 Class Reference	45
4.9.1 Detailed Description	46
4.10 vec3::Vector3Array Class Reference	46
4.10.1 Detailed Description	47
Index	49

Chapter 1

Namespace Index

1.1 Namespace List

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

Benchmark	Contains functions that are used to benchmark performance of various methods concerning Coil and CoilGroup . Performance data obtained from them is very useful for assessing implementation efficiency	5
Compare	Contains functions that are used to compare precision of Coil and CoilGroup methods with relevant literature, as well as some miscellaneous CPU, GPU, and MTD value generation	7
Legendre	Contains matrices with precomputed Gauss-Legendre quadrature weights and positions up to maxLegendreOrder. Row defines the quadrature order n, and column the appropriate index i (up to n)	8
Test	Contains functions that are used to test whether compute methods are implemented correctly. This module is accessible only from C++ and is not included in Python	8
threadPool	Contains a wrapper for the CTPL library (https://github.com/vit-vit/CTPL) which is used for multithreading	9
vec3	Contains custom objects (Vector3 and Matrix3) used to define the coordinate system and appropriate transformations	10

Chapter 2

Class Index

2.1 Class List

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

Coil	Primary class in this project. Has a unique identifier. Models a circular coil with a rectangular cross section and uniform current density	11
CoilGroup	Represents a collection of unique Coil instances and is useful for representing multi-coil systems. Enables faster field and interaction calculations, especially when using the GPU	30
CoilPairArguments	Structure used to store precision data (block and increment count) for a system of two coils. Used for custom precision arguments in interaction calculations	39
vec3::Matrix3	Represents a rank 2 tensor of dimension 3, which is represented as a square matrix	39
vec3::Matrix3Array	Represents <code>std::vector<vec3::Matrix3></code> for easier handling and additional features. Allows only xx, xy, xz, yx, yy, yz, zx, zy, or zz components, and det() values, to be extracted to a <code>std::vector</code>	40
PrecisionArguments	Structure used to store precision data (block and increment count) for an individual Coil	41
PrecisionFactor	Structure used to represent universal calculation precision. A custom precision measure from interval [1.0, 15.0]. Increasing the factor by 1.0 doubles the performance	44
vec3::Triplet	Represents a general ordered sequence (tuple) with 3 elements	44
vec3::Vector3	Represents a rank 1 tensor of dimension 3, which is a member of the oriented Euclidean vector space. It is commonly referred to as a 3D (Cartesian) Vector	45
vec3::Vector3Array	Represents <code>std::vector<vec3::Vector3></code> for easier handling and additional features. Allows only x, y, or z components, as well as abs() values, to be extracted to <code>std::vector<double></code>	46

Chapter 3

Namespace Documentation

3.1 Benchmark Namespace Reference

Contains functions that are used to benchmark performance of various methods concerning [Coil](#) and [CoilGroup](#). Performance data obtained from them is very useful for assessing implementation efficiency.

Functions

- void [mathFunctions](#) ()
Benchmarks performance of basic math functions such as +, cos, log, and atan2.
- void [computeFieldsST](#) (int opCount=50 '000)
Benchmarks single thread (CPU_ST) performance of [Coil](#) field compute methods for a given number of points.
- void [computeAllFields](#) ([PrecisionFactor](#) precisionFactor=[PrecisionFactor](#)(), int opCount=20 '000, int repeatCount=1, int threadCount=g_defaultThreadCount)
Benchmarks performance of [Coil](#) field compute methods for different ComputeMethods, a given number of points, and number of threads. The number of repetitions is determined by repeatCount.
- void [computeAllFieldsEveryCoilType](#) (int opCount=100 '000, int threadCount=g_defaultThreadCount)
A very comprehensive benchmark of precision factor and threadCount influence on performance of field calculations. Filaments, thin, flat, and rectangular coils are included.
- void [computeAllFieldsWorkloadScalingMT](#) ([PrecisionFactor](#) precisionFactor=[PrecisionFactor](#)(), int threadCount=g_defaultThreadCount, int maxPointsLog2=g_maxPot)
Benchmarks CPU_MT performance scaling of [Coil](#) field compute methods with the number of points for {2^0, 2^1, 2^2, ..., 2^maxPointsLog2}, given [PrecisionFactor](#), and number of threads.
- void [computeAllFieldsWorkloadScalingGPU](#) ([PrecisionFactor](#) precisionFactor=[PrecisionFactor](#)(), int maxPointsLog2=g_maxPot)
Benchmarks GPU performance scaling of [Coil](#) field compute methods with the number of points from {2^0, 2^1, 2^2, ..., 2^maxPointsLog2}, and given [PrecisionFactor](#).
- void [mInductanceZAxis](#) (ComputeMethod computeMethod=CPU_ST, int threadCount=g_defaultThreadCount)
Benchmarks [Coil::computeMutualInductance](#) for z-axis case with two thick coils.
- void [mInductanceZAxisMTScaling](#) (int maxThreadCount=g_defaultThreadCount)
Benchmarks [Coil::computeMutualInductance](#) for z-axis case with two thick coils, CPU_MT ComputeMethod, and number of threads from {1,..., maxThreadCount}.
- void [selfInductance](#) ()
Benchmarks [Coil::computeAndSetSelfInductance](#) for a thick coil and PrecisionFactors from {1, 2,..., 15}.
- void [mInductanceGeneral](#) (ComputeMethod computeMethod=CPU_ST, int threadCount=g_defaultThreadCount)

- Benchmarks [Coil::computeMutualInductance](#) for general case with two thick coils.
- void [mInductanceGeneralMTScaling](#) (int maxThreadCount=g_defaultThreadCount)

Benchmarks [Coil::computeMutualInductance](#) for general case with two thick coils, CPU_MT ComputeMethod, and number of threads from {1,..., maxThreadCount}.
- void [forceZAxis](#) (ComputeMethod computeMethod=CPU_ST, int threadCount=g_defaultThreadCount)

Benchmarks [Coil::computeForceTorque](#) for z-axis case with two thick coils.
- void [forceZAxisMTScaling](#) (int maxThreadCount=g_defaultThreadCount)

Benchmarks [Coil::computeForceTorque](#) for z-axis case with two thick coils, CPU_MT ComputeMethod, and number of threads from {1,..., maxThreadCount}.
- void [forceGeneral](#) (ComputeMethod computeMethod=CPU_ST, int threadCount=g_defaultThreadCount)

Benchmarks [Coil::computeForceTorque](#) for general case with two thick coils.
- void [forceGeneralMTScaling](#) (int maxThreadCount=g_defaultThreadCount)

Benchmarks [Coil::computeForceTorque](#) for general case with two thick coils for CPU_MT ComputeMethod, and number of threads from {1,..., maxThreadCount}.
- void [coilMInductanceAndForceComputeAll](#) (int configCount=100, int threadCount=g_defaultThreadCount)

Benchmarks [Coil::computeAllMutualInductanceArrangements](#) and [Coil::computeAllForceTorqueArrangements](#) for a given number of configurations.
- void [coilMInductanceAndForceComputeAllMTvsMTD](#) (PrecisionFactor precisionFactor=PrecisionFactor(), int threadCount=g_defaultThreadCount)

Benchmarks CPU_MT (MT vs MTD) performance of [Coil::computeAllMutualInductanceArrangements](#) and [Coil::computeAllForceTorqueArrangements](#) for given PrecisionFactor and threadCount.
- void [coilMInductanceAndForceComputeAllGPU](#) (int configCount=10 '000)

Benchmarks GPU performance of [Coil::computeAllMutualInductanceArrangements](#) and [Coil::computeAllForceTorqueArrangements](#) for a given number of configurations.
- void [coilGroupComputeAllFields](#) (PrecisionFactor precisionFactor=PrecisionFactor(), int coilCount=50, int opCount=100 '000, int threadCount=g_defaultThreadCount)

Benchmarks performance of [CoilGroup](#) field calculations with different compute methods with given PrecisionFactor, number of coils, and number of points.
- void [coilGroupComputeAllFieldsMTvsMTD](#) (int threadCount=g_defaultThreadCount, int pointCount=20 '000)

Benchmarks performance of CPU_MT ComputeMethod (MT vs MTD) for [CoilGroup](#) fieldCalculations.
- void [coilGroupComputeAllFieldsGPU](#) (int coilCount=100, int opCount=131 '072)

Benchmarks GPU performance of [CoilGroup](#) fieldCalculations for a given number of coils and points.
- void [coilGroupComputeAllFieldsMTScaling](#) (PrecisionFactor precisionFactor=PrecisionFactor(), int threadCount=g_defaultThreadCount, int coilCount=100, int maxPointsLog2=g_maxPotGroup)

Benchmarks CPU_MT performance scaling of [CoilGroup](#) field compute methods with the number of points from {2⁰, 2¹, 2², ..., 2^{maxPointsLog2}}, given PrecisionFactor, number of coils, and number of threads.
- void [coilGroupComputeAllFieldsGPUScaling](#) (PrecisionFactor precisionFactor=PrecisionFactor(), int coilCount=100, int maxPointsLog2=g_maxPotGroup)

Benchmarks GPU performance scaling of [CoilGroup](#) field compute methods with the number of points for {2⁰, 2¹, 2², ..., 2^{maxPointsLog2}}, given PrecisionFactor, and number of coils.
- void [coilGroupMInductanceAndForce](#) (int repeatCount=2, int threadCount=g_defaultThreadCount)

Benchmarks performance of [CoilGroup::computeMInductance](#) and [CoilGroup::computeForceTorque](#) for different ComputeMethods and a given number of threads. The number of repetitions is determined by repeatCount.
- void [coilGroupMInductanceAndForceAll](#) (int coilCount=50, int opCount=10, int threadCount=g_defaultThreadCount)

Benchmarks performance of [CoilGroup::computeAllMutualInductanceArrangements](#) and [CoilGroup::computeAllForceTorqueArrangements](#) for different ComputeMethods and a given number of threads. The number of repetitions is determined by repeatCount.
- void [coilGroupMInductanceAndForceAllGPU](#) (int coilCount=100, int opCount=1000)

Benchmarks GPU performance of [CoilGroup::computeAllMutualInductanceArrangements](#) and [CoilGroup::computeAllForceTorqueArrangements](#) and a given number of threads. The number of repetitions is determined by repeatCount.

3.1.1 Detailed Description

Contains functions that are used to benchmark performance of various methods concerning [Coil](#) and [CoilGroup](#). Performance data obtained from them is very useful for assessing implementation efficiency.

3.2 Compare Namespace Reference

Contains functions that are used to compare precision of [Coil](#) and [CoilGroup](#) methods with relevant literature, as well as some miscellaneous CPU, GPU, and MTD value generation.

Functions

- void [fieldsPrecisionCPUvsGPU](#) ()
Compares vector potential A and magnetic field B computed by the CPU and GPU. Relative errors are printed.
- void [mutualInductanceAndForceTorquePrecisionCPUvsGPU](#) ()
Compares the precision of CPU and GPU methods for computing mutual inductance, force, and torque.
- void [mutualInductanceSpecialCase](#) ()
Prints mutual inductance parallel case values from paper K. Song, J. Feng, R. Zhao, X. Wu, "A general mutual inductance formula for parallel non-coaxial circular coils," in ACES Journal, vol. 34, no. 9, pp. 1385-1390, September 2019.
- void [forceTorqueFilamentsZAxis](#) ()
Prints force between a pair of filaments placed on the z-axis which is used to asses the precision of approach in paper Z. J. Wang and Y. Ren, "Magnetic Force and Torque Calculation Between Circular Coils With Nonparallel Axes," in IEEE Trans. Appl. Supercond., vol. 24, no. 4, pp. 1-5, Aug. 2014, Art no. 4901505.
- void [forceTorqueThickCoilsGeneral](#) ()
Prints values from paper Z. J. Wang and Y. Ren, "Magnetic Force and Torque Calculation Between Circular Coils With Nonparallel Axes," in IEEE Trans. Appl. Supercond., vol. 24, no. 4, pp. 1-5, Aug. 2014, Art no. 4901505.
- void [forceTorqueThinCoilsZAxis](#) ()
Prints values from paper S. I. Babic and C. Akyel, "Magnetic Force Calculation Between Thin Coaxial Circular Coils in Air," IEEE Trans. Magn., vol. 44, no. 4, pp. 445-452, April 2008.
- void [forceTorqueFilamentsGeneral](#) ()
Prints values from S. Babic and C. Akyel, "Magnetic Force Between Inclined Circular Filaments Placed in Any Desired Position," IEEE Trans. Magn., vol. 48, no. 1, pp. 69-80, Jan. 2012.
- void [forceTorqueZAxis](#) ()
Prints values for z-axis force on a system of custom coils used in our prior research.
- void [forceOnDipoleVsForceTorque](#) ()
Evaluates the approximation of a [Coil](#) with a dipole moment for appropriate cases: coils far apart and one coil much smaller than the other.
- void [mutualInductanceMisalignedCoils](#) ()
Prints mutual inductance general case values from paper S. Babic, C. Akyel and S. J. Salon, "New procedures for calculating the mutual inductance of the system: filamentary circular coil-massive circular solenoid," in IEEE Trans. Magn., vol. 39, no. 3, pp. 1131-1134, May 2003.
- void [mutualInductanceParallelAxesGraphs](#) ()
Outputs and prints mutual inductance parallel case values from paper J. T. Conway, "Inductance Calculations for Circular Coils of Rectangular Cross Section and Parallel Axes Using Bessel and Struve Functions," IEEE Trans. Magn., vol. 46, no. 1, pp. 75-81, Jan. 2010.
- void [mutualInductanceParallelAxes](#) ()
Outputs and prints mutual inductance parallel case values from paper Y. Luo, X. Wang and X. Zhou, "Inductance Calculations for Circular Coils With Rectangular Cross Section and Parallel Axes Using Inverse Mellin Transform and Generalized Hypergeometric Functions," IEEE Trans. Power Electron., vol. 32, no. 2, pp. 1367-1374, Feb. 2017.
- void [mutualInductanceGeneralCase](#) ()

Outputs and prints mutual inductance general case values from paper J. T. Conway, "Mutual inductance of thick coils for arbitrary relative orientation and position," 2017 Progress in Electromagnetics Research Symposium - Fall (PIERS - FALL), 2017, pp. 1388-1395.

- void [mutualInductanceGeneralGraphs](#) ()

Outputs mutual inductance general case values from paper Y. Wang, X. Xie, Y. Zhou and W. Huan, "Calculation and Modeling Analysis of Mutual Inductance Between Coreless Circular Coils With Rectangular Cross Section in Arbitrary Spatial Position," 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC), 2020, pp. 1258-1267.

- void [mutualInductanceGeneralEdgeCases](#) ()

Prints mutual inductance general case values which we found to show reduced precision (edge cases).

- void [mutualInductanceZAxis](#) ()

Prints mutual inductance z-axis case values from paper T. Župan, Ž. Štih and B. Trkulja, "Fast and Precise Method for Inductance Calculation of Coaxial Circular Coils With Rectangular Cross Section Using the One-Dimensional Integration of Elementary Functions Applicable to Superconducting Magnets," IEEE Trans. Appl. Supercond., vol. 24, no. 2, pp. 81-89, April 2014, Art no. 4901309.

- void [selfInductance](#) ()

Outputs and prints self inductance values from paper J. T. Conway, "Inductance Calculations for Circular Coils of Rectangular Cross Section and Parallel Axes Using Bessel and Struve Functions," IEEE Trans. Magn., vol. 46, no. 1, pp. 75-81, Jan. 2010.

- void [fieldsCoilGroupMTD](#) (int coilCount=100, int pointCount=10 '000, int threadCount=g_defaultThreadCount, bool print=true)

Calculates and can print values of magnetic field for a number of coils in toroidal arrangement.

3.2.1 Detailed Description

Contains functions that are used to compare precision of [Coil](#) and [CoilGroup](#) methods with relevant literature, as well as some miscellaneous CPU, GPU, and MTD value generation.

3.3 Legendre Namespace Reference

Contains matrices with precomputed Gauss-Legendre quadrature weights and positions up to maxLegendreOrder. Row defines the quadrature order n, and column the appropriate index i (up to n).

Variables

- const double **positionMatrix** [maxLegendreOrder][maxLegendreOrder]
- const double **weightsMatrix** [maxLegendreOrder][maxLegendreOrder]
- const int **maxLegendreOrder** = 100

3.3.1 Detailed Description

Contains matrices with precomputed Gauss-Legendre quadrature weights and positions up to maxLegendreOrder. Row defines the quadrature order n, and column the appropriate index i (up to n).

3.4 Test Namespace Reference

Contains functions that are used to test whether compute methods are implemented correctly. This module is accessible only from C++ and is not included in Python.

Functions

- void [testNewCoilParameters](#) ()
Tests basic constructors and lazy loading of different parameters.
- void [testCoilPositionAndRotation](#) ()
Tests if the rotation matrix properly transforms fields (coordinate conversion).
- void [testCoilGroupComputeAllMTD](#) ()
Tests if coarse-grained multithreading (MTD) is implemented correctly (CPU_ST is reference).
- void [testMutualInductanceGeneralForZAxis](#) (ComputeMethod computeMethod)
Tests if general mutual inductance method returns values similar to z-axis mutual inductance method.
- void [testMInductanceZAxisArgumentGeneration](#) ()
Tests the increment balancing algorithm for z-axis [CoilPairArguments](#) generation.
- void [testMInductanceZAxisDifferentGeometries](#) ()
Tests 16 possible configurations of coils for z-axis case. There are two circular coils, each either a filament, flat coil, thin coil, or a rectangular coil.
- void [testMInductanceGeneralArgumentGeneration](#) ()
Tests the increment balancing algorithm for general [CoilPairArguments](#) generation.
- void [testMInductanceGeneralDifferentGeometries](#) ()
Tests 16 possible configurations of coils for general case. There are two circular coils, each either a filament, flat coil, thin coil, or a rectangular coil.
- void [testAmpereForceGeneralForZAxis](#) ()
Tests if general force and torque method returns values similar to z-axis force method.
- void [testGradientTensor](#) ()
Tests if the magnetic gradient tensor behaves properly, especially for z-axis positions (singular case).
- void [testCoilGroupFieldsMTD](#) ()
Tests if [CoilGroup](#) coarse-grained multithreading (MTD) is implemented properly.
- void [testCoilMInductanceArrangements](#) ()
Tests if [Coil::computeAllMutualInductanceArrangements](#) is implemented correctly (CPU_ST is reference)
- void [testCoilForceArrangements](#) ()
Tests if [Coil::computeAllForceTorqueArrangements](#) is implemented correctly (CPU_ST is reference)
- void [testGroupMInductanceArrangements](#) ()
Tests if [CoilGroup::computeAllMutualInductanceArrangements](#) is implemented correctly (CPU_ST is reference)
- void [testGroupForceArrangements](#) ()
Tests if [CoilGroup::computeAllForceTorqueArrangements](#) is implemented correctly (CPU_ST is reference)

3.4.1 Detailed Description

Contains functions that are used to test whether compute methods are implemented correctly. This module is accessible only from C++ and is not included in Python.

3.5 threadPool Namespace Reference

Contains a wrapper for the CTPL library (<https://github.com/vit-vit/CTPL>) which is used for multi-threading.

Classes

- class [ThreadPoolControl](#)

Variables

- `ctpl::thread_pool` **threadPool**

3.5.1 Detailed Description

Contains a wrapper for the CTPL library (<https://github.com/vit-vit/CTPL>) which is used for multi-threading.

3.6 vec3 Namespace Reference

Contains custom objects ([Vector3](#) and [Matrix3](#)) used to define the coordinate system and appropriate transformations.

Classes

- class [Vector3](#)
Represents a rank 1 tensor of dimension 3, which is a member of the oriented Euclidean vector space. It is commonly referred to as a 3D (Cartesian) Vector.
- class [Matrix3](#)
Represents a rank 2 tensor of dimension 3, which is represented as a square matrix.
- class [Triplet](#)
Represents a general ordered sequence (tuple) with 3 elements.
- class [Vector3Array](#)
Represents `std::vector<vec3::Vector3>` for easier handling and additional features. Allows only x, y, or z components, as well as `abs()` values, to be extracted to `std::vector<double>`.
- class [Matrix3Array](#)
Represents `std::vector<vec3::Matrix3>` for easier handling and additional features. Allows only xx, xy, xz, yx, yy, yz, zx, zy, or zz components, and `det()` values, to be extracted to a `std::vector`.

3.6.1 Detailed Description

Contains custom objects ([Vector3](#) and [Matrix3](#)) used to define the coordinate system and appropriate transformations.

Chapter 4

Class Documentation

4.1 Coil Class Reference

Primary class in this project. Has a unique identifier. Models a circular coil with a rectangular cross section and uniform current density.

```
#include <Coil.h>
```

Public Member Functions

- unsigned long long [getId](#) () const
Returns the unique coil identifier.
- double [getInnerRadius](#) () const
Returns the radius of inner cylinder of the circular coil.
- double [getThickness](#) () const
Returns the thickness of windings of the circular coil.
- double [getLength](#) () const
Returns the length of the circular coil.
- int [getNumOfTurns](#) () const
Returns the number of windings (turns) of the circular coil.
- double [getCurrentDensity](#) () const
Returns the current density of a circular coil. Ill defined for thin coils, pancakes and filaments.
- double [getCurrent](#) () const
Returns a current passing through each winding of a circular coil.
- double [getWireResistivity](#) () const
Returns current wire resistivity, determined by the used material. By default, copper is used.
- bool [isSineDriven](#) () const
Returns if the coil is sine wave (AC) driven or DC driven.
- double [getSineFrequency](#) () const
Returns the frequency of the AC sine wave driving the coil. 0.0 if the it is DC driven.
- [vec3::Vector3](#) [getMagneticMoment](#) ()
Uses lazy loading and returns an equivalent magnetic dipole moment (approximation at large distance)
- double [getAverageWireThickness](#) () const
Calculates average wire thickness supposing the winding is orthogonal.
- double [getSelfInductance](#) () const

- Returns last set or calculated value of self inductance.*

 - double [getResistance](#) ()
Uses lazy loading and returns coil resistance of the coil with skin effect compensation.
 - double [getReactance](#) ()
Uses lazy loading and returns inductive reactance of the coil, capacitance not included.
 - double [getImpedance](#) ()
Uses lazy loading and returns the magnitude of the coil impedance.
 - const [PrecisionArguments](#) & [getPrecisionSettingsCPU](#) () const
Returns the default [PrecisionArguments](#) used for CPU calculations.
 - const [PrecisionArguments](#) & [getPrecisionSettingsGPU](#) () const
Returns the default [PrecisionArguments](#) used for GPU calculations.
 - int [getThreadCount](#) () const
Returns the default number of threads.
 - bool [isUsingFastMethod](#) () const
Returns the type of methods the coil is using. Thin and rectangular coils use fast methods.
 - CoilType [getCoilType](#) () const
Returns the type of circular coil with rectangular cross section.
 - [vec3::Vector3](#) [getPositionVector](#) () const
Returns position of the coil in external Cartesian coordinate system.
 - std::pair< double, double > [getRotationAngles](#) () const
Returns a pair of angles <yAxisAngle, zAxisAngle> which represent the coil orientation.
 - [vec3::Matrix3](#) [getTransformationMatrix](#) () const
Returns the inverse transformation matrix used to simplify field calculation.
 - [vec3::Matrix3](#) [getInverseTransformationMatrix](#) () const
Returns the transformation matrix used to adapt field tensors to the external coordinate system.
 - void [setCurrentDensity](#) (double currentDensity)
Sets current density and calculates appropriate current. Not suitable for thin coils, pancakes and filaments.
 - void [setCurrent](#) (double current)
Sets the current through windings and calculates the appropriate current density.
 - void [setWireResistivity](#) (double wireResistivity)
Sets wire resistivity, necessary for resistance calculation.
 - void [setSineFrequency](#) (double sineFrequency)
Sets AC sine driven frequency. 0.0 is used for DC.
 - void [setDefaultPrecisionCPU](#) (const [PrecisionArguments](#) &precisionSettings)
Sets the given, custom [PrecisionArguments](#) as default for CPU calculations.
 - void [setDefaultPrecisionCPU](#) ([PrecisionFactor](#) precisionFactor=[PrecisionFactor](#)())
Calculates the [PrecisionArguments](#) for the appropriate [PrecisionFactor](#) and sets them as default for CPU calculations.
 - void [setDefaultPrecisionGPU](#) (const [PrecisionArguments](#) &precisionSettings)
Sets the given, custom [PrecisionArguments](#) as default for GPU calculations.
 - void [setDefaultPrecisionGPU](#) ([PrecisionFactor](#) precisionFactor=[PrecisionFactor](#)())
Calculates the [PrecisionArguments](#) for the appropriate [PrecisionFactor](#) and sets them as default for GPU calculations.
 - void [setDefaultPrecision](#) ([PrecisionFactor](#) precisionFactor=[PrecisionFactor](#)())
Calculates the [PrecisionArguments](#) for the appropriate [PrecisionFactor](#) and sets them as default for both CPU and GPU.
 - void [setThreadCount](#) (int threadCount)
Sets the number of threads used in field and interaction calculations.
 - void [setPositionAndOrientation](#) ([vec3::Vector3](#) positionVector=[vec3::Vector3](#)(), double yAxisAngle=0.0, double zAxisAngle=0.0)
Repositions and reorients the coil in the external coordinate system.
 - void [setSelfInductance](#) (double selfInductance)
Sets self inductance of the coil to the provided value, overriding all previous calculations.

- `vec3::Vector3 computeAPotentialVector (vec3::Vector3 pointVector) const`
Calculates vector potential A of the magnetic field at the specified point. Uses precision internally defined by default← PrecisionCPU.
- `vec3::Vector3 computeAPotentialVector (vec3::Vector3 pointVector, const PrecisionArguments &used← Precision) const`
Calculates vector potential A of the magnetic field at the specified point. Uses provided PrecisionArguments for precision settings.
- `vec3::Vector3 computeBFieldVector (vec3::Vector3 pointVector) const`
Calculates magnetic flux density B (magnetic field) at the specified point. Uses precision internally defined by defaultPrecisionCPU.
- `vec3::Vector3 computeBFieldVector (vec3::Vector3 pointVector, const PrecisionArguments &usedPrecision) const`
Calculates magnetic flux density B (magnetic field) at the specified point. Uses provided PrecisionArguments for precision settings.
- `vec3::Vector3 computeEFieldVector (vec3::Vector3 pointVector) const`
Calculates the amplitude vector of electric field E in sinusoidal steady-state at the specified point. Uses precision internally defined by defaultPrecisionCPU.
- `vec3::Vector3 computeEFieldVector (vec3::Vector3 pointVector, const PrecisionArguments &usedPrecision) const`
Calculates the amplitude vector of electric field E in sinusoidal steady-state at the specified point. Uses provided PrecisionArguments for precision settings.
- `vec3::Matrix3 computeBGradientMatrix (vec3::Vector3 pointVector) const`
Calculates the gradient G of the magnetic field (total derivative of B) at the specified point. Uses precision internally defined by defaultPrecisionCPU.
- `vec3::Matrix3 computeBGradientMatrix (vec3::Vector3 pointVector, const PrecisionArguments &used← Precision) const`
Calculates the gradient G of the magnetic field (total derivative of B) at the specified point. Uses provided PrecisionArguments for precision settings.
- `vec3::Vector3Array computeAllAPotentialVectors (const vec3::Vector3Array &pointVectors, ComputeMethod computeMethod=CPU_ST) const`
Calculates vector potential A of the magnetic field for a number of specified points. Uses precision internally defined by defaultPrecisionCPU.
- `vec3::Vector3Array computeAllAPotentialVectors (const vec3::Vector3Array &pointVectors, const PrecisionArguments &usedPrecision, ComputeMethod computeMethod=CPU_ST) const`
Calculates vector potential A of the magnetic field for a number of specified points. Uses provided PrecisionArguments for precision settings.
- `vec3::Vector3Array computeAllBFieldVectors (const vec3::Vector3Array &pointVectors, ComputeMethod computeMethod=CPU_ST) const`
Calculates magnetic flux density B (magnetic field) for a number of specified points. Uses precision internally defined by defaultPrecisionCPU.
- `vec3::Vector3Array computeAllBFieldVectors (const vec3::Vector3Array &pointVectors, const PrecisionArguments &usedPrecision, ComputeMethod computeMethod=CPU_ST) const`
Calculates magnetic flux density B (magnetic field) for a number of specified points. Uses provided PrecisionArguments for precision settings.
- `vec3::Vector3Array computeAllEFieldVectors (const vec3::Vector3Array &pointVectors, ComputeMethod computeMethod=CPU_ST) const`
Calculates the amplitude vector of electric field E in sinusoidal steady-state for a number of specified points. Uses precision internally defined by defaultPrecisionCPU.
- `vec3::Vector3Array computeAllEFieldVectors (const vec3::Vector3Array &pointVectors, const PrecisionArguments &usedPrecision, ComputeMethod computeMethod=CPU_ST) const`
Calculates the amplitude vector of electric field E in sinusoidal steady-state for a number of specified points. Uses provided PrecisionArguments for precision settings.
- `vec3::Matrix3Array computeAllBGradientMatrices (const vec3::Vector3Array &pointVectors, ComputeMethod computeMethod=CPU_ST) const`

Calculates the gradient G of the magnetic field (total derivative of B) for a number of specified points. Uses precision internally defined by defaultPrecisionCPU.

- `vec3::Matrix3Array computeAllBGradientMatrices` (const `vec3::Vector3Array` &pointVectors, const `PrecisionArguments` &usedPrecision, `ComputeMethod` computeMethod=`CPU_ST`) const

Calculates the gradient G of the magnetic field (total derivative of B) for a number of specified points. Uses provided `PrecisionArguments` for precision settings.

- double `computeSecondaryInducedVoltage` (const `Coil` &secondary, `PrecisionFactor` precisionFactor=`PrecisionFactor()`, `ComputeMethod` computeMethod=`CPU_ST`) const

Calculates the magnitude of sinusoidal steady-state voltage induced in the secondary coil. Generates `CoilPairArguments` from given precisionFactor. Similar to `computeMutualInductance`.

- double `computeSecondaryInducedVoltage` (const `Coil` &secondary, const `CoilPairArguments` &inductanceArguments, `ComputeMethod` computeMethod=`CPU_ST`) const

Calculates the magnitude of sinusoidal steady-state voltage induced in the secondary coil. Uses provided `CoilPairArguments` for precision settings. Similar to `computeMutualInductance`.

- double `computeAndSetSelfInductance` (`PrecisionFactor` precisionFactor)
Special method which returns self inductance L of the given coil and sets it internally.

- std::pair< `vec3::Vector3`, `vec3::Vector3` > `computeForceOnDipoleMoment` (`vec3::Vector3` pointVector, `vec3::Vector3` dipoleMoment) const

Calculates force F and torque T between a coil and magnetostatic object with a dipole moment. Uses precision internally defined by defaultPrecisionCPU.

- std::pair< `vec3::Vector3`, `vec3::Vector3` > `computeForceOnDipoleMoment` (`vec3::Vector3` pointVector, `vec3::Vector3` dipoleMoment, const `PrecisionArguments` &usedPrecision) const

Calculates force F and torque T between a coil and magnetostatic object with a dipole moment. Uses provided `PrecisionArguments` for precision settings.

- operator std::string () const

Generates a string object with all properties of the `Coil` instance.

CoilConstructors

- **Coil** (double innerRadius, double thickness, double length, int numOfTurns, double current, double wireResistivity, double sineFrequency, `PrecisionFactor` precisionFactor=`PrecisionFactor()`, int threadCount=g_defaultThreadCount, `vec3::Vector3` coordinatePosition=`vec3::Vector3()`, double yAxisAngle=0.0, double zAxisAngle=0.0)
- **Coil** (double innerRadius, double thickness, double length, int numOfTurns, double current, double wireResistivity, double sineFrequency, const `PrecisionArguments` &precisionSettingsCPU, const `PrecisionArguments` &precisionSettingsGPU, int threadCount=g_defaultThreadCount, `vec3::Vector3` coordinatePosition=`vec3::Vector3()`, double yAxisAngle=0.0, double zAxisAngle=0.0)
- **Coil** (double innerRadius, double thickness, double length, int numOfTurns, double current, double sineFrequency, `PrecisionFactor` precisionFactor=`PrecisionFactor()`, int threadCount=g_defaultThreadCount, `vec3::Vector3` coordinatePosition=`vec3::Vector3()`, double yAxisAngle=0.0, double zAxisAngle=0.0)
- **Coil** (double innerRadius, double thickness, double length, int numOfTurns, double current, double sineFrequency, const `PrecisionArguments` &precisionSettingsCPU, const `PrecisionArguments` &precisionSettingsGPU, int threadCount=g_defaultThreadCount, `vec3::Vector3` coordinatePosition=`vec3::Vector3()`, double yAxisAngle=0.0, double zAxisAngle=0.0)
- **Coil** (double innerRadius, double thickness, double length, int numOfTurns, double current, `PrecisionFactor` precisionFactor=`PrecisionFactor()`, int threadCount=g_defaultThreadCount, `vec3::Vector3` coordinatePosition=`vec3::Vector3()`, double yAxisAngle=0.0, double zAxisAngle=0.0)
- **Coil** (double innerRadius, double thickness, double length, int numOfTurns, double current, const `PrecisionArguments` &precisionSettingsCPU, const `PrecisionArguments` &precisionSettingsGPU, int threadCount=g_defaultThreadCount, `vec3::Vector3` coordinatePosition=`vec3::Vector3()`, double yAxisAngle=0.0, double zAxisAngle=0.0)
- **Coil** (double innerRadius, double thickness, double length, int numOfTurns, `PrecisionFactor` precisionFactor=`PrecisionFactor()`, int threadCount=g_defaultThreadCount, `vec3::Vector3` coordinatePosition=`vec3::Vector3()`, double yAxisAngle=0.0, double zAxisAngle=0.0)
- **Coil** (double innerRadius, double thickness, double length, int numOfTurns, const `PrecisionArguments` &precisionSettingsCPU, const `PrecisionArguments` &precisionSettingsGPU, int threadCount=g_defaultThreadCount, `vec3::Vector3` coordinatePosition=`vec3::Vector3()`, double yAxisAngle=0.0, double zAxisAngle=0.0)

Static Public Member Functions

- static double `computeMutualInductance` (const `Coil` &primary, const `Coil` &secondary, `PrecisionFactor` precisionFactor=`PrecisionFactor`(), `ComputeMethod` computeMethod=`CPU_ST`)
Calculates the mutual inductance M between two given coils. Generates `CoilPairArguments` from given precisionFactor.
- static double `computeMutualInductance` (const `Coil` &primary, const `Coil` &secondary, const `CoilPairArguments` &inductanceArguments, `ComputeMethod` computeMethod=`CPU_ST`)
Calculates the mutual inductance M between two given coils. Uses provided `CoilPairArguments` for precision settings.
- static std::pair< `vec3::Vector3`, `vec3::Vector3` > `computeForceTorque` (const `Coil` &primary, const `Coil` &secondary, `PrecisionFactor` precisionFactor=`PrecisionFactor`(), `ComputeMethod` computeMethod=`CPU_ST`)
Calculates the force F and torque T between two coils. Generates `CoilPairArguments` from given precisionFactor.
- static std::pair< `vec3::Vector3`, `vec3::Vector3` > `computeForceTorque` (const `Coil` &primary, const `Coil` &secondary, const `CoilPairArguments` &forceArguments, `ComputeMethod` computeMethod=`CPU_ST`)
Calculates the force F and torque T between two coils. Generates `CoilPairArguments` from given precisionFactor.
- static std::vector< double > `computeAllMutualInductanceArrangements` (const `Coil` &primary, const `Coil` &secondary, const `vec3::Vector3Array` &primaryPositions, const `vec3::Vector3Array` &secondaryPositions, const std::vector< double > &primaryYAngles, const std::vector< double > &primaryZAngles, const std::vector< double > &secondaryYAngles, const std::vector< double > &secondaryZAngles, `PrecisionFactor` precisionFactor=`PrecisionFactor`(), `ComputeMethod` computeMethod=`CPU_ST`)
Calculates mutual inductance M between two coils for different coil configurations. All positional arguments can be changed. Generates `CoilPairArguments` from given precisionFactor.
- static std::vector< std::pair< `vec3::Vector3`, `vec3::Vector3` > > `computeAllForceTorqueArrangements` (const `Coil` &primary, const `Coil` &secondary, const `vec3::Vector3Array` &primaryPositions, const `vec3::Vector3Array` &secondaryPositions, const std::vector< double > &primaryYAngles, const std::vector< double > &primaryZAngles, const std::vector< double > &secondaryYAngles, const std::vector< double > &secondaryZAngles, `PrecisionFactor` precisionFactor=`PrecisionFactor`(), `ComputeMethod` computeMethod=`CPU_ST`)
Calculates force F and torque T between two coils for different coil configurations. All positional arguments can be changed. Generates `CoilPairArguments` from given precisionFactor.

4.1.1 Detailed Description

Primary class in this project. Has a unique identifier. Models a circular coil with a rectangular cross section and uniform current density.

The model works best for a solid block of material and when the effects of windings are negligible. Primary attributes are length (b), thickness (a), the inner radius (R) of the internal cylindrical hole, and the number of turns of wire (N). These attributes cannot be changed. Length and thickness can be set to 0.0 and that is interpreted as having a thin coil ($b = 0.0$), flat coil ($a = 0.0$), or filament ($a = 0.0$, $b = 0.0$). The coil is oriented like a spherical vector, when angles are (0.0, 0.0) the coil axis is along the z-axis. The first angle is the rotation around the y-axis, and the second around the z-axis. Default precision settings are stored and used if custom ones are not provided. Calculating fields inside the coil is not recommended.

4.1.2 Member Function Documentation

4.1.2.1 computeAllAPotentialVectors() [1/2]

```
vec3::Vector3Array Coil::computeAllAPotentialVectors (
    const vec3::Vector3Array & pointVectors,
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates vector potential A of the magnetic field for a number of specified points. Uses precision internally defined by defaultPrecisionCPU.

There are multiple compute methods, GPU acceleration is best suited for a large number of points.

Parameters

<i>pointVectors</i>	An array of radius vectors wrapped in class Vector3Array
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of Cartesian vectors which represent vector potential A at specified points.

4.1.2.2 computeAllAPotentialVectors() [2/2]

```
vec3::Vector3Array Coil::computeAllAPotentialVectors (
    const vec3::Vector3Array & pointVectors,
    const PrecisionArguments & usedPrecision,
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates vector potential A of the magnetic field for a number of specified points. Uses provided [PrecisionArguments](#) for precision settings.

There are multiple compute methods, GPU acceleration is best suited for a large number of points.

Parameters

<i>pointVectors</i>	An array of radius vectors wrapped in class Vector3Array.
<i>usedPrecision</i>	Custom precision settings used for this particular calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of Cartesian vectors which represent vector potential A at specified points.

4.1.2.3 computeAllBFieldVectors() [1/2]

```
vec3::Vector3Array Coil::computeAllBFieldVectors (
    const vec3::Vector3Array & pointVectors,
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates magnetic flux density B (magnetic field) for a number of specified points. Uses precision internally defined by defaultPrecisionCPU.

There are multiple compute methods, GPU acceleration is best suited for a large number of points.

Parameters

<i>pointVectors</i>	An array of radius vectors wrapped in class Vector3Array.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of Cartesian vectors which represent magnetic flux B at specified points.

4.1.2.4 computeAllBFieldVectors() [2/2]

```
vec3::Vector3Array Coil::computeAllBFieldVectors (
    const vec3::Vector3Array & pointVectors,
    const PrecisionArguments & usedPrecision,
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates magnetic flux density B (magnetic field) for a number of specified points. Uses provided [PrecisionArguments](#) for precision settings.

There are multiple compute methods, GPU acceleration is best suited for a large number of points.

Parameters

<i>pointVectors</i>	An array of radius vectors wrapped in class Vector3Array.
<i>usedPrecision</i>	Custom precision settings used for this particular calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of Cartesian vectors which represent magnetic flux B at specified points.

4.1.2.5 computeAllBGradientMatrices() [1/2]

```
vec3::Matrix3Array Coil::computeAllBGradientMatrices (
    const vec3::Vector3Array & pointVectors,
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates the gradient G of the magnetic field (total derivative of B) for a number of specified points. Uses precision internally defined by defaultPrecisionCPU.

There are multiple compute methods, GPU acceleration is best suited for a large number of points.

Parameters

<i>pointVectors</i>	An array of radius vectors wrapped in class Vector3Array.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of 3x3 matrices which represents the magnetic gradient matrix G at specified points.

4.1.2.6 computeAllBGradientMatrices() [2/2]

```
vec3::Matrix3Array Coil::computeAllBGradientMatrices (
    const vec3::Vector3Array & pointVectors,
    const PrecisionArguments & usedPrecision,
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates the gradient G of the magnetic field (total derivative of B) for a number of specified points. Uses provided [PrecisionArguments](#) for precision settings.

There are multiple compute methods, GPU acceleration is best suited for a large number of points.

Parameters

<i>pointVectors</i>	An array of radius vectors wrapped in class Vector3Array.
<i>usedPrecision</i>	Custom precision settings used for this particular calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of 3x3 matrices which represents the magnetic gradient matrix G at specified points.

4.1.2.7 computeAllEFieldVectors() [1/2]

```
vec3::Vector3Array Coil::computeAllEFieldVectors (
    const vec3::Vector3Array & pointVectors,
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates the amplitude vector of electric field E in sinusoidal steady-state for a number of specified points. Uses precision internally defined by defaultPrecisionCPU.

There are multiple compute methods, GPU acceleration is best suited for a large number of points.

Parameters

<i>pointVectors</i>	An array of radius vectors wrapped in class Vector3Array.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of Cartesian vectors which represent the amplitude of electric field E at specified points.

4.1.2.8 computeAllEFieldVectors() [2/2]

```
vec3::Vector3Array Coil::computeAllEFieldVectors (
    const vec3::Vector3Array & pointVectors,
```

```
const PrecisionArguments & usedPrecision,
ComputeMethod computeMethod = CPU_ST ) const
```

Calculates the amplitude vector of electric field E in sinusoidal steady-state for a number of specified points. Uses provided [PrecisionArguments](#) for precision settings.

There are multiple compute methods, GPU acceleration is best suited for a large number of points.

Parameters

<i>pointVectors</i>	An array of radius vectors wrapped in class Vector3Array.
<i>usedPrecision</i>	Custom precision settings used for this particular calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of Cartesian vectors which represent the amplitude of electric field E at specified points.

4.1.2.9 computeAllForceTorqueArrangements()

```
std::vector< std::pair< vec3::Vector3, vec3::Vector3 > > Coil::computeAllForceTorqueArrangements
(
    const Coil & primary,
    const Coil & secondary,
    const vec3::Vector3Array & primaryPositions,
    const vec3::Vector3Array & secondaryPositions,
    const std::vector< double > & primaryYAngles,
    const std::vector< double > & primaryZAngles,
    const std::vector< double > & secondaryYAngles,
    const std::vector< double > & secondaryZAngles,
    PrecisionFactor precisionFactor = PrecisionFactor(),
    ComputeMethod computeMethod = CPU_ST ) [static]
```

Calculates force F and torque T between two coils for different coil configurations. All positional arguments can be changed. Generates [CoilPairArguments](#) from given precisionFactor.

This method is exceptionally powerful because it can more efficiently utilise the CPU with distributed (coarse-grained) multithreading, and especially the GPU with a special pure GPU implementation of force and torque calculation. Computation times can be as low as several microseconds per configuration.

Parameters

<i>primary</i>	The coil that generates the magnetic field
<i>secondary</i>	The coil that is represented with a number of points for which the field is calculated.
<i>primaryPositions</i>	Positions of the center of the primary coil.
<i>secondaryPositions</i>	Positions of the center of the secondary coil.
<i>primaryYAngles</i>	Primary coil rotation angles along the y-axis.
<i>primaryZAngles</i>	Primary coil rotation angles along the z-axis.
<i>secondaryYAngles</i>	Secondary coil rotation angles along the y-axis.
<i>secondaryZAngles</i>	Secondary coil rotation angles along the z-axis.
<i>precisionFactor</i>	Determines the precision of given calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of pairs of force (first) and torque (second) vectors, one for each appropriate configuration.

4.1.2.10 computeAllMutualInductanceArrangements()

```
std::vector< double > Coil::computeAllMutualInductanceArrangements (
    const Coil & primary,
    const Coil & secondary,
    const vec3::Vector3Array & primaryPositions,
    const vec3::Vector3Array & secondaryPositions,
    const std::vector< double > & primaryYAngles,
    const std::vector< double > & primaryZAngles,
    const std::vector< double > & secondaryYAngles,
    const std::vector< double > & secondaryZAngles,
    PrecisionFactor precisionFactor = PrecisionFactor(),
    ComputeMethod computeMethod = CPU_ST ) [static]
```

Calculates mutual inductance M between two coils for different coil configurations. All positional arguments can be changed. Generates [CoilPairArguments](#) from given precisionFactor.

This method is exceptionally powerful because it can more efficiently utilise the CPU with distributed (coarse-grained) multithreading, and especially the GPU with a special pure GPU implementation of mutual inductance calculation. Computation times can be as low as several microseconds per configuration.

Parameters

<i>primary</i>	The coil that generates the vector potential
<i>secondary</i>	The coil that is represented with a number of points for which the potential is calculated.
<i>primaryPositions</i>	Positions of the center of the primary coil.
<i>secondaryPositions</i>	Positions of the center of the secondary coil.
<i>primaryYAngles</i>	Primary coil rotation angles along the y-axis.
<i>primaryZAngles</i>	Primary coil rotation angles along the z-axis.
<i>secondaryYAngles</i>	Secondary coil rotation angles along the y-axis.
<i>secondaryZAngles</i>	Secondary coil rotation angles along the z-axis.
<i>precisionFactor</i>	Determines the precision of given calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of mutual inductance values, one for each appropriate configuration.

4.1.2.11 computeAndSetSelfInductance()

```
double Coil::computeAndSetSelfInductance (
    PrecisionFactor precisionFactor )
```

Special method which returns self inductance L of the given coil and sets it internally.

This method is exclusively single threaded and represents a shortcoming of this approach. Low precision factors, below 5.0, are not advisable and good precision (error of order $1e-6$) can be achieved with precision factor 10.0. It works well for thick and thin coils, but poorly for flat coils, and does not work for filaments (loops) as the integral is inherently divergent.

Parameters

<i>precisionFactor</i>	Determines the precision of given calculation.
------------------------	--

Returns

Self inductance of the coil.

4.1.2.12 computeAPotentialVector() [1/2]

```
vec3::Vector3 Coil::computeAPotentialVector (
    vec3::Vector3 pointVector ) const
```

Calculates vector potential A of the magnetic field at the specified point. Uses precision internally defined by defaultPrecisionCPU.

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the field is calculated.
--------------------	---

Returns

3D Cartesian vector which represents vector potential A.

4.1.2.13 computeAPotentialVector() [2/2]

```
vec3::Vector3 Coil::computeAPotentialVector (
    vec3::Vector3 pointVector,
    const PrecisionArguments & usedPrecision ) const
```

Calculates vector potential A of the magnetic field at the specified point. Uses provided [PrecisionArguments](#) for precision settings.

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the field is calculated.
<i>usedPrecision</i>	Custom precision settings used for this particular calculation.

Returns

3D Cartesian vector which represents vector potential A.

4.1.2.14 computeBFieldVector() [1/2]

```
vec3::Vector3 Coil::computeBFieldVector (
    vec3::Vector3 pointVector ) const
```

Calculates magnetic flux density B (magnetic field) at the specified point. Uses precision internally defined by defaultPrecisionCPU.

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the field is calculated.
--------------------	---

Returns

3D Cartesian vector which represents magnetic flux density B.

4.1.2.15 computeBFieldVector() [2/2]

```
vec3::Vector3 Coil::computeBFieldVector (
    vec3::Vector3 pointVector,
    const PrecisionArguments & usedPrecision ) const
```

Calculates magnetic flux density B (magnetic field) at the specified point. Uses provided [PrecisionArguments](#) for precision settings.

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the field is calculated.
<i>usedPrecision</i>	Custom precision settings used for this particular calculation.

Returns

3D Cartesian vector which represents magnetic flux density B.

4.1.2.16 computeBGradientMatrix() [1/2]

```
vec3::Matrix3 Coil::computeBGradientMatrix (
    vec3::Vector3 pointVector ) const
```

Calculates the gradient G of the magnetic field (total derivative of B) at the specified point. Uses precision internally defined by defaultPrecisionCPU.

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the field is calculated.
--------------------	---

Returns

3x3 Matrix which represents the magnetic gradient matrix G.

4.1.2.17 computeBGradientMatrix() [2/2]

```
vec3::Matrix3 Coil::computeBGradientMatrix (
    vec3::Vector3 pointVector,
    const PrecisionArguments & usedPrecision ) const
```

Calculates the gradient G of the magnetic field (total derivative of B) at the specified point. Uses provided [PrecisionArguments](#) for precision settings.

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the field is calculated.
<i>usedPrecision</i>	Custom precision settings used for this particular calculation.

Returns

3x3 matrix which represents the magnetic gradient matrix G.

4.1.2.18 computeEFieldVector() [1/2]

```
vec3::Vector3 Coil::computeEFieldVector (
    vec3::Vector3 pointVector ) const
```

Calculates the amplitude vector of electric field E in sinusoidal steady-state at the specified point. Uses precision internally defined by defaultPrecisionCPU.

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the field is calculated.
--------------------	---

Returns

3D Cartesian vector which represents the amplitude of electric field E.

4.1.2.19 computeEFieldVector() [2/2]

```
vec3::Vector3 Coil::computeEFieldVector (
    vec3::Vector3 pointVector,
    const PrecisionArguments & usedPrecision ) const
```

Calculates the amplitude vector of electric field E in sinusoidal steady-state at the specified point. Uses provided [PrecisionArguments](#) for precision settings.

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the field is calculated.
<i>usedPrecision</i>	Custom precision settings used for this particular calculation.

Returns

3D Cartesian vector which represents the amplitude of electric field E.

4.1.2.20 computeForceOnDipoleMoment() [1/2]

```
std::pair< vec3::Vector3, vec3::Vector3 > Coil::computeForceOnDipoleMoment (
    vec3::Vector3 pointVector,
    vec3::Vector3 dipoleMoment ) const
```

Calculates force F and torque T between a coil and magnetostatic object with a dipole moment. Uses precision internally defined by defaultPrecisionCPU.

This method can prove particularly useful for approximating the force and torque between two coils which are sufficiently far apart, or when the secondary coil is very small. It can also be useful in particle simulations where the magnetic dipole moment is not negligible

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the magnetic dipole is located.
<i>dipoleMoment</i>	Magnetic dipole moment vector of a secondary coil or another object (magnet, particle).

Returns

Pair of Cartesian vectors which represent force (first) and torque (second).

4.1.2.21 computeForceOnDipoleMoment() [2/2]

```
std::pair< vec3::Vector3, vec3::Vector3 > Coil::computeForceOnDipoleMoment (
    vec3::Vector3 pointVector,
```

```
vec3::Vector3 dipoleMoment,
const PrecisionArguments & usedPrecision ) const
```

Calculates force F and torque T between a coil and magnetostatic object with a dipole moment. Uses provided [PrecisionArguments](#) for precision settings.

This method can prove particularly useful for approximating the force and and torque between two coils which are sufficiently far apart, or when the secondary coil is very small. It can also be useful in particle simulations where the magnetic dipole moment is not negligible

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the magnetic dipole is located.
<i>dipoleMoment</i>	Magnetic dipole moment vector of a secondary coil or another object (magnet, particle).
<i>usedPrecision</i>	Custom precision settings used for this particular calculation.

Returns

Pair of Cartesian vectors which represent force (first) and torque (second).

4.1.2.22 computeForceTorque() [1/2]

```
std::pair< vec3::Vector3, vec3::Vector3 > Coil::computeForceTorque (
    const Coil & primary,
    const Coil & secondary,
    const CoilPairArguments & forceArguments,
    ComputeMethod computeMethod = CPU_ST ) [static]
```

Calculates the force F and torque T between two coils. Generates [CoilPairArguments](#) from given precisionFactor.

For better precision, the primary coil should be the bigger one, length is the most important parameter. There are more performant implementations if both coils lie on the z-axis and have rotation angles set to 0. Using CPU_MT compute method is highly advisable, especially for higher precision factors, and the GPU is a good option when an error of 1e-5 is good enough for the application (usual error of order 1e-6).

Parameters

<i>primary</i>	The coil that generates the magnetic field.
<i>secondary</i>	The coil that is represented with a number of points for which the field is calculated.
<i>inductanceArguments</i>	Custom precision settings used for this particular calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Pair of Cartesian vectors which represent force (first) and torque (second).

4.1.2.23 computeForceTorque() [2/2]

```
std::pair< vec3::Vector3, vec3::Vector3 > Coil::computeForceTorque (
    const Coil & primary,
    const Coil & secondary,
    PrecisionFactor precisionFactor = PrecisionFactor(),
    ComputeMethod computeMethod = CPU_ST ) [static]
```

Calculates the force F and torque T between two coils. Generates [CoilPairArguments](#) from given precisionFactor.

For better precision, the primary coil should be the bigger one, length is the most important parameter. There are more performant implementations if both coils lie on the z-axis and have rotation angles set to 0. Using CPU_MT compute method is highly advisable, especially for higher precision factors and GPU is a good option when an error of 1e-5 is good enough for the application (usual error of order 1e-6).

Parameters

<i>primary</i>	The coil that generates the magnetic field.
<i>secondary</i>	The coil that is represented with a number of points for which the field is calculated.
<i>precisionFactor</i>	Determines the precision of given calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Pair of Cartesian vectors which represent force (first) and torque (second).

4.1.2.24 computeMutualInductance() [1/2]

```
double Coil::computeMutualInductance (
    const Coil & primary,
    const Coil & secondary,
    const CoilPairArguments & inductanceArguments,
    ComputeMethod computeMethod = CPU_ST ) [static]
```

Calculates the mutual inductance M between two given coils. Uses provided [CoilPairArguments](#) for precision settings.

For better precision, the primary coil should be the bigger one, length is the most important parameter. There are more performant implementations if both coils lie on the z-axis and have rotation angles set to 0. Using CPU_MT compute method is highly advisable, especially for higher precision factors. GPU is a good option when an error of 1e-5 is good enough for the application (usual error of order 1e-6).

Parameters

<i>primary</i>	The coil that generates the vector potential.
<i>secondary</i>	The coil that is represented with a number of points for which the potential is calculated.
<i>inductanceArguments</i>	Custom precision settings used for this particular calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Mutual inductance of the system of two coils.

4.1.2.25 computeMutualInductance() [2/2]

```
double Coil::computeMutualInductance (
    const Coil & primary,
    const Coil & secondary,
    PrecisionFactor precisionFactor = PrecisionFactor(),
    ComputeMethod computeMethod = CPU_ST ) [static]
```

Calculates the mutual inductance M between two given coils. Generates [CoilPairArguments](#) from given precisionFactor.

For better precision, the primary coil should be the bigger one, length is the most important parameter. There are more performant implementations if both coils lie on the z-axis and have rotation angles set to 0. Using CPU_MT compute method is highly advisable, especially for higher precision factors. GPU is a good option when an error of 1e-5 is good enough for the application (usual error of order 1e-6).

Parameters

<i>primary</i>	The coil that generates the vector potential.
<i>secondary</i>	The coil that is represented with a number of points for which the potential is calculated.
<i>precisionFactor</i>	Determines the precision of given calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Mutual inductance of the system of two coils.

4.1.2.26 computeSecondaryInducedVoltage() [1/2]

```
double Coil::computeSecondaryInducedVoltage (
    const Coil & secondary,
    const CoilPairArguments & inductanceArguments,
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates the magnitude of sinusoidal steady-state voltage induced in the secondary coil. Uses provided [CoilPairArguments](#) for precision settings. Similar to computeMutualInductance.

Parameters

<i>secondary</i>	The coil that is represented with a number of points for which the E field is calculated.
<i>inductanceArguments</i>	Custom precision settings used for this particular calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Magnitude of the voltage induced on the secondary coil.

4.1.2.27 computeSecondaryInducedVoltage() [2/2]

```
double Coil::computeSecondaryInducedVoltage (
    const Coil & secondary,
    PrecisionFactor precisionFactor = PrecisionFactor(),
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates the magnitude of sinusoidal steady-state voltage induced in the secondary coil. Generates [CoilPairArguments](#) from given precisionFactor. Similar to computeMutualInductance.

Parameters

<i>secondary</i>	The coil that is represented with a number of points for which the E field is calculated.
<i>precisionFactor</i>	Determines the precision of given calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Magnitude of the voltage induced on the secondary coil.

4.1.2.28 setPositionAndOrientation()

```
void Coil::setPositionAndOrientation (
    vec3::Vector3 positionVector = vec3::Vector3(),
    double yAxisAngle = 0.0,
    double zAxisAngle = 0.0 )
```

Repositions and reorients the coil in the external coordinate system.

Parameters

<i>positionVector</i>	Position of the coil center.
<i>yAxisAngle</i>	Rotation angle around the y-axis from [0, PI].
<i>zAxisAngle</i>	Rotation angle around the z-axis from [0, 2PI].

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

- /home/davor/C++/Coil-Evolution/src/Coil/Coil.h
- /home/davor/C++/Coil-Evolution/src/Coil/Coil.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/Fields/CalculateAllFieldsGPU.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/Fields/CalculateAllFieldsMT.cxx

- /home/davor/C++/Coil-Evolution/src/Coil/Fields/CalculateFieldsFast.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/Fields/CalculateFieldsMethods.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/Fields/CalculateFieldsSlow.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/Fields/ComputeFieldsMethods.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/ForceAndTorque/CalculateForceArrangements.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/ForceAndTorque/CalculateForceGeneral.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/ForceAndTorque/CalculateForceZAxis.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/ForceAndTorque/ComputeForceMethods.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/MInductance/CalculateMInductanceArrangements.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/MInductance/CalculateMInductanceGeneral.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/MInductance/CalculateMInductanceZAxis.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/MInductance/ComputeMInductanceMethods.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/Utils/CalculateAttributes.cxx
- /home/davor/C++/Coil-Evolution/src/Coil/Utils/SupportFunctions.cxx

4.2 CoilGroup Class Reference

Represents a collection of unique [Coil](#) instances and is useful for representing multi-coil systems. Enables faster field and interaction calculations, especially when using the GPU.

```
#include <CoilGroup.h>
```

Public Member Functions

- [CoilGroup](#) (std::vector< std::shared_ptr< [Coil](#) >> memberCoils=std::vector< std::shared_ptr< [Coil](#) >>(), [PrecisionFactor](#) precisionFactor=[PrecisionFactor](#)(), int threadCount=g_defaultThreadCount)
All arguments have defaults so it is also a default constructor, best used that way.
- [PrecisionFactor](#) getDefaultPrecisionFactor () const
Returns the default [PrecisionFactor](#) according to which default arguments for all members are generated.
- int getThreadCount () const
Returns the number of threads used in CPU_MT calculations.
- const std::vector< std::shared_ptr< [Coil](#) > > & getMemberCoils () const
Returns a constant reference to internal std::vector.
- void setDefaultPrecisionFactor ([PrecisionFactor](#) precisionFactor=[PrecisionFactor](#)())
Setts the default [PrecisionFactor](#) which is immediately applied to all members.
- void setThreadCount (int threadCount)
Setts the default number of threads which is immediately applied to all members.
- void addCoil (double innerRadius, double thickness, double length, int numOfTurns, double current=1.0, [PrecisionFactor](#) precisionFactor=[PrecisionFactor](#)(), int coilThreads=g_defaultThreadCount, [vec3::Vector3](#) coordinatePosition=[vec3::Vector3](#)(), double yAxisAngle=0.0, double zAxisAngle=0.0)
Adds a new member [Coil](#) to the back, most common [Coil](#) constructor is imitated for simplicity.
- void removeCoil (size_t index)
Removes the [Coil](#) at the selected index from the [CoilGroup](#).
- [Coil](#) & operator[] (size_t index) const
- [vec3::Vector3](#) computeAPotentialVector ([vec3::Vector3](#) pointVector) const
Calculates vector potential A of the magnetic field at the specified point from all member Coils. Uses internally defined defaultPrecisionFactor.
- [vec3::Vector3](#) computeBFieldVector ([vec3::Vector3](#) pointVector) const
Calculates magnetic flux density B (magnetic field) at the specified point from all member Coils. Uses internally defined defaultPrecisionFactor.

- [vec3::Vector3 computeEFieldVector](#) ([vec3::Vector3](#) pointVector) const
Calculates the amplitude vector of electric field E in sinusoidal steady-state at the specified point from all member Coils. Uses internally defined defaultPrecisionFactor.
- [vec3::Matrix3 computeBGradientMatrix](#) ([vec3::Vector3](#) pointVector) const
Calculates the gradient G of the magnetic field (total derivative of B) at the specified point from all member Coils. Uses internally defined defaultPrecisionFactor.
- [vec3::Vector3Array computeAllAPotentialVectors](#) (const [vec3::Vector3Array](#) &pointVectors, ComputeMethod computeMethod=CPU_ST) const
Calculates vector potential A of the magnetic field from all member Coils for a number of specified points. Uses internally defined defaultPrecisionFactor.
- [vec3::Vector3Array computeAllBFieldVectors](#) (const [vec3::Vector3Array](#) &pointVectors, ComputeMethod computeMethod=CPU_ST) const
Calculates magnetic flux density B (magnetic field) from all member Coils for a number of specified points. Uses internally defined defaultPrecisionFactor.
- [vec3::Vector3Array computeAllEFieldVectors](#) (const [vec3::Vector3Array](#) &pointVectors, ComputeMethod computeMethod=CPU_ST) const
Calculates the amplitude vector of electric field E in sinusoidal steady-state from all member Coils for a number of specified points. Uses internally defined defaultPrecisionFactor.
- [vec3::Matrix3Array computeAllBGradientMatrices](#) (const [vec3::Vector3Array](#) &pointVectors, ComputeMethod computeMethod=CPU_ST) const
Calculates the gradient G of the magnetic field (total derivative of B) for a number of specified points. Uses internally defined defaultPrecisionFactor.
- double [computeMutualInductance](#) (const [Coil](#) &secondary, [PrecisionFactor](#) precisionFactor=[PrecisionFactor](#)(), ComputeMethod computeMethod=CPU_ST) const
Calculates the mutual inductance M between a provided coil and the rest of the member coils. Uses separate [CoilPairArguments](#) for every pair of coils, generated with the given [PrecisionFactor](#).
- std::pair< [vec3::Vector3](#), [vec3::Vector3](#) > [computeForceTorque](#) (const [Coil](#) &secondary, [PrecisionFactor](#) precisionFactor=[PrecisionFactor](#)(), ComputeMethod computeMethod=CPU_ST) const
Calculates the force F and torque T on a provided coil from the rest of the member coils. Uses separate [CoilPairArguments](#) for every pair of coils, generated with the given [PrecisionFactor](#).
- std::pair< [vec3::Vector3](#), [vec3::Vector3](#) > [computeForceOnDipoleMoment](#) ([vec3::Vector3](#) pointVector, [vec3::Vector3](#) dipoleMoment) const
Calculates force F and torque T between a coil and magnetostatic object with a dipole moment. Uses precision internally defined by defaultPrecisionCPU.
- std::vector< double > [computeAllMutualInductanceArrangements](#) (const [Coil](#) &secondary, const [vec3::Vector3Array](#) &secondaryPositions, const std::vector< double > &secondaryYAngles, const std::vector< double > &secondaryZAngles, [PrecisionFactor](#) precisionFactor=[PrecisionFactor](#)(), ComputeMethod computeMethod=CPU_ST) const
Calculates the mutual inductance M between a provided coil and the rest of the member coils for multiple positions and orientations of the secondary coil.
- std::vector< std::pair< [vec3::Vector3](#), [vec3::Vector3](#) > > [computeAllForceTorqueArrangements](#) (const [Coil](#) &secondary, const [vec3::Vector3Array](#) &secondaryPositions, const std::vector< double > &secondaryYAngles, const std::vector< double > &secondaryZAngles, [PrecisionFactor](#) precisionFactor=[PrecisionFactor](#)(), ComputeMethod computeMethod=CPU_ST) const
Calculates force F and torque T on a provided coil from the rest of the member coils for multiple positions and orientations of the secondary coil.
- [operator std::string](#) () const
Generates a string object with all properties of the [CoilGroup](#) instance (and appropriate member Coils).

4.2.1 Detailed Description

Represents a collection of unique [Coil](#) instances and is useful for representing multi-coil systems. Enables faster field and interaction calculations, especially when using the GPU.

As all coils are unique, shared pointers are used to ensure there is only once instance of each [Coil](#). The coils are accessed individually as if this class were a list. The default [PrecisionFactor](#) is defined as well as the number of threads used for calculations. When there are many coils, calculations are accelerated with MTD (Multi-Threading Distributed, or more commonly Coarse-grained parallelism) methods for 10-100% more performance, especially when using high core counts. Most methods are similar to [Coil](#) methods. When calculating MInductance or Force, a [Coil](#) from the group, can be passed as an argument and its contribution is then ignored.

4.2.2 Member Function Documentation

4.2.2.1 computeAllAPotentialVectors()

```
vec3::Vector3Array CoilGroup::computeAllAPotentialVectors (
    const vec3::Vector3Array & pointVectors,
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates vector potential A of the magnetic field from all member Coils for a number of specified points. Uses internally defined defaultPrecisionFactor.

There are multiple compute methods, GPU acceleration is best suited for over 100 points. MTD is used if the number of coils is two or more times the number of threads and CPU_MT is selected.

Parameters

<i>pointVectors</i>	An array of radius vectors wrapped in class Vector3Array.
<i>usedPrecision</i>	Custom precision settings used for this particular calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of Cartesian vectors which represent vector potential A at specified points.

4.2.2.2 computeAllBFieldVectors()

```
vec3::Vector3Array CoilGroup::computeAllBFieldVectors (
    const vec3::Vector3Array & pointVectors,
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates magnetic flux density B (magnetic field) from all member Coils for a number of specified points. Uses internally defined defaultPrecisionFactor.

There are multiple compute methods, GPU acceleration is best suited for over 100 points. MTD is used if the number of coils is two or more times the number of threads and CPU_MT is selected.

Parameters

<i>pointVectors</i>	An array of radius vectors wrapped in class Vector3Array.
<i>usedPrecision</i>	Custom precision settings used for this particular calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of Cartesian vectors which represent magnetic flux B at specified points.

4.2.2.3 computeAllBGradientMatrices()

```
vec3::Matrix3Array CoilGroup::computeAllBGradientMatrices (
    const vec3::Vector3Array & pointVectors,
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates the gradient G of the magnetic field (total derivative of B) for a number of specified points. Uses internally defined defaultPrecisionFactor.

There are multiple compute methods, GPU acceleration is best suited for over 100 points. MTD is used if the number of coils is two or more times the number of threads and CPU_MT is selected.

Parameters

<i>pointVectors</i>	An array of radius vectors wrapped in class Vector3Array.
<i>usedPrecision</i>	Custom precision settings used for this particular calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of 3x3 matrices which represents the magnetic gradient matrix G at specified points.

4.2.2.4 computeAllEFieldVectors()

```
vec3::Vector3Array CoilGroup::computeAllEFieldVectors (
    const vec3::Vector3Array & pointVectors,
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates the amplitude vector of electric field E in sinusoidal steady-state from all member Coils for a number of specified points. Uses internally defined defaultPrecisionFactor.

There are multiple compute methods, GPU acceleration is best suited for over 100 points. MTD is used if the number of coils is two or more times the number of threads and CPU_MT is selected.

Parameters

<i>pointVectors</i>	An array of radius vectors wrapped in class Vector3Array.
<i>usedPrecision</i>	Custom precision settings used for this particular calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of Cartesian vectors which represent the amplitude of electric field E at specified points.

4.2.2.5 computeAllForceTorqueArrangements()

```
std::vector< std::pair< vec3::Vector3, vec3::Vector3 > > CoilGroup::computeAllForceTorque↵
Arrangements (
    const Coil & secondary,
    const vec3::Vector3Array & secondaryPositions,
    const std::vector< double > & secondaryYAngles,
    const std::vector< double > & secondaryZAngles,
    PrecisionFactor precisionFactor = PrecisionFactor(),
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates force F and torque T on a provided coil from the rest of the member coils for multiple positions and orientations of the secondary coil.

Uses separate [CoilPairArguments](#) for every pair of coils, generated with the given [PrecisionFactor](#), when using the CPU. When using the GPU every [Coil](#) is assigned a calculated precision factor which makes the precision equivalent, in terms of total increments, to given [PrecisionFactor](#). This method is exceptionally powerful because it can more efficiently utilise the CPU with MTD, and especially the GPU with a special pure GPU implementation of mutual inductance calculation.

Parameters

<i>secondary</i>	The coil that is represented with a number of points for which the magnetic field is calculated.
<i>secondaryPositions</i>	Positions of the center of the secondary coil.
<i>secondaryYAngles</i>	Secondary coil rotation angles along the y-axis.
<i>secondaryZAngles</i>	Secondary coil rotation angles along the z-axis.
<i>precisionFactor</i>	Determines the precision of given calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default. Array of pairs of force (first) and torque (second) vectors, one for each appropriate configuration.

4.2.2.6 computeAllMutualInductanceArrangements()

```
std::vector< double > CoilGroup::computeAllMutualInductanceArrangements (
    const Coil & secondary,
    const vec3::Vector3Array & secondaryPositions,
    const std::vector< double > & secondaryYAngles,
    const std::vector< double > & secondaryZAngles,
    PrecisionFactor precisionFactor = PrecisionFactor(),
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates the mutual inductance M between a provided coil and the rest of the member coils for multiple positions and orientations of the secondary coil.

Uses separate [CoilPairArguments](#) for every pair of coils, generated with the given [PrecisionFactor](#), when using the CPU. When using the GPU every [Coil](#) is assigned a calculated precision factor which makes the precision equivalent, in terms of total increments, to given [PrecisionFactor](#). This method is exceptionally powerful because it can more efficiently utilise the CPU with MTD, and especially the GPU with a special pure GPU implementation of mutual inductance calculation.

Parameters

<i>secondary</i>	The coil that is represented with a number of points for which the potential is calculated.
<i>secondaryPositions</i>	Positions of the center of the secondary coil.
<i>secondaryYAngles</i>	Secondary coil rotation angles along the y-axis.
<i>secondaryZAngles</i>	Secondary coil rotation angles along the z-axis.
<i>precisionFactor</i>	Determines the precision of given calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Array of mutual inductance values, one for each appropriate configuration.

4.2.2.7 computeAPotentialVector()

```
vec3::Vector3 CoilGroup::computeAPotentialVector (
    vec3::Vector3 pointVector ) const
```

Calculates vector potential A of the magnetic field at the specified point from all member Coils. Uses internally defined defaultPrecisionFactor.

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the field is calculated.
--------------------	---

Returns

3D Cartesian vector which represents vector potential A.

4.2.2.8 computeBFieldVector()

```
vec3::Vector3 CoilGroup::computeBFieldVector (
    vec3::Vector3 pointVector ) const
```

Calculates magnetic flux density B (magnetic field) at the specified point from all member Coils. Uses internally defined defaultPrecisionFactor.

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the field is calculated.
--------------------	---

Returns

3D Cartesian vector which represents vector potential A.

4.2.2.9 computeBGradientMatrix()

```
vec3::Matrix3 CoilGroup::computeBGradientMatrix (
    vec3::Vector3 pointVector ) const
```

Calculates the gradient G of the magnetic field (total derivative of B) at the specified point from all member Coils. Uses internally defined defaultPrecisionFactor.

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the field is calculated.
--------------------	---

Returns

3D Cartesian vector which represents vector potential A.

4.2.2.10 computeEFieldVector()

```
vec3::Vector3 CoilGroup::computeEFieldVector (
    vec3::Vector3 pointVector ) const
```

Calculates the amplitude vector of electric field E in sinusoidal steady-state at the specified point from all member Coils. Uses internally defined defaultPrecisionFactor.

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the field is calculated.
--------------------	---

Returns

3D Cartesian vector which represents vector potential A.

4.2.2.11 computeForceOnDipoleMoment()

```
std::pair< vec3::Vector3, vec3::Vector3 > CoilGroup::computeForceOnDipoleMoment (
    vec3::Vector3 pointVector,
    vec3::Vector3 dipoleMoment ) const
```

Calculates force F and torque T between a coil and magnetostatic object with a dipole moment. Uses precision internally defined by default `PrecisionCPU`.

This method can prove particularly useful for approximating the force and torque of [CoilGroup](#) on a coil which is sufficiently far apart, or when the coil is very small. It can also be useful in particle simulations where the magnetic dipole moment is not negligible

Parameters

<i>pointVector</i>	Radius vector from the origin to the point where the magnetic dipole is located.
<i>dipoleMoment</i>	Magnetic dipole moment vector of a secondary coil or another object (magnet, particle).

Returns

Pair of Cartesian vectors which represent force (first) and torque (second).

4.2.2.12 computeForceTorque()

```
std::pair< vec3::Vector3, vec3::Vector3 > CoilGroup::computeForceTorque (
    const Coil & secondary,
    PrecisionFactor precisionFactor = PrecisionFactor(),
    ComputeMethod computeMethod = CPU_ST ) const
```

Calculates the force F and torque T on a provided coil from the rest of the member coils. Uses separate [CoilPairArguments](#) for every pair of coils, generated with the given [PrecisionFactor](#).

Using `CPU_MT` compute method is highly advisable, especially for higher precision factors. GPU is a good option when an error of $1e-5$ is good enough for the application (usual error of order $1e-6$). `MTD` is used if the number of coils is two or more times the number of threads and `CPU_MT` is selected.

Parameters

<i>secondary</i>	The coil represented with a number of points for which the magnetic field is calculated. Can be a member of CoilGroup or another Coil .
<i>precisionFactor</i>	Determines the precision of given calculation.
<i>computeMethod</i>	Three calculation options: <code>CPU_ST</code> , <code>CPU_MT</code> , and <code>GPU</code> . <code>CPU_ST</code> is default.

Returns

Pair of Cartesian vectors which represent force (first) and torque (second).

4.2.2.13 computeMutualInductance()

```
double CoilGroup::computeMutualInductance (
    const Coil & secondary,
```

```
PrecisionFactor precisionFactor = PrecisionFactor(),
ComputeMethod computeMethod = CPU_ST ) const
```

Calculates the mutual inductance M between a provided coil and the rest of the member coils. Uses separate [CoilPairArguments](#) for every pair of coils, generated with the given [PrecisionFactor](#).

Using CPU_MT compute method is highly advisable, especially for higher precision factors. GPU is a good option when an error of $1e-5$ is good enough for the application (usual error of order $1e-6$). MTD is used if the number of coils is two or more times the number of threads and CPU_MT is selected.

Parameters

<i>secondary</i>	The coil represented with a number of points for which the potential is calculated. Can be a member of CoilGroup or another Coil .
<i>precisionFactor</i>	Determines the precision of given calculation.
<i>computeMethod</i>	Three calculation options: CPU_ST, CPU_MT, and GPU. CPU_ST is default.

Returns

Mutual inductance between the system and secondary coil.

4.2.2.14 removeCoil()

```
void CoilGroup::removeCoil (
    size_t index )
```

Removes the [Coil](#) at the selected index from the [CoilGroup](#).

Parameters

<i>index</i>	Index within the CoilGroup of the Coil that is being removed.
--------------	---

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

- /home/davor/C++/Coil-Evolution/src/CoilGroup/CoilGroup.h
- /home/davor/C++/Coil-Evolution/src/CoilGroup/CoilGroup.cxx
- /home/davor/C++/Coil-Evolution/src/CoilGroup/Fields/CalculateFieldsGPU.cxx
- /home/davor/C++/Coil-Evolution/src/CoilGroup/Fields/CalculateFieldsMT.cxx
- /home/davor/C++/Coil-Evolution/src/CoilGroup/Fields/CalculateFieldsMTD.cxx
- /home/davor/C++/Coil-Evolution/src/CoilGroup/Fields/ComputeFields.cxx
- /home/davor/C++/Coil-Evolution/src/CoilGroup/ForceAndTorque/CalculateForceGPU.cxx
- /home/davor/C++/Coil-Evolution/src/CoilGroup/ForceAndTorque/CalculateForceMTD.cxx
- /home/davor/C++/Coil-Evolution/src/CoilGroup/ForceAndTorque/ComputeForce.cxx
- /home/davor/C++/Coil-Evolution/src/CoilGroup/MInductance/CalculateMInductanceGPU.cxx
- /home/davor/C++/Coil-Evolution/src/CoilGroup/MInductance/CalculateMInductanceMTD.cxx
- /home/davor/C++/Coil-Evolution/src/CoilGroup/MInductance/ComputeMInductance.cxx
- /home/davor/C++/Coil-Evolution/src/CoilGroup/Utils/GPUArgumentGeneration.cxx

4.3 CoilPairArguments Struct Reference

Structure used to store precision data (block and increment count) for a system of two coils. Used for custom precision arguments in interaction calculations.

```
#include <Coil.h>
```

Collaboration diagram for CoilPairArguments:

4.4 vec3::Matrix3 Class Reference

Represents a rank 2 tensor of dimension 3, which is represented as a square matrix.

```
#include <Tensor.h>
```

Public Member Functions

- [Matrix3](#) ()
Default constructor, creates a null matrix.
- [Matrix3](#) (double xx, double xy, double xz, double yx, double yy, double yz, double zx, double zy, double zz)
Creates matrix with components ((xx, xy, xz), (yx, yy, yz), (zx, zy, zz)).
- double [det](#) () const
Returns the determinant of the determinant of a square matrix.
- [Matrix3](#) **operator+** (const [Matrix3](#) &mat) const
- [Matrix3](#) **operator+=** (const [Matrix3](#) &mat)
- void **operator*=** (double multiplier)
- [Matrix3](#) **operator*** (double multiplier) const
- [Matrix3](#) **operator*** (const [Matrix3](#) &mat) const
- [Vector3](#) **operator*** (const [Vector3](#) &vec) const
- [operator std::string](#) () const
Generates a string object with all components of the 3x3 Matrix.

Public Attributes

- double **xx**
- double **xy**
- double **xz**
- double **yx**
- double **yy**
- double **yz**
- double **zx**
- double **zy**
- double **zz**

4.4.1 Detailed Description

Represents a rank 2 tensor of dimension 3, which is represented as a square matrix.

Basic operations are supported, such as matrix addition (+, +=), multiplication by a scalar (*, *=), [Vector3](#) transformation (*), matrix multiplication (*), and determinant calculation ([det\(\)](#)).

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

- /home/davor/C++/Coil-Evolution/src/Tensor/Tensor.h
- /home/davor/C++/Coil-Evolution/src/Tensor/Matrix/Matrix3.cxx

4.5 vec3::Matrix3Array Class Reference

Represents `std::vector<vec3::Matrix3>` for easier handling and additional features. Allows only `xx`, `xy`, `xz`, `yx`, `yy`, `yz`, `zx`, `zy`, or `zz` components, and [det\(\)](#) values, to be extracted to a `std::vector`.

```
#include <Tensor.h>
```

Public Member Functions

- [Matrix3Array](#) ()
Default constructor, creates an empty encapsulated `std::vector<vec3::Matrix3>`.
- [Matrix3Array](#) (size_t initSize)
Creates an encapsulated `std::vector<vec3::Matrix3>` of given size.
- [Matrix3Array](#) (const std::vector< [Matrix3](#) > &matrixArray)
Creates an encapsulated `std::vector<vec3::Matrix3>` filled with given values.
- void [append](#) (const [Matrix3](#) &appendedMatrix3)
Applies `std::vector push_back` with the given [Matrix3](#).
- void [append](#) (double [xx](#), double [xy](#), double [xz](#), double [yx](#), double [yy](#), double [yz](#), double [zx](#), double [zy](#), double [zz](#))
Applies `std::vector emplace_back` with given 9 values ((`xx`, `xy`, `xz`), (`yx`, `yy`, `yz`), (`zx`, `zy`, `zz`)).
- void [reserve](#) (size_t reserveSize)
Applies `std::vector reserve` with the provided size for faster append operations.
- void [resize](#) (size_t newSize)
Applies `std::vector resize` with the provided size.
- void [clear](#) ()
Returns the size of encapsulated `std::vector`.
- size_t [size](#) () const
- std::vector< [Matrix3](#) > &[getItems](#) ()
Returns a reference to encapsulated `std::vector<vec3::Matrix3>`.
- std::vector< double > [xx](#) () const
Returns a `std::vector<double>` of only `xx` components of [Matrix3](#).
- std::vector< double > [xy](#) () const
Returns a `std::vector<double>` of only `xy` components of [Matrix3](#).
- std::vector< double > [xz](#) () const
Returns a `std::vector<double>` of only `xz` components of [Matrix3](#).
- std::vector< double > [yx](#) () const
Returns a `std::vector<double>` of only `yx` components of [Matrix3](#).

- `std::vector< double > yy () const`
Returns a `std::vector<double>` of only yy components of [Matrix3](#).
- `std::vector< double > yz () const`
Returns a `std::vector<double>` of only yz components of [Matrix3](#).
- `std::vector< double > zx () const`
Returns a `std::vector<double>` of only zx components of [Matrix3](#).
- `std::vector< double > zy () const`
Returns a `std::vector<double>` of only zy components of [Matrix3](#).
- `std::vector< double > zz () const`
Returns a `std::vector<double>` of only zz components of [Matrix3](#).
- `std::vector< double > det () const`
Returns a `std::vector<double>` of determinants of [Matrix3](#).
- [Matrix3](#) & `operator[]` (size_t index)
- const [Matrix3](#) & `operator[]` (size_t index) const
- [Matrix3Array](#) & `operator+=` (const [Matrix3](#) &appendedMatrix3)
- `operator std::string () const`
Generates a string object from all elements of encapsulated `std::vector`.

4.5.1 Detailed Description

Represents `std::vector<vec3::Matrix3>` for easier handling and additional features. Allows only xx, xy, xz, yx, yy, yz, zx, zy, or zz components, and `det()` values, to be extracted to a `std::vector`.

A reference to the encapsulated `std::vector<vec3::Matrix3>` can be retrieved for faster C++ calculations. Basic `std::vector` functionality is implemented (reserve, resize, clear, size) with a python inspired append method for adding elements. Elements can also be added with +=. Reduces memory use in Python.

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

- `/home/davor/C++/Coil-Evolution/src/Tensor/Tensor.h`
- `/home/davor/C++/Coil-Evolution/src/Tensor/Matrix/Matrix3Array.cxx`

4.6 PrecisionArguments Struct Reference

Structure used to store precision data (block and increment count) for an individual [Coil](#).

```
#include <Coil.h>
```

Public Member Functions

- [PrecisionArguments](#) ()
Default constructor, sets all blocks to 1, and all increments to default quadrature value, currently 20.
- [PrecisionArguments](#) (int angularBlocks, int thicknessBlocks, int lengthBlocks, int angularIncrements, int thicknessIncrements, int lengthIncrements)
Takes 6 integer values specifying the number of blocks and increments assigned to each layer.
- `operator std::string () const`

Static Public Member Functions

- static [PrecisionArguments](#) [getCoilPrecisionArgumentsCPU](#) (const [Coil](#) &coil, [PrecisionFactor](#) precisionFactor)
Returns results of ordinary (CPU) increment balancing algorithm for one coil.
- static [PrecisionArguments](#) [getCoilPrecisionArgumentsGPU](#) (const [Coil](#) &coil, [PrecisionFactor](#) precisionFactor)
Returns results of specialised GPU increment balancing algorithm for one coil.
- static [PrecisionArguments](#) [getSecondaryCoilPrecisionArgumentsGPU](#) (const [Coil](#) &coil, [PrecisionFactor](#) precisionFactor)
Usually only used for [CoilGroup](#) computeAll MInductance and Force. Returns results of specialised GPU increment balancing algorithm for secondary coil.

Public Attributes

- int **angularBlocks**
- int **thicknessBlocks**
- int **lengthBlocks**
- int **angularIncrements**
- int **thicknessIncrements**
- int **lengthIncrements**

4.6.1 Detailed Description

Structure used to store precision data (block and increment count) for an individual [Coil](#).

There are 3 integration layers and thus 6 values arranged in 3 block-increment pairs. The number of blocks is the number of sub-intervals in integration and increments determine quadrature order

4.6.2 Constructor & Destructor Documentation

4.6.2.1 PrecisionArguments()

```
PrecisionArguments::PrecisionArguments (
    int angularBlocks,
    int thicknessBlocks,
    int lengthBlocks,
    int angularIncrements,
    int thicknessIncrements,
    int lengthIncrements ) [explicit]
```

Takes 6 integer values specifying the number of blocks and increments assigned to each layer.

Angular increments are along phi, thickness along a, and length along b (length is not utilised)

4.6.3 Member Function Documentation

4.6.3.1 getCoilPrecisionArgumentsCPU()

```
PrecisionArguments PrecisionArguments::getCoilPrecisionArgumentsCPU (
    const Coil & coil,
    PrecisionFactor precisionFactor ) [static]
```

Returns results of ordinary (CPU) increment balancing algorithm for one coil.

The CPU can divide the interval of integration into multiple blocks with

Parameters

<i>coil</i>	Reference to the coil for which PrecisionArguments are generated
<i>precisionFactor</i>	Relative precision, determines the total number of increments

4.6.3.2 getCoilPrecisionArgumentsGPU()

```
PrecisionArguments PrecisionArguments::getCoilPrecisionArgumentsGPU (
    const Coil & coil,
    PrecisionFactor precisionFactor ) [static]
```

Returns results of specialised GPU increment balancing algorithm for one coil.

The GPU uses only 1 block and a maximum of GPU_INCREMENTS increments per layer. Precision factor definition may therefore be invalid as less increments are assigned than specified.

Parameters

<i>coil</i>	Reference to the coil for which PrecisionArguments are generated
<i>precisionFactor</i>	Relative precision, determines the total number of increments

4.6.3.3 getSecondaryCoilPrecisionArgumentsGPU()

```
PrecisionArguments PrecisionArguments::getSecondaryCoilPrecisionArgumentsGPU (
    const Coil & coil,
    PrecisionFactor precisionFactor ) [static]
```

Usually only used for [CoilGroup](#) computeAll MInductance and Force. Returns results of specialised GPU increment balancing algorithm for secondary coil.

The GPU uses only 1 block and a maximum of GPU_INCREMENTS increments per layer. Precision factor definition may therefore be invalid as less increments are assigned than specified.

Parameters

<i>coil</i>	Secondary coil which is represented as a number of points
<i>precisionFactor</i>	Relative precision, determines the total number of increments

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

- /home/davor/C++/Coil-Evolution/src/Coil/Coil.h
- /home/davor/C++/Coil-Evolution/src/Coil/PrecisionArguments/PrecisionArguments.cxx

4.7 PrecisionFactor Struct Reference

Structure used to represent universal calculation precision. A custom precision measure from interval [1.0, 15.0]. Increasing the factor by 1.0 doubles the performance.

```
#include <Coil.h>
```

Public Member Functions

- [PrecisionFactor](#) ()
Default constructor, sets relativePrecision to 5.0.
- [PrecisionFactor](#) (double relativePrecision)
Sets relativePrecision to the given double value.
- **operator std::string** () const

Public Attributes

- double **relativePrecision**

4.7.1 Detailed Description

Structure used to represent universal calculation precision. A custom precision measure from interval [1.0, 15.0]. Increasing the factor by 1.0 doubles the performance.

Choosing a forbidden value results in the default value of 5.0, the same one used in the default constructor. The total number of increments, where k is the number of integration dimensions, m the base number of increments, and p the relativePrecision, is given as $m^k * 2^{(p-1)}$. Currently, $m = 10$ for both CPU and GPU.

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

- /home/davor/C++/Coil-Evolution/src/Coil/Coil.h
- /home/davor/C++/Coil-Evolution/src/Coil/PrecisionArguments/PrecisionFactor.cxx

4.8 vec3::Triplet Class Reference

Represents a general ordered sequence (tuple) with 3 elements.

```
#include <Tensor.h>
```

Public Member Functions

- [Triplet](#) ()
Default constructor, returns a tuple of zeros (0, 0, 0).
- [Triplet](#) (double first, double second, double third)
Creates a 3-tuple with elements (first, second, third).
- [operator std::string](#) () const
Generates a string object with the values of the triplet.

Public Attributes

- double **first**
- double **second**
- double **third**

4.8.1 Detailed Description

Represents a general ordered sequence (tuple) with 3 elements.

Used for representing [Vector3](#) data in different forms, such as cylindrical and spherical coordinates, which are not apt for proper calculations, but are useful for intermediate calculations.

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

- /home/davor/C++/Coil-Evolution/src/Tensor/Tensor.h
- /home/davor/C++/Coil-Evolution/src/Tensor/Vector/Vector3.cxx

4.9 vec3::Vector3 Class Reference

Represents a rank 1 tensor of dimension 3, which is a member of the oriented Euclidean vector space. It is commonly referred to as a 3D (Cartesian) Vector.

```
#include <Tensor.h>
```

Public Member Functions

- [Vector3](#) ()
Default constructor, creates a null vector (0, 0, 0).
- [Vector3](#) (double x, double y, double z)
Creates a vector with coordinates (x, y, z).
- [Vector3 operator+](#) (const [Vector3](#) &otherVec) const
- [Vector3 operator+=](#) (const [Vector3](#) &otherVec)
- [Vector3 operator-](#) (const [Vector3](#) &otherVec) const
- [Vector3 operator-=](#) (const [Vector3](#) &otherVec)
- [Vector3 operator*](#) (double multiplier) const
- [Vector3 operator*=](#) (double multiplier)
- double [abs](#) () const
Returns the magnitude (Euclidean norm) of the vector.
- [Triplet getAsCylindricalCoords](#) () const
Returns a [Triplet](#) representing the vector in cylindrical coordinates (z, r, phi).
- [Triplet getAsSphericalCoords](#) () const
Returns a [Triplet](#) representing the vector in spherical coordinates (z, r, phi).
- [operator std::string](#) () const
Generates a string object with all components of the 3D Vector.

Static Public Member Functions

- static double `scalarProduct` (`Vector3` vector1, `Vector3` vector2)
Returns the inner product of two `Vector3` objects.
- static `Vector3` `crossProduct` (`Vector3` vector1, `Vector3` vector2)
Returns the vector product of two `Vector3` objects.
- static `Vector3` `getFromCylindricalCoords` (double z, double r, double phi)
Creates a `Vector3` from cylindrical coordinates (z, r, phi), phi [0, 2PI].
- static `Vector3` `getFromSphericalCoords` (double r, double theta, double phi)
Creates a `Vector3` from spherical coordinates (r,theta, phi), theta [0, PI], phi [0, 2PI].

Public Attributes

- double `x`
- double `y`
- double `z`

4.9.1 Detailed Description

Represents a rank 1 tensor of dimension 3, which is a member of the oriented Euclidean vector space. It is commonly referred to as a 3D (Cartesian) Vector.

Basic operations are supported such as addition (+, +=), subtraction (-, -=), multiplication (*, *=), and magnitude (`abs()`). The inner (dot) product and vector (cross) product are defined for any two `Vector3` objects. For enhanced flexibility, the vector can be defined and obtained in spherical and cylindrical coordinates.

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

- /home/davor/C++/Coil-Evolution/src/Tensor/Tensor.h
- /home/davor/C++/Coil-Evolution/src/Tensor/Vector/Vector3.cxx

4.10 `vec3::Vector3Array` Class Reference

Represents `std::vector<vec3::Vector3>` for easier handling and additional features. Allows only x, y, or z components, as well as `abs()` values, to be extracted to `std::vector<double>`.

```
#include <Tensor.h>
```

Public Member Functions

- [Vector3Array](#) ()
Default constructor, creates an empty encapsulated `std::vector<vec3::Vector3>`.
- [Vector3Array](#) (size_t initSize)
Creates an encapsulated `std::vector<vec3::Vector3>` of given size.
- [Vector3Array](#) (const std::vector< [Vector3](#) > &vectorArray)
Creates an encapsulated `std::vector<vec3::Vector3>` filled with given values.
- void [append](#) (const [Vector3](#) &appendedVector3)
Applies `std::vector push_back` with the given [Vector3](#).
- void [append](#) (double x, double y, double z)
Applies `std::vector emplace_back` with given 3 values (x, y, z).
- void [reserve](#) (size_t reserveSize)
Applies `std::vector reserve` with the provided size for faster append operations.
- void [resize](#) (size_t newSize)
Applies `std::vector resize` with the provided size.
- void [clear](#) ()
Applies `std::vector clear`.
- size_t [size](#) () const
Returns the size of encapsulated `std::vector`.
- std::vector< [Vector3](#) > & [getItems](#) ()
Returns a reference to encapsulated `std::vector<vec3::Vector3>`.
- std::vector< double > [x](#) () const
Returns a `std::vector<double>` of only x components of [Vector3](#).
- std::vector< double > [y](#) () const
Returns a `std::vector<double>` of only y components of [Vector3](#).
- std::vector< double > [z](#) () const
Returns a `std::vector<double>` of only z components of [Vector3](#).
- std::vector< double > [abs](#) () const
Returns a `std::vector<double>` of magnitudes of [Vector3](#).
- [Vector3](#) & [operator\[\]](#) (size_t index)
- const [Vector3](#) & [operator\[\]](#) (size_t index) const
- [Vector3Array](#) & [operator+=](#) (const [Vector3](#) &appendedVector3)
- [operator std::string](#) () const
Generates a string object from all elements of encapsulated `std::vector`.

4.10.1 Detailed Description

Represents `std::vector<vec3::Vector3>` for easier handling and additional features. Allows only x, y, or z components, as well as [abs\(\)](#) values, to be extracted to `std::vector<double>`.

A reference to the encapsulated `std::vector<vec3::Vector3>` can be retrieved for faster C++ calculations. Basic `std::vector` functionality is implemented (reserve, resize, clear, size) with a python inspired append method for adding elements. Elements can also be added with +=. Reduces memory use in Python.

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

- /home/davor/C++/Coil-Evolution/src/Tensor/Tensor.h
- /home/davor/C++/Coil-Evolution/src/Tensor/Vector/Vector3Array.cxx

Index

Benchmark, [5](#)

Coil, [11](#)

- [computeAllAPotentialVectors](#), [15](#), [17](#)
- [computeAllBFieldVectors](#), [17](#), [18](#)
- [computeAllBGradientMatrices](#), [18](#)
- [computeAllEFieldVectors](#), [19](#)
- [computeAllForceTorqueArrangements](#), [20](#)
- [computeAllMutualInductanceArrangements](#), [21](#)
- [computeAndSetSelfInductance](#), [21](#)
- [computeAPotentialVector](#), [22](#)
- [computeBFieldVector](#), [23](#)
- [computeBGradientMatrix](#), [23](#), [24](#)
- [computeEFieldVector](#), [24](#)
- [computeForceOnDipoleMoment](#), [25](#)
- [computeForceTorque](#), [26](#)
- [computeMutualInductance](#), [27](#), [28](#)
- [computeSecondaryInducedVoltage](#), [28](#), [29](#)
- [setPositionAndOrientation](#), [29](#)

CoilGroup, [30](#)

- [computeAllAPotentialVectors](#), [32](#)
- [computeAllBFieldVectors](#), [32](#)
- [computeAllBGradientMatrices](#), [33](#)
- [computeAllEFieldVectors](#), [33](#)
- [computeAllForceTorqueArrangements](#), [34](#)
- [computeAllMutualInductanceArrangements](#), [34](#)
- [computeAPotentialVector](#), [35](#)
- [computeBFieldVector](#), [35](#)
- [computeBGradientMatrix](#), [36](#)
- [computeEFieldVector](#), [36](#)
- [computeForceOnDipoleMoment](#), [36](#)
- [computeForceTorque](#), [37](#)
- [computeMutualInductance](#), [37](#)
- [removeCoil](#), [38](#)

CoilPairArguments, [39](#)

Compare, [7](#)

[computeAllAPotentialVectors](#)

Coil, [15](#), [17](#)

CoilGroup, [32](#)

[computeAllBFieldVectors](#)

Coil, [17](#), [18](#)

CoilGroup, [32](#)

[computeAllBGradientMatrices](#)

Coil, [18](#)

CoilGroup, [33](#)

[computeAllEFieldVectors](#)

Coil, [19](#)

CoilGroup, [33](#)

[computeAllForceTorqueArrangements](#)

Coil, [20](#)

CoilGroup, [34](#)

[computeAllMutualInductanceArrangements](#)

Coil, [21](#)

CoilGroup, [34](#)

[computeAndSetSelfInductance](#)

Coil, [21](#)

[computeAPotentialVector](#)

Coil, [22](#)

CoilGroup, [35](#)

[computeBFieldVector](#)

Coil, [23](#)

CoilGroup, [35](#)

[computeBGradientMatrix](#)

Coil, [23](#), [24](#)

CoilGroup, [36](#)

[computeEFieldVector](#)

Coil, [24](#)

CoilGroup, [36](#)

[computeForceOnDipoleMoment](#)

Coil, [25](#)

CoilGroup, [36](#)

[computeForceTorque](#)

Coil, [26](#)

CoilGroup, [37](#)

[computeMutualInductance](#)

Coil, [27](#), [28](#)

CoilGroup, [37](#)

[computeSecondaryInducedVoltage](#)

Coil, [28](#), [29](#)

[getCoilPrecisionArgumentsCPU](#)

PrecisionArguments, [42](#)

[getCoilPrecisionArgumentsGPU](#)

PrecisionArguments, [43](#)

[getSecondaryCoilPrecisionArgumentsGPU](#)

PrecisionArguments, [43](#)

Legendre, [8](#)

PrecisionArguments, [41](#)

[getCoilPrecisionArgumentsCPU](#), [42](#)

[getCoilPrecisionArgumentsGPU](#), [43](#)

[getSecondaryCoilPrecisionArgumentsGPU](#), [43](#)

PrecisionArguments, [42](#)

PrecisionFactor, [44](#)

[removeCoil](#)

CoilGroup, [38](#)

[setPositionAndOrientation](#)

Coil, [29](#)

Test, [8](#)

threadPool, [9](#)

vec3, [10](#)

vec3::Matrix3, [39](#)

vec3::Matrix3Array, [40](#)

vec3::Triplet, [44](#)

vec3::Vector3, [45](#)

vec3::Vector3Array, [46](#)