TensorUtils

Generated by Doxygen 1.9.3

1 TensorUtils Version 0.1 1

1.1 Introduction . . . 1

1.2 Compile . . . 2

1.3 Installation (UBUNTU) . . . 2

1.4 Usage without installation / Installation with user-defined paths . . . 3

1.5 Examples . . . 4

1.6 Error Handling . . . 6

1.7 License . . . 7

2 Module Index 9

2.1 Modules . . . 9

3 Hierarchical Index 11

3.1 Class Hierarchy . . . 11

4 Class Index 13

4.1 Class List . . . 13

5 File Index 15

5.1 File List . . . 15

6 Module Documentation 17

6.1 TensorUtils . . . 17

6.1.1 Detailed Description . . . 18

6.1.2 Typedef Documentation . . . 18

6.1.2.1 tensor . . . 18

6.2 ErrorHandler . . . 18

6.2.1 Detailed Description . . . 18

7 Class Documentation 19

7.1 TensorUtils::ErrorHandler::RankMismatch Class Reference . . . 19

7.1.1 Detailed Description . . . 20

7.2 TensorUtils::ErrorHandler::ShapeMismatch Class Reference . . . 20

7.2.1 Detailed Description . . . 21

7.3 TensorUtils::TensorBase< T > Class Template Reference . . . 21

7.3.1 Detailed Description . . . 24

7.3.2 Constructor & Destructor Documentation . . . 24

7.3.2.1 TensorBase() [1/3] . . . 24

7.3.2.2 TensorBase() [2/3] . . . 24

7.3.2.3 TensorBase() [3/3] . . . 24

7.3.3 Member Function Documentation . . . 26

7.3.3.1 add() . . . 26

7.3.3.2 alloc() [1/2] . . . 26

7.3.3.3 alloc() [2/2] . . . 27

7.3.3.4 arange() 27
7.3.3.5 assign() 28
7.3.3.6 clear() 28
7.3.3.7 divide() 28
7.3.3.8 dot() 29
7.3.3.9 init() 29
7.3.3.10 minus() 30
7.3.3.11 multiply() 30
7.3.3.12 operator()() 31
7.3.3.13 operator∗() 31
7.3.3.14 operator∗=() 32
7.3.3.15 operator+() 32
7.3.3.16 operator+=() 32
7.3.3.17 operator-() 33
7.3.3.18 operator-=() 33
7.3.3.19 operator/() 33
7.3.3.20 operator/=() 34
7.3.3.21 operator<<() 34
7.3.3.22 operator=() [1/2] 34
7.3.3.23 operator=() [2/2] 35
7.3.3.24 operator>>() 35
7.3.3.25 plus() 35
7.3.3.26 print() 36
7.3.3.27 product() 36
7.3.3.28 quotient() 36
7.3.3.29 read() 37
7.3.3.30 reshape() 38
7.3.3.31 slice() 38
7.3.3.32 substract() 39
7.3.3.33 transpose() 39
7.3.3.34 write() [1/2] 40
7.3.3.35 write() [2/2] 41
7.3.4 Friends And Related Function Documentation 41
7.3.4.1 operator∗ 41
7.3.5 Member Data Documentation 42
7.3.5.1 incr 42
7.3.5.2 shape 42
7.4 TensorUtils::TensorDerived< T, N > Class Template Reference 43
7.4.1 Detailed Description 44
7.5 TensorUtils::TensorDerived< T,-1 > Class Template Reference 45
7.5.1 Detailed Description 46
7.6 TensorUtils::ErrorHandler::UnableToOpenFile Class Reference 46

7.6.1 Detailed Description 47

8 File Documentation 49

8.1 ErrorHandler.hpp 49

8.2 TensorBase.hpp 49

8.3 TensorDerived.hpp 51

8.4 TensorUtils.hpp 52

Index 53

Chapter 1

TensorUtils Version 0.1

Date

22.02.2022

Author

Christoph Widder

Copyright

TensorUtils is free software. See License for the terms of use. You are welcome to report any bugs to tensorutils@gmail.com. Please visit https://github.com/TensorUtils/TensorUtils for the latest version.

1.1 Introduction

TensorUtils presents a tensor class which is derived from std::vector<T>. It allows the usage of all std::vector routines, but has its own constructors. The tensor class allows to allocate, initialize, read and write tensors of floating or integral types up to rank 8. It provides text and binary file formats as well as element-wise operations with support for type conversions and chaining. The usage of this library might help to avoid memory leaks, segmentation faults, nested loops as well as error-prone index conversions. All methods are explicitly instantiated and stored in a shared library, which minimizes the compile time of your source code. Additionally, you will find routines to transpose, reshape and slice tensors as well as a generalized tensor product.

Supported floating point types are:

Data Type	Extension
float	.f32
double	.f64
long double	.f80

Supported integral types are:

Data Type	Extension	Data Type	Extension
signed char	.sc	unsigned char	.uc
short	.s	unsigned short	.us
int	.int	unsigned	.u
long	.l	unsigned long	.ul
long long	.ll	unsigned long long	.ull

The whole project is wrapped into the namespace TensorUtils from "TensorUtils.hpp". See the main class TensorUtils::TensorBase<T> for routines and examples. Although this base class is fully functional, it is recommended to use the derived class TensorUtils::TensorDerived<T,N> which allows you to use tensors of arbitrary rank as well as tensors with fixed rank. This will be helpful if you need distinct types for tensors of different ranks. More details on error-handling can be found in ErrorHandler.

Once the library is installed, you can use the following alias declarations for the class TensorUtils::TensorDerived<T,N>.

```
#include "TensorUtils.hpp"
int main()
{
    using namespace TensorUtils;
    tensor<double> foo;   // arbitrary rank:  TensorDerived<double,-1>
    tensor<double,4> bar; // fixed rank:      TensorDerived<double,4>
    return 0;
}
```

1.2 Compile

From within the project folder, type:

```
make
```

This will create a shared library at:

```
PATH_TO_TENSOR_UTILS/lib/Release/libtensor_utils.so
PATH_TO_TENSOR_UTILS/lib/Debug/libtensor_utilsd.so
```

1.3 Installation (UBUNTU)

If you don't want to install the library or if you don't want to use the default location, see Usage without installation / Installation with user

To install the library at the default locations "/usr/local/lib" and "/usr/local/include", type:

```
sudo make install
make clean
```

The header files are now installed as read only (444) in:

```
/usr/local/lib/tensor_utils
```

The shaed library is installed with read and execute permissions (555) at:

```
/usr/local/lib/libtensor_utils.so   # use this library for your release
/usr/local/lib/libtensor_utilsd.so  # use this library for debugging
```

To deinstall the library type:

```
sudo make uninstall
```

Include the header files:

```
-I/usr/local/include/tensor_utils
```

Link the shared library:

```
-L/usr/local/lib/
-ltensor_utils
-ltensor_utilsd
```

Your compile commands could look something like:

```
# debug
g++ -Wall -std=c++17 -fexceptions -g -I/usr/local/include/tensor_utils -c main.cpp -o obj/Debug/main.o
g++ -L/usr/local/lib -o bin/Debug/main obj/Debug/main.o   -ltensor_utilsd

# release
g++ -Wall -std=c++17 -fexceptions -O3 -I/usr/local/include/tensor_utils -c main.cpp -o obj/Release/main.o
g++ -L/usr/local/lib -o bin/Release/main obj/Release/main.o   -ltensor_utils
```

You are ready to run your executable!

1.4 Usage without installation / Installation with user-defined paths

Include the header files:

```
-I/PATH_TO_TENSOR_UTILS/include
```

Link the shared library:

```
-L/PATH_TO_TENSOR_UTILS/lib/Release
-L/PATH_TO_TENSOR_UTILS/lib/Debug
-ltensor_utils
-ltensor_utilsd
```

Your compile commands could look something like:

```
# debug
g++ -Wall -std=c++17 -fexceptions -g -I/PATH_TO_TENSOR_UTILS/include -c main.cpp -o obj/Debug/main.o
g++ -L/PATH_TO_TENSOR_UTILS/lib/Debug -o bin/Debug/main obj/Debug/main.o   -ltensor_utilsd

# release
g++ -Wall -std=c++17 -fexceptions -O3 -I/usr/local/include/tensor_utils -c main.cpp -o obj/Release/main.o
g++ -L/PATH_TO_TENSOR_UTILS/lib/Release -o bin/Release/main obj/Release/main.o   -ltensor_utils
```

To run your executable, you need to make sure that your operating system will find the shared library.

On UBUNTU:

```
# Release
cd PATH_TO_TENSOR_UTILS/lib/Release
export LD_LIBRARY_PATH="$(pwd)"

# Debug
cd PATH_TO_TENSOR_UTILS/lib/Debug
export LD_LIBRARY_PATH="$(pwd)"
```

You are ready to run your executable!

In order to install the library path permanently, create a .conf file in

```
/etc/ld.so.conf.d/your_config.conf
```

add the following paths in this file

```
PATH_TO_TENSOR_UTILS/lib/Release
PATH_TO_TENSOR_UTILS/lib/Debug
```

and update the cache:

```
sudo ldconfig
```

1.5 Examples

```
#include "TensorUtils.hpp"
#include <iostream>
using namespace std;
using namespace TensorUtils;
using namespace ErrorHandler;
void write_test_data()
{
    tensor<long double> A;
    A.alloc({2,3,5,7});
    A.arange();
    try{
        A.write("A.txt", ".");
        A.write("A.f32", ".");
        A.write("A.f64", ".");
        A.write("A.f80", ".");
        A.write("A.uc", ".");
        A.write("A.sc", ".");
        A.write("A.us", ".");
        A.write("A.s", ".");
        A.write("A.u", ".");
        A.write("A.int", ".");
        A.write("A.ul", ".");
        A.write("A.l", ".");
        A.write("A.ull", ".");
        A.write("A.ll", ".");
    } catch(exception &ex){cout«ex.what()«endl;}
}
int main()
{
    write_test_data();
    // CONSTRUCT, ALLOCATE AND INITIALIZE
    tensor<long double> A;
    tensor<double> B({2,3,5,7});
    tensor<float> C(B.shape, 1.0f);
    A.alloc(B.shape);
    A.alloc(B.shape, 2.0L);
    B.init(3.0);
    A=B=C;      // OK! Short for: B=C; A=B;
    A = vector<long double>(A.size(), 1.0L); // initialize from a vector
    if( A == vector<long double>(A.size(), 1.0L)) // bit-wise comparison
```

```
{
    //
}
A.arange(); // initialize with 0,1,2,3,... in lexicographical order.
long double raw_data[A.size()];
A » raw_data[0]; // copy data to array
A « raw_data[0]; // initialize from array
long double multi_array[2][3][5][7];
A » multi_array[0][0][0][0]; // copy data to multi-dimensional array
A « multi_array[0][0][0][0]; // initialize from multi-dimensional array
A.print();
if(!A.empty())
{
    A.clear();
}
//  READ AND WRITE
A.read("A.txt");    // text file
A.read("./A.f32");  // binary: float
A.read("./A.f64");  // binary: double
A.read("./A.f80");  // binary: long double
A.write("A.txt", "./");     // text file. If floating point type: write std::numeric_limits<T>
   significant digits
A.write("A.txt", "./", 10); // text file. If floating point type: write 10 significant digits
A.write("A.f32", ".");      // binary: float
A.write("A.f64", ".");      // binary: double
A.write("A.f80", ".");      // binary: long double
//  OPERATORS
B += B;
B -= B;
B = B+B;
B = B-B;
B *= 2.0;
B /= 2.0;     // use *= instead for best performance!
B = 2.0*B;
B = B*2.0;
B = B/2.0;    // use * instead for best performance!
if(A.shape == B.shape && B.shape == C.shape)
{
    // Operators will use implicit type conversion of components if necessary:
    A += B;
    A -= B;
    A = B-C;
    A = B+C;
    A = 2*A + 2*( (1.0/3)*B - C );
    C = (-2.0/3)*( 3*C - B ) + 2*A; // same but faster (operators return tensors of the smaller type)
}
else
{
    throw ShapeMismatch("Shape mismatch!");
}
// ACCESS ELEMENTS
int elem = 0;
for(size_t n0=0; n0<A.shape[0]; n0++)
{
    for(size_t n1=0; n1<A.shape[1]; n1++)
    {
        for(size_t n2=0; n2<A.shape[2]; n2++)
        {
            for(size_t n3=0; n3<A.shape[3]; n3++)
            {
                A(n0,n1,n2,n3) = elem;
                elem++;
            }
        }
    }
}
elem=0;
for(auto it=A.begin(); it!=A.end(); it++)
{
    *it = elem;
    elem++;
}
// SUBTENSORS
tensor<int> G({6,2,3,5,7});
A.alloc({2,3,5,7}, 1.0);
G.arange();
A.assign(G,{0},{1,1});
A.add(G, {}, {4});
A.substract(G, {}, {4});
A.multiply(2.0, {});
A.divide(0.5, {});
A = A.plus(G,{},{1});
A = A.minus(G,{},{1});
A = A.product(2.0,{0,0});
A = G.slice({1,1});
A = A.quotient(2.0,{1,2});
//  TRANSPOSE AND RESHAPE
```

```
    tensor<float> H({2,3,5,7},0);
    H.arange();
    H = H.transpose({3,1,2,0});
    H.reshape({7*3,5*2});
    //  GENERALIZED TENSOR PRODUCT
    tensor<double> X({2,3,5,7},1);
    tensor<double> Y({2,3,5,7},2);
    tensor<double> Z;
    Z = X.dot(Y,{-1,-2,-3,-4},{-1,-2,-3,-4});   // full contraction: Z is a scalar!
    Z = X.dot(Y,{1,2,3,4},{1,2,3,4});       // Hadamard product: Z has shape {2,3,5,7}
    Z = X.dot(Y,{1,2,3,4},{8,7,6,5});       // tensor product: Z has shape {2,3,5,7,7,5,3,2}
    Z = X.dot(Y,{1,2,3,4},{5,6,7,8},{1,2,4,6}); // compute sub-tensor of tensor-product
    X.alloc({2,3,7,7},1);
    Y.alloc({7,5,3,11},2);
    Z = X.dot(Y,{3,2,-5,-5},{-5,4,2,1});   // generalized tensor product: Z has shape {11,3,2,5}
    //  TENSORS WITH FIXED RANK AND DISTINGUISHABLE TYPES:
    //      In many situations you might want to keep the types of tensors with different rank
       distinguishable,
    //      i.e. to overload functions that depend on the rank of its arguments.
    //      Everything works exactly the same, but tensors have fixed ranks!
    tensor<double> E({2,3,5},1);
    tensor<float,4> F({2,3,5,7},0); // tensor with fixed rank 4
    try
    {
        F = E;              // throws
        F.alloc({2,3,5});   // throws
        F.alloc({2,3,5},0);  // throws
        tensor<long double,4> G({2,3,5});   // throws
        tensor<long double,4> H({2,3,5},0); // throws
    }
    catch(RankMismatch &ex)
    {
        //
    }
    E = F; // OK!
    //  ERROR HANDLING (see TensorUtils::ErrorHandling for more)
    //      Most error handling is enabled only for the debug-library libtensor_utilsd.so
    //      This will enable you to trace down any occurrence of invalid indices or shape mismatches.
    try
    {
        A(1,2,3,5);
    }
    catch(ShapeMismatch &ex)// wrong number of indices
    {
        cout « ex.what() « endl;
    }
    catch(out_of_range &ex) // at least one index is out of range
    {
        cout « ex.what() « endl;
    }
    try
    {
        A.read("./A.txt");
        A.read("./A.f32");
    }
    catch(UnableToOpenFile & ex) // probably the required file does not exist
    {
        throw ex;
    }
    catch(ShapeMismatch & ex)   // shape does not match data: corrupted file?
    {
        throw ex;
    }
    catch(exception & ex) // catch any other exception
    {
        throw ex;
    }
    return 0;
}
```

1.6 Error Handling

```
#include "TensorUtils.hpp"
#include <iostream>
using namespace std;
using namespace TensorUtils;
using namespace ErrorHandler;
int main()
{
    tensor<double> A;
    // READING FILES
    try
    {
        A.read("my_tensor.txt");
    }
    catch(UnableToOpenFile &ex) // unable to open file
    {
        cerr « ex.what() « endl;
    }
    catch(ShapeMismatch & ex)   // shape does not match data: corrupted file?
    {
        throw ex;
    }
    catch(exception & ex) // catch any other exception
    {
        throw ex;
    }
    // ACCESSING COMPONENTS
    A.alloc({2,3,5,7},1.0);
    try
    {
        A(1,2);         // OK! Returns A(1,2,0,0) by reference!
        A(0,0,0,0,0);   // too many indices: throws ShapeMismatch
        A(1,2,4,7);     // index out of range: throws std::out_of_range
    }
    catch(ShapeMismatch &ex) // more indices than expected!
    {
        cerr « ex.what() « endl;
    }
    catch(out_of_range &ex) // at least one index is out of range
    {
        cerr « ex.what() « endl;
    }
    // OPERATORS AND MEMBER FUNCTIONS
    tensor<double>          B({2,3,5,8},1.0);
    tensor<float>           C({2*3,5*7},1.0);
    tensor<long double>     D({},1.0); // scalar
    tensor<int,3>           E({3,5,7},1.0);
    tensor<unsigned long>   F({3,5,7},1.0);
    try
    {
        A += B; // different number of components: throws ShapeMismatch.
        A += C; // OK! Same number of elements, but different shapes!
        E = A;  // RankMismatch: unable to assign with a tensor of different rank!
        E = F;  // OK! Different types, but the ranks are the same.
        A = E;  // OK! A can have arbitrary rank.
        E.alloc({2,3,5,7}); // RankMismatch: E has a fixed rank!
        A.alloc({2,3,5,7},1.0);
        A.assign(B, {1,2}, {1,2});  // ShapeMismatch: assignment with sub-tensor of invalid shape.
        A.assign(C, {1,2}, {0});    // OK! Same number of elements.
        A.assign(C, {1,3}, {0});    // invalid index: throws std::out_of_range.
        F = F.transpose({0,2,1}); // OK! Swap last two axes.
        F = F.transpose({1,3,2}); // ShapeMismatch: Reshape must be a permutation of (0,1,...,N-1).
        C = A.dot(A, {1,2,3}, {1,2,3,4});   // ShapeMismatch: axes must have the same size as the shapes.
        C = A.dot(A, {1,2,3,4}, {5,6,7,8}, {0,0,0,7});  // invalid index: std::out_of_range.
    }
    catch(ShapeMismatch &ex)
    {
        cerr « ex.what() « endl;
    }
    catch(RankMismatch &ex)
    {
        cerr « ex.what() « endl;
    }
    catch(out_of_range &ex)
    {
        cerr « ex.what() « endl;
    }
    return 0;
}
```

1.7 License

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

TensorUtils 17

ErrorHandler 18

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

- std::runtime_error
  - TensorUtils::ErrorHandler::RankMismatch 19
  - TensorUtils::ErrorHandler::ShapeMismatch 20
  - TensorUtils::ErrorHandler::UnableToOpenFile 46
- std::vector
  - TensorUtils::TensorBase< T > 21
    - TensorUtils::TensorDerived< T, N > 43
    - TensorUtils::TensorDerived< T,-1 > 45

Chapter 4

Class Index

4.1 Class List

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

TensorUtils::ErrorHandler::RankMismatch
This error is thrown if any method would change the rank of a tensor with fixed rank. Inherits from std::runtime_error 19

TensorUtils::ErrorHandler::ShapeMismatch
This error is thrown if any tensor operation is called with invalid shapes or an invalid number of indices. Inherits from std::runtime_error 20

TensorUtils::TensorBase< T >
This is the main class of this project. It inherits from std::vector<T> and adds methods to make it a tensor 21

TensorUtils::TensorDerived< T, N >
This class defines a tensor with fixed rank N=0,1,... and inherits from TensorBase. The spezialization for N=-1 defines a tensor with mutable rank 43

TensorUtils::TensorDerived< T,-1 >
This class specialization defines a tensor with mutable rank and inherits from TensorBase . . . 45

TensorUtils::ErrorHandler::UnableToOpenFile
This error is thrown, if a file cannot be opened. Inherits from std::runtime_error 46

Chapter 5

File Index

5.1 File List

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

ErrorHandler.hpp 49
TensorBase.hpp 49
TensorDerived.hpp 51
TensorUtils.hpp 52

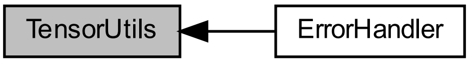
Chapter 6

Module Documentation

6.1 TensorUtils

This is the main namespace that wraps the entire implementation of this project.

Collaboration diagram for TensorUtils:



Modules

- ErrorHandler

  This namespace contains error handler classes that inherit from "std::runtime_error". Most error handling is enabled only for the debug library "libtensor_utilsd.so".

Classes

- class TensorUtils::TensorDerived< T, N >

  This class defines a tensor with fixed rank N=0,1,... and inherits from TensorBase. The spezialization for N=-1 defines a tensor with mutable rank.
- class TensorUtils::TensorDerived< T,-1 >

  This class specialization defines a tensor with mutable rank and inherits from TensorBase.

Typedefs

- template<class T , int N = -1>
  using TensorUtils::tensor = TensorDerived< T, N >

  Alias declaration for derived class "TensorDerived<T,N>", where "T" is the type of the components and "N" is the rank. "TensorDerived<T,N>" inherits all its functionality from the base class "TensorBase<T>".

6.1.1 Detailed Description

This is the main namespace that wraps the entire implementation of this project.

6.1.2 Typedef Documentation

6.1.2.1 tensor

```
template<class T , int N = -1>
using TensorUtils::tensor = typedef TensorDerived<T,N>
```

Alias declaration for derived class "TensorDerived<T,N>", where "T" is the type of the components and "N" is the rank. "TensorDerived<T,N>" inherits all its functionality from the base class "TensorBase<T>".

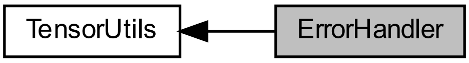
Construct tensors with arbitrary or fixed rank:

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<double> my_tensor;
    return 0;
}
```

6.2 ErrorHandler

This namespace contains error handler classes that inherit from "std::runtime_error". Most error handling is enabled only for the debug library "libtensor_utilsd.so".

Collaboration diagram for ErrorHandler:



Classes

- class TensorUtils::ErrorHandler::UnableToOpenFile

  This error is thrown, if a file cannot be opened. Inherits from std::runtime_error.
- class TensorUtils::ErrorHandler::ShapeMismatch

  This error is thrown if any tensor operation is called with invalid shapes or an invalid number of indices. Inherits from std::runtime_error.
- class TensorUtils::ErrorHandler::RankMismatch

  This error is thrown if any method would change the rank of a tensor with fixed rank. Inherits from std::runtime_error.

6.2.1 Detailed Description

This namespace contains error handler classes that inherit from "std::runtime_error". Most error handling is enabled only for the debug library "libtensor_utilsd.so".

TensorUtils provides error handling to trace down rank or shape mismatches, invalid indices and invalid file paths.

Chapter 7

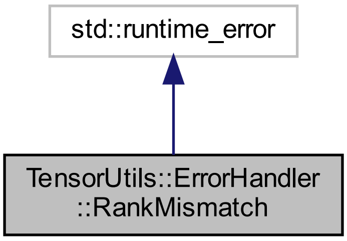
Class Documentation

7.1 TensorUtils::ErrorHandler::RankMismatch Class Reference

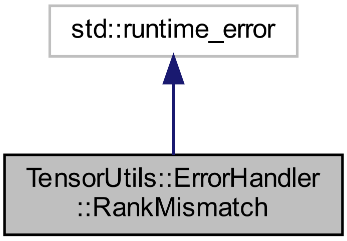
This error is thrown if any method would change the rank of a tensor with fixed rank. Inherits from std::runtime_error.

```
#include <ErrorHandler.hpp>
```

Inheritance diagram for TensorUtils::ErrorHandler::RankMismatch:



Collaboration diagram for TensorUtils::ErrorHandler::RankMismatch:



Public Member Functions

- **RankMismatch** (const std::string &what_arg)
  Constructor inherited from std::runtime_error.

7.1.1 Detailed Description

This error is thrown if any method would change the rank of a tensor with fixed rank. Inherits from std::runtime_error.

See ErrorHandler for details.

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

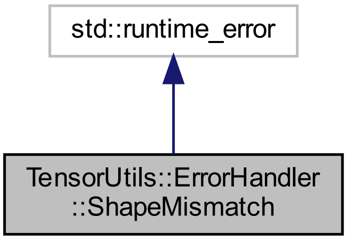
- ErrorHandler.hpp

7.2 TensorUtils::ErrorHandler::ShapeMismatch Class Reference

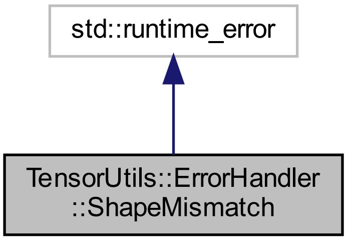
This error is thrown if any tensor operation is called with invalid shapes or an invalid number of indices. Inherits from std::runtime_error.

```
#include <ErrorHandler.hpp>
```

Inheritance diagram for TensorUtils::ErrorHandler::ShapeMismatch:



Collaboration diagram for TensorUtils::ErrorHandler::ShapeMismatch:



Public Member Functions

- **ShapeMismatch** (const std::string &what_arg)

  Constructor inherited from std::runtime_error.

7.2.1 Detailed Description

This error is thrown if any tensor operation is called with invalid shapes or an invalid number of indices. Inherits from std::runtime_error.

If an index is out of range, std::out_of_range is thrown instead. Invalid usage of tensors with fixed ranks have their own error class RankMismatch. See ErrorHandler for details.

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

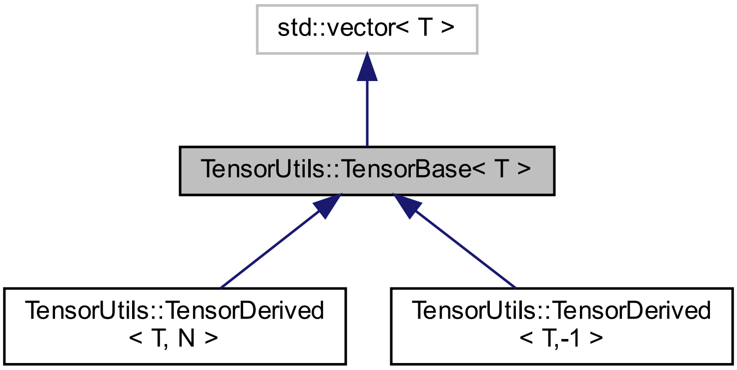
- ErrorHandler.hpp

7.3 TensorUtils::TensorBase< T > Class Template Reference

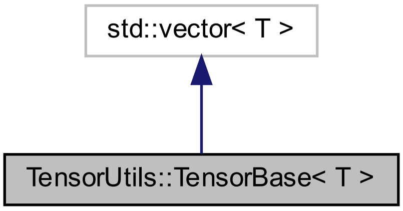
This is the main class of this project. It inherits from std::vector<T> and adds methods to make it a tensor.

```
#include <TensorBase.hpp>
```

Inheritance diagram for TensorUtils::TensorBase< T >:


std::vector< T >
TensorUtils::TensorBase< T >
TensorUtils::TensorDerived < T, N >
TensorUtils::TensorDerived < T,-1 >

Collaboration diagram for TensorUtils::TensorBase< T >:



Public Member Functions

- TensorBase ()
- TensorBase (const std::vector< size_t > &shape)
- TensorBase (const std::vector< size_t > &shape, const T &val)
- void alloc (const std::vector< size_t > &shape)
- void alloc (const std::vector< size_t > &shape, const T &val)
- void init (const T &val)
- void arange (T val=0)
- void clear ()
- void print ()
- void read (std::string path)
- void write (std::string oname, std::string folder)
- void write (std::string oname, std::string folder, int precision)
- TensorBase< T > transpose (const std::vector< unsigned > &axes)
- TensorBase< T > slice (const std::vector< size_t > &idx_at)
- TensorBase< T > & reshape (const std::vector< size_t > &shape)
- template<class T2 >
  TensorBase< T > dot (TensorBase< T2 > &rhs, const std::vector< int > &idx_lhs, const std::vector< int > &idx_rhs, const std::vector< size_t > &idx_at={})
- TensorBase< T > & operator= (const std::vector< T > &rhs)
- template<class T2 >
  TensorBase< T > & operator= (const TensorBase< T2 > &rhs)
- template<class T2 >
  TensorBase< T > & operator+= (const TensorBase< T2 > &rhs)
- template<class T2 >
  TensorBase< T > operator+ (const TensorBase< T2 > &rhs)
- template<class T2 >
  TensorBase< T > & operator-= (const TensorBase< T2 > &rhs)
- template<class T2 >
  TensorBase< T > operator- (const TensorBase< T2 > &rhs)
- TensorBase< T > & operator*= (const T &rhs)
- TensorBase< T > operator* (const T &rhs)
- TensorBase< T > & operator/= (const T &rhs)
- TensorBase< T > operator/ (const T &rhs)
- template<class T2 >
  TensorBase< T > & operator<< (T2 &rhs)

- template<class T2 >
  T2 & operator>> (T2 &rhs)
- template<class T2 >
  TensorBase< T > & assign (TensorBase< T2 > &rhs, const std::vector< size_t > &at_lhs={}, const std↩ ::vector< size_t > &at_rhs={})
- template<class T2 >
  TensorBase< T > & add (TensorBase< T2 > &rhs, const std::vector< size_t > &at_lhs={}, const std↩ ::vector< size_t > &at_rhs={})
- template<class T2 >
  TensorBase< T > & substract (TensorBase< T2 > &rhs, const std::vector< size_t > &at_lhs={}, const std::vector< size_t > &at_rhs={})
- TensorBase< T > & multiply (const T &rhs, const std::vector< size_t > &at_lhs={})
- TensorBase< T > & divide (const T &rhs, const std::vector< size_t > &at_lhs={})
- template<class T2 >
  TensorBase< T > plus (TensorBase< T2 > &rhs, const std::vector< size_t > &at_lhs={}, const std::vector< size_t > &at_rhs={})
- template<class T2 >
  TensorBase< T > minus (TensorBase< T2 > &rhs, const std::vector< size_t > &at_lhs={}, const std↩ ::vector< size_t > &at_rhs={})
- TensorBase< T > product (const T &rhs, const std::vector< size_t > &at_lhs={})
- TensorBase< T > quotient (const T &rhs, const std::vector< size_t > &at_lhs={})
- T & operator() (const std::vector< size_t > &indices)
- T & **operator()** (const std::vector< size_t * > &indices)

  See operator()(const std::vector<size_t> &).
- T & **operator()** ()

  See operator()(const std::vector<size_t> &).
- T & **operator()** (size_t n0)

  See operator()(const std::vector<size_t> &).
- T & **operator()** (size_t n0, size_t n1)

  See operator()(const std::vector<size_t> &).
- T & **operator()** (size_t n0, size_t n1, size_t n2)

  See operator()(const std::vector<size_t> &).
- T & **operator()** (size_t n0, size_t n1, size_t n2, size_t n3)

  See operator()(const std::vector<size_t> &).
- T & **operator()** (size_t n0, size_t n1, size_t n2, size_t n3, size_t n4)

  See operator()(const std::vector<size_t> &).
- T & **operator()** (size_t n0, size_t n1, size_t n2, size_t n3, size_t n4, size_t n5)

  See operator()(const std::vector<size_t> &).
- T & **operator()** (size_t n0, size_t n1, size_t n2, size_t n3, size_t n4, size_t n5, size_t n6)

  See operator()(const std::vector<size_t> &).
- T & **operator()** (size_t n0, size_t n1, size_t n2, size_t n3, size_t n4, size_t n5, size_t n6, size_t n7)

  See operator()(const std::vector<size_t> &).

Public Attributes

- std::vector< size_t > shape
- std::vector< size_t > incr

Friends

- TensorBase< T > operator* (const T &lhs, TensorBase< T > rhs)

7.3.1 Detailed Description

template<class T>
class TensorUtils::TensorBase< T >

This is the main class of this project. It inherits from std::vector<T> and adds methods to make it a tensor.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 TensorBase() [1/3]

```
template<class T >
TensorUtils::TensorBase< T >::TensorBase ( )
```

Empty constructor.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<double> foo;
    return 0;
}
```

7.3.2.2 TensorBase() [2/3]

```
template<class T >
TensorUtils::TensorBase< T >::TensorBase (
             const std::vector< size_t > & shape )
```

Constructor. Calls alloc(const std::vector<size_t> &).

Parameters

shape	Specifies the number of indices and their ranges.

```
#include "TensorUtils.hpp"
int main()
{
    // TensorUtils::tensor<double> foo({}); // invalid syntax: ambiguity with copy and move constructor!
    TensorUtils::tensor<double> foo({2,3,5,7});
    return 0;
}
```

7.3.2.3 TensorBase() [3/3]

```
template<class T >
TensorUtils::TensorBase< T >::TensorBase (
             const std::vector< size_t > & shape,
             const T & val )
```

Constructor. Calls alloc(const std::vector<size_t> &, const T&).

Parameters

shape	Specifies the number of indices and their ranges.
val	All components are initialized with this value.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<float> foo({}, 1.0); // scalar
    TensorUtils::tensor<float> bar({2,3,5,7}, 1.0);
    return 0;
}
```

7.3.3 Member Function Documentation

7.3.3.1 add()

```
template<class T >
template<class T2 >
TensorBase< T > & TensorUtils::TensorBase< T >::add (
            TensorBase< T2 > & rhs,
            const std::vector< size_t > & at_lhs = {},
            const std::vector< size_t > & at_rhs = {} )
```

Add a sub-tensor of `rhs` to a sub-tensor of this tensor. Number of components must match, else ErrorHandler::ShapeMismatch is thrown.

Parameters

rhs	Second operand.
at_lhs	Indices specifying the sub-tensor of the first operand.
at_rhs	Indices specifying the sub-tensor of the second operand.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    TensorUtils::tensor<float> bar({2*3,5,7},1.0);
    foo.add(bar, {1,2}, {5});
    return 0;
}
```

7.3.3.2 alloc() [1/2]

```
template<class T >
void TensorUtils::TensorBase< T >::alloc (
            const std::vector< size_t > & shape )
```

Allocates the necessary memory and initializes shape and incr accordingly. If an empty shape is received, the tensor is a scalar with exactly one component.

Parameters

shape	Used to initialize shape.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<float> foo;
    foo.alloc({2,3,5,7});
    return 0;
}
```

7.3.3.3 alloc() [2/2]

```
template<class T >
void TensorUtils::TensorBase< T >::alloc (
             const std::vector< size_t > & shape,
             const T & val )
```

Allocate memory and initialize all components. Calls alloc(const std::vector<size_t> &shape) and init(const T& val).

Parameters

shape	Used to initialize shape.
val	All components are initialized with this value.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<float> foo;
    foo.alloc({2,3,5,7}, 1.0);
    return 0;
}
```

7.3.3.4 arange()

```
template<class T >
void TensorUtils::TensorBase< T >::arange (
             T val = 0 )
```

Initialize all components with lexicographical enumeration.

Parameters

val	Value of first component.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<double> my_tensor({10,10});
    my_tensor.arange(1); // same as the following
    my_tensor(0,0)=1;
    my_tensor(0,1)=2;
    // ...
    my_tensor(9,9)=100;
    return 0;
}
```

7.3.3.5 assign()

```
template<class T >
template<class T2 >
TensorBase< T > & TensorUtils::TensorBase< T >::assign (
            TensorBase< T2 > & rhs,
            const std::vector< size_t > & at_lhs = {},
            const std::vector< size_t > & at_rhs = {} )
```

Assign a sub-tensor this tensor with a sub-tensor of `rhs`. Number of components must match, else ErrorHandler::ShapeMismatch is thrown.

Parameters

rhs	Second operand.
at_lhs	Indices specifying the sub-tensor of the first operand.
at_rhs	Indices specifying the sub-tensor of the second operand.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    TensorUtils::tensor<float> bar({2*3,5,7},1.0);
    foo.assign(bar, {1,2}, {5});
    return 0;
}
```

7.3.3.6 clear()

```
template<class T >
void TensorUtils::TensorBase< T >::clear ( )
```

Clears the memory and the member variables shape and incr.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<float> foo({2,3,5,7}, 1.0);
    foo.clear();
    return 0;
}
```

7.3.3.7 divide()

```
template<class T >
TensorBase< T > & TensorUtils::TensorBase< T >::divide (
            const T & rhs,
            const std::vector< size_t > & at_lhs = {} )
```

Divide a sub-tensor of this tensor with `rhs`.

Parameters

rhs	Second operand.
at_lhs	Indices specifying the sub-tensor of the first operand.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    foo.divide(5, {1,2});
    return 0;
}
```

7.3.3.8 dot()

```
template<class T >
template<class T2 >
TensorBase< T > TensorUtils::TensorBase< T >::dot (
              TensorBase< T2 > & rhs,
              const std::vector< int > & idx_lhs,
              const std::vector< int > & idx_rhs,
              const std::vector< size_t > & idx_at = {} )
```

Returns a generalized tensor product by value. Indices are represented by signed integers. The parameters `idx_lhs` and `idx_rhs` specify the indices for the two operands. Negative integers are summed over. Multiple occurrences of the same index performs element-wise multiplication (Hadamard product). Distinct indices perform the usual tensor product. It is possible to mix summation, element-wise multiplication and the usual tensor product as desired. The order of the return value can be set as desired and is given in increasing order of the resulting indices. Additionally, it is possible to compute only a sub-tensor of the final result by setting the parameter `idx_at`.

Parameters

rhs	Second operand.
idx_lhs	Indices of first operand represented by signed intergers.
idx_rhs	Indices of second operand represented by signed integers.
idx_at	Indices specifying the sub-tensor to be computed.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<double> X({2,3,5,7},1);
    TensorUtils::tensor<double> Y({2,3,5,7},2);
    TensorUtils::tensor<double> Z;
    Z = X.dot(Y,{-1,-2,-3,-4},{-1,-2,-3,-4});   // full contraction: Z is a scalar!
    Z = X.dot(Y,{1,2,3,4},{1,2,3,4});           // Hadamard product: Z has shape {2,3,5,7}
    Z = X.dot(Y,{1,2,3,4},{8,7,6,5});           // tensor product: Z has shape {2,3,5,7,7,5,3,2}
    Z = X.dot(Y,{1,2,3,4},{5,6,7,8}, {1,2,4,6}); // compute sub-tensor of tensor-product
    X.alloc({2,3,7,7},1);
    Y.alloc({7,5,3,11},2);
    Z = X.dot(Y,{3,2,-5,-5},{-5,4,2,1});    // generalized tensor product: Z has shape {11,3,2,5}
    return 0;
}
```

7.3.3.9 init()

```
template<class T >
void TensorUtils::TensorBase< T >::init (
              const T & val )
```

Parameters

val	All components are initialized with this value.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<double> foo({2,3,5,7});
    long double some_value = 1.0L;
    foo.init(some_value);
    return 0;
}
```

7.3.3.10 minus()

```
template<class T >
template<class T2 >
TensorBase< T > TensorUtils::TensorBase< T >::minus (
            TensorBase< T2 > & rhs,
            const std::vector< size_t > & at_lhs = {},
            const std::vector< size_t > & at_rhs = {} )
```

Return the difference of a sub-tensor of this tensor with a sub-tensor of `rhs`. Number of components must match, else ErrorHandler::ShapeMismatch is thrown.

Parameters

rhs	Second operand.
at_lhs	Indices specifying the sub-tensor of the first operand.
at_rhs	Indices specifying the sub-tensor of the second operand.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    TensorUtils::tensor<float> bar({2*3,5,7},1.0);
    TensorUtils::tensor<double> foobar;
    foobar = foo.minus(bar, {1,2}, {5}); // foobar has shape {5,7}
    return 0;
}
```

7.3.3.11 multiply()

```
template<class T >
TensorBase< T > & TensorUtils::TensorBase< T >::multiply (
            const T & rhs,
            const std::vector< size_t > & at_lhs = {} )
```

Multiply a sub-tensor of this tensor with `rhs`.

Parameters

rhs	Second operand.
at_lhs	Indices specifying the sub-tensor of the first operand.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    foo.multiply(5, {1,2});
    return 0;
}
```

7.3.3.12 operator()()

```
template<class T >
T & TensorUtils::TensorBase< T >::operator() (
            const std::vector< size_t > & indices )
```

Access a component or the first component of a sub-tensor.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    int elem = 0;
    for(size_t n0=0; n0<foo.shape[0]; n0++)
    {
        for(size_t n1=0; n1<foo.shape[1]; n1++)
        {
            for(size_t n2=0; n2<foo.shape[2]; n2++)
            {
                for(size_t n3=0; n3<foo.shape[3]; n3++)
                {
                    foo(n0,n1,n2,n3) = elem;
                    elem++;
                }
            }
        }
    }
    // Remember that TensorBase<T> inherits from std::vector<T>
    elem=0;
    for(auto it=foo.begin(); it!=foo.end(); it++)
    {
        *it = elem;
        elem++;
    }
    std::vector<size_t> index = {1,2,4,6};
    foo(index) = 5.0;
    std::vector<size_t*> index_ptr = { &index[0],&index[1],&index[2],&index[3] };
    foo(index_ptr) = 5.0;
    long double* ptr_to_subtensor = &foo({1,2});
    return 0;
}
```

7.3.3.13 operator*()

```
template<class T >
TensorBase< T > TensorUtils::TensorBase< T >::operator* (
            const T & rhs )
```

Scalar multiplication from the right.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    foo = foo*2;
    return 0;
}
```

7.3.3.14 operator*=()

```
template<class T >
TensorBase< T > & TensorUtils::TensorBase< T >::operator*= (
             const T & rhs )
```

Multiply this tensor with `rhs`.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    foo *= 2;
    return 0;
}
```

7.3.3.15 operator+()

```
template<class T >
template<class T2 >
TensorBase< T > TensorUtils::TensorBase< T >::operator+ (
             const TensorBase< T2 > & rhs )
```

Returns the sum of this tensor `with` rhs. Number of components must match, else ErrorHandler::ShapeMismatch is thrown.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    TensorUtils::tensor<float> bar({2,3,5,7},1.0);
    foo = foo + bar;
    bar.alloc({2*3,5*7},1.0);
    foo = foo + bar;
    return 0;
}
```

7.3.3.16 operator+=()

```
template<class T >
template<class T2 >
TensorBase< T > & TensorUtils::TensorBase< T >::operator+= (
             const TensorBase< T2 > & rhs )
```

Add the tensor `rhs`. Number of components must match, else ErrorHandler::ShapeMismatch is thrown.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    TensorUtils::tensor<float> bar({2,3,5,7},1.0);
    foo += bar;
    bar.alloc({2*3,5*7},1.0);
    foo += bar;
    return 0;
}
```

7.3.3.17 operator-()

```
template<class T >
template<class T2 >
TensorBase< T > TensorUtils::TensorBase< T >::operator- (
             const TensorBase< T2 > & rhs )
```

Returns the difference of this tensor `with` rhs. Number of components must match, else ErrorHandler::ShapeMismatch is thrown.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    TensorUtils::tensor<float> bar({2,3,5,7},1.0);
    foo = foo - bar;
    bar.alloc({2*3,5*7},1.0);
    foo = foo - bar;
    return 0;
}
```

7.3.3.18 operator-=()

```
template<class T >
template<class T2 >
TensorBase< T > & TensorUtils::TensorBase< T >::operator-= (
             const TensorBase< T2 > & rhs )
```

Substract the tensor `rhs`. Number of components must match, else ErrorHandler::ShapeMismatch is thrown.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    TensorUtils::tensor<float> bar({2,3,5,7},1.0);
    foo -= bar;
    bar.alloc({2*3,5*7},1.0);
    foo -= bar;
    return 0;
}
```

7.3.3.19 operator/()

```
template<class T >
TensorBase< T > TensorUtils::TensorBase< T >::operator/ (
             const T & rhs )
```

Element-wise division.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    foo = foo/2;
    return 0;
}
```

7.3.3.20 operator/=()

```
template<class T >
TensorBase< T > & TensorUtils::TensorBase< T >::operator/= (
              const T & rhs )
```

Divide this tensor with `rhs`.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    foo /= 2;
    return 0;
}
```

7.3.3.21 operator<<()

```
template<class T >
template<class T2 >
TensorBase< T > & TensorUtils::TensorBase< T >::operator<< (
              T2 & rhs )
```

Initialize this tensor from an array in lexicographical order. No error-handling!

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7});
    foo.arange();
    long double raw_data[2*3*5*7];
    foo » raw_data[0]; // copy data to array
    foo « raw_data[0]; // initialize from array
    long double multi_array[2][3][5][7];
    foo » multi_array[0][0][0][0]; // copy data to multi-dimensional array
    foo « multi_array[0][0][0][0]; // initialize from multi-dimensional array
    return 0;
}
```

7.3.3.22 operator=() [1/2]

```
template<class T >
TensorBase< T > & TensorUtils::TensorBase< T >::operator= (
              const std::vector< T > & rhs )
```

Assigns the components in lexicographical order from a vector.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7});
    foo = std::vector<long double>(foo.size(), 1.0L); // initialize from a vector
    return 0;
}
```

7.3.3.23 operator=() [2/2]

```
template<class T >
template<class T2 >
TensorBase< T > & TensorUtils::TensorBase< T >::operator= (
             const TensorBase< T2 > & rhs )
```

Assigns this tensor with `rhs`. If the components have the same type, the default copy assignment is invoked.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    TensorUtils::tensor<float> bar;
    bar = foo;
    return 0;
}
```

7.3.3.24 operator>>()

```
template<class T >
template<class T2 >
T2 & TensorUtils::TensorBase< T >::operator>> (
             T2 & rhs )
```

Copy the components in lexicographical order to an array. No error-handling!

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7});
    foo.arange();
    long double raw_data[2*3*5*7];
    foo » raw_data[0]; // copy data to array
    foo « raw_data[0]; // initialize from array
    long double multi_array[2][3][5][7];
    foo » multi_array[0][0][0][0]; // copy data to multi-dimensional array
    foo « multi_array[0][0][0][0]; // initialize from multi-dimensional array
    return 0;
}
```

7.3.3.25 plus()

```
template<class T >
template<class T2 >
TensorBase< T > TensorUtils::TensorBase< T >::plus (
             TensorBase< T2 > & rhs,
             const std::vector< size_t > & at_lhs = {},
             const std::vector< size_t > & at_rhs = {} )
```

Return the sum of a sub-tensor of this tensor with a sub-tensor of `rhs`. Number of components must match, else ErrorHandler::ShapeMismatch is thrown.

Parameters

rhs	Second operand.
at_lhs	Indices specifying the sub-tensor of the first operand.
at_rhs	Indices specifying the sub-tensor of the second operand.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    TensorUtils::tensor<float> bar({2*3,5,7},1.0);
    TensorUtils::tensor<double> foobar;
    foobar = foo.plus(bar, {1,2}, {5}); // foobar has shape {5,7}
    return 0;
}
```

7.3.3.26 print()

```
template<class T >
void TensorUtils::TensorBase< T >::print ( )
```

Prints all sub-matrices in lexicographical order to "std::cout". Vectors are printed as row-vectors. The format is the same as for write.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<float> foo({2,3,5,7});
    foo.arange();
    foo.print();
    return 0;
}
```

7.3.3.27 product()

```
template<class T >
TensorBase< T > TensorUtils::TensorBase< T >::product (
            const T & rhs,
            const std::vector< size_t > & at_lhs = {} )
```

Return the product of a sub-tensor of this tensor with `rhs`.

Parameters

rhs	Second operand.
at_lhs	Indices specifying the sub-tensor of the first operand.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    TensorUtils::tensor<float> bar;
    bar = foo.product(5, {1,2}); // bar has shape {5,7}
    return 0;
}
```

7.3.3.28 quotient()

```
template<class T >
TensorBase< T > TensorUtils::TensorBase< T >::quotient (
             const T & rhs,
             const std::vector< size_t > & at_lhs = {} )
```

Return the quotient of a sub-tensor of this tensor with `rhs`.

Parameters

rhs	Second operand.
at_lhs	Indices specifying the sub-tensor of the first operand.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    TensorUtils::tensor<float> bar;
    bar = foo.quotient(5, {1,2}); // bar has shape {5,7}
    return 0;
}
```

7.3.3.29 read()

```
template<class T >
void TensorUtils::TensorBase< T >::read (
              std::string path )
```

Reads arbitrary tensors from text or binary files.

Parameters

path	Specifies the source path. The extension specifies the file format. For text files use any extension except the following that are used for binary files: • .f32 float • .f64 double • .f80 long double • .uc unsigned char • .sc signed char • .us unsigned short • .s short • .u unsigned • .int int • .ul unsigned long • .l long • .ull unsigned long long • .ll long long

For text files, the first line must contain the shape of the tensor. Empty lines are ignored. The header line is followed by a lexicographical list of all sub-matrices. Vectors are row-vectors. Note that print will display the same format.

Binary files are formatted as follows. The first block contains sizeof(size_t) bytes specifying shape.size(). The second block contains shape.size()∗sizeof(size_t) bytes specifying the components of shape. The third block contains sizeof(size_t) bytes specifying the container size. The fourth block contains this->size()∗sizeof(T) bytes specifying the components of the tensor, where T is the type of the components specified by the extension.

```
#include "TensorUtils.hpp"
int main()
{
    using namespace TensorUtils;
    using namespace ErrorHandler;
    tensor<double> foo;
    try
    {
        foo.read("foo.txt");
        foo.read("foo.f32");
        foo.read("foo.ull");
    }
    catch(UnableToOpenFile &ex) // unable to open file
    {
        //
    }
    catch(ShapeMismatch &ex) // Shape in header does not match given data: corrupted file?
    {
        //
    }
    catch(std::exception &ex) // catch any other exception
    {
        //
    }
    return 0;
}
```

7.3.3.30 reshape()

```
template<class T >
TensorBase< T > & TensorUtils::TensorBase< T >::reshape (
             const std::vector< size_t > & shape )
```

Assigns a new shape to this tensor and updates incr.

Parameters

shape	Specifies the new shape.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<double> foo({2,3,5,7});
    foo.arange();
    foo.reshape({2*3,5*7}); // same components with same order, but different shape
    return 0;
}
```

7.3.3.31 slice()

```
template<class T >
TensorBase< T > TensorUtils::TensorBase< T >::slice (
             const std::vector< size_t > & idx_at )
```

Slices a sub-tensor and returns by value.

Parameters

idx↩_at	Permutation of (0,1,...,N-1), where N is the rank. Indices are transposed accordingly.

Returns

Returns a the sub-tensor addressed by `idx_at` by value.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<double> foo({2,3,5,7});
    foo.arange();
    TensorUtils::tensor<double> bar;
    bar = foo.slice({1,2}); // contains the last sub-tensor with shape {5,7}
    bar.print();
    return 0;
}
```

7.3.3.32 substract()

```
template<class T >
template<class T2 >
TensorBase< T > & TensorUtils::TensorBase< T >::substract (
             TensorBase< T2 > & rhs,
             const std::vector< size_t > & at_lhs = {},
             const std::vector< size_t > & at_rhs = {} )
```

Substract a sub-tensor of `rhs` from a sub-tensor of this tensor. Number of components must match, else ErrorHandler::ShapeMismatch is thrown.

Parameters

rhs	Second operand.
at_lhs	Indices specifying the sub-tensor of the first operand.
at_rhs	Indices specifying the sub-tensor of the second operand.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    TensorUtils::tensor<float> bar({2*3,5,7},1.0);
    foo.substract(bar, {1,2}, {5});
    return 0;
}
```

7.3.3.33 transpose()

```
template<class T >
TensorBase< T > TensorUtils::TensorBase< T >::transpose (
             const std::vector< unsigned > & axes )
```

Permutes the indices of the tensor and returns by value.

Parameters

axes	Permutation of (0,1,...,N-1), where N is the rank. Indices are transposed accordingly.

Returns

Tensor with transposed indices.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<double> foo({2,3,5,7});
    foo.arange();
    foo = foo.transpose({0,2,1,3}); // New shape is: {2,5,3,7}
    return 0;
}
```

7.3.3.34 write() [1/2]

```
template<class T >
void TensorUtils::TensorBase< T >::write (
            std::string oname,
            std::string folder )
```

Write arbitrary tensors to text or binary files.

Parameters

oname	Specifies the file name. The extension specifies the file format. For text files use any extension except the following that are used for binary files: • .f32 float • .f64 double • .f80 long double • .uc unsigned char • .sc signed char • .us unsigned short • .s short • .u unsigned • .int int • .ul unsigned long • .l long • .ull unsigned long long • .ll long long
folder	Specifies the output path.

See read for details on the file format. You may add the number of significant digits when writing text files, see write(std::string,std::string,int).

```
#include "TensorUtils.hpp"
int main()
{
    using namespace TensorUtils;
    tensor<double> foo({2,3,5,7}, 1.0);
    foo.write("foo.txt", ".", 10);  // text file: writes 10 significant digits
    foo.write("foo.dat", "./"); // text file: if floating point: uses std::numeric_limits<T>::max_digits10
    foo.write("foo.f32", ".");  // binary file: float
    foo.write("foo.ull", ".");  // binary file: unsigned long long
    return 0;
}
```

7.3.3.35 write() [2/2]

```
template<class T >
void TensorUtils::TensorBase< T >::write (
            std::string oname,
            std::string folder,
            int precision )
```

For text files only. See also write(std::string,std::string) for details.

Parameters

oname	Specifies the file name.
folder	Specifies the output path.
precision	Number of significant digits when writing text files for floating point types.

```
#include "TensorUtils.hpp"
int main()
{
    using namespace TensorUtils;
    tensor<double> foo({2,3,5,7}, 1.0);
    try
    {
        foo.write("foo.txt", ".", 10); // OK!
        foo.write("foo.f32", ".", 10); // throws an error!
    }
    catch(std::runtime_error &ex) // Binary file extension but text file requested!
    {
        return 1;
    }
    return 0;
}
```

7.3.4 Friends And Related Function Documentation

7.3.4.1 operator*

```
template<class T >
TensorBase< T > operator* (
            const T & lhs,
            TensorBase< T > rhs )   [friend]
```

Scalar multiplication from the left.

```
#include "TensorUtils.hpp"
```

```
int main()
{
    TensorUtils::tensor<long double> foo({2,3,5,7},1.0);
    foo = 2*foo;
    return 0;
}
```

7.3.5 Member Data Documentation

7.3.5.1 incr

template<class T >
std::vector<size_t> TensorUtils::TensorBase< T >::incr

Internally accelerates access of components.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<double> foo({10,10,10});
    foo.arange();
    bool i_am_true = foo(2,3,5) == foo[2*foo.incr[0]+3*foo.incr[1]+5*foo.incr[2]];
    return 0;
}
```

7.3.5.2 shape

template<class T >
std::vector<size_t> TensorUtils::TensorBase< T >::shape

Storage for all components in lexicographical order. The initialized memory will be exactly the same as for multidimensional arrays. See also operator<< and operator>>.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<double> my_tensor({10,10});
    my_tensor.arange(1); // same as the following
    my_tensor[0]=1;
    my_tensor[1]=2;
    // ...
    my_tensor[99]=100;
    return 0;
}
```

Specifies the range for all indices.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<double> foo({2,3});
    bool i_am_true = foo.shape == std::vector<size_t>{2,3} ;
    for(unsigned i=0; i<foo.shape[0]; i++)
    {
        for(unsigned j=0; j<foo.shape[1]; j++)
        {
            foo(i,j);
        }
    }
    return 0;
}
```

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

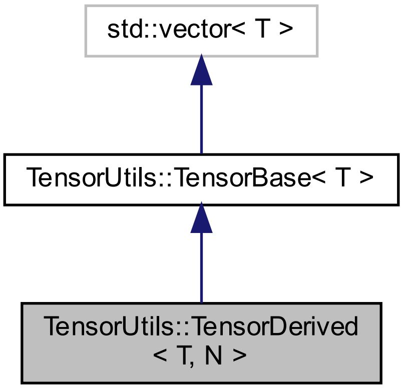
- TensorBase.hpp

7.4 TensorUtils::TensorDerived< T, N > Class Template Reference

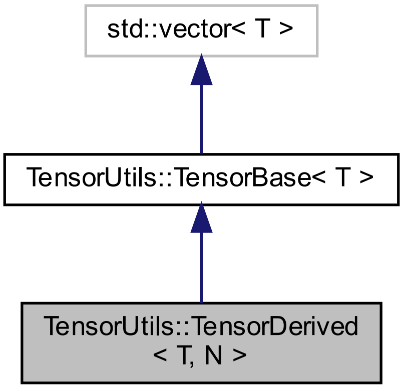
This class defines a tensor with fixed rank N=0,1,... and inherits from TensorBase. The spezialization for N=-1 defines a tensor with mutable rank.

```
#include <TensorDerived.hpp>
```

Inheritance diagram for TensorUtils::TensorDerived< T, N >:



Collaboration diagram for TensorUtils::TensorDerived< T, N >:



Public Member Functions

- **TensorDerived** ()

Constructor is inherited from TensorBase and resizes shape and incr with size N.

- **TensorDerived** (const std::vector< size_t > shape)

  Inherits from TensorBase and throws ErrorHandler::RankMismatch if shape.size()!=N.
- **TensorDerived** (const std::vector< size_t > shape, const T &val)

  Inherits from TensorBase and throws ErrorHandler::RankMismatch if shape.size()!=N.
- void **alloc** (const std::vector< size_t > shape)

  Inherits from TensorBase and throws ErrorHandler::RankMismatch if shape.size()!=N.
- void **alloc** (const std::vector< size_t > shape, const T &val)

  Inherits from TensorBase and throws ErrorHandler::RankMismatch if shape.size()!=N.
- void **clear** ()

  Inherits from TensorBase and resizes shape and incr with size N.
- template<class T2 >
  TensorDerived< T, N > & **operator=** (const TensorBase< T2 > &rhs)

  Calls TensorBase<T>::operator= and returns ∗this by reference. Throws ErrorHandler::RankMismatch if shape.↩size()!=N.
- TensorDerived< T, N > & **operator=** (const std::vector< T > &rhs)

  Calls TensorBase<T>::operator= and returns ∗this by reference. Throws ErrorHandler::RankMismatch if shape.↩size()!=N.

Additional Inherited Members

7.4.1 Detailed Description

template<class T, int N>
class TensorUtils::TensorDerived< T, N >

This class defines a tensor with fixed rank N=0,1,... and inherits from TensorBase. The spezialization for N=-1 defines a tensor with mutable rank.

This class inherits all functionality from the base class TensorBase. It allows to separate the types for tensors of different ranks, if desired. This is in particular useful to overload functions for different ranks of its arguments. If any method would change the rank for N>=0, ErrorHandler::RankMismatch is thrown.

```
#include "TensorUtils.hpp"
int main()
{
    TensorUtils::tensor<double,3> bar;
    bool i_am_true = (bar.empty() && bar.shape.size() == 3 && bar.incr.size()==3);
    TensorUtils::tensor<float,4> foo({2,3,5,7}, 1.0);
    TensorUtils::tensor<long double> foobar({2,3,5,7,11}, 2.0);
    // foo = foobar;            // throws RankMismatch!
    // foo.alloc(bar.shape);    // throws RankMismatch!
    foobar = foo;               // OK!
    return 0;
}
```

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

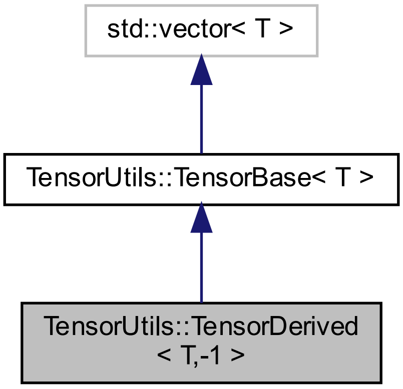
- TensorDerived.hpp

7.5 TensorUtils::TensorDerived< T,-1 > Class Template Reference

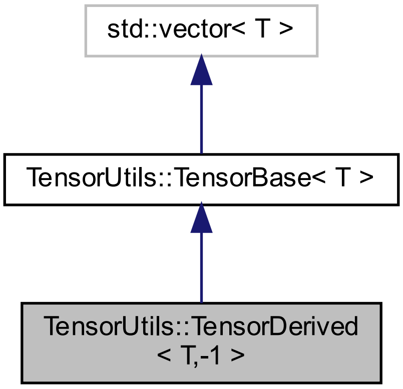
This class specialization defines a tensor with mutable rank and inherits from TensorBase.

```
#include <TensorDerived.hpp>
```

Inheritance diagram for TensorUtils::TensorDerived< T,-1 >:



Collaboration diagram for TensorUtils::TensorDerived< T,-1 >:



Public Member Functions

- **TensorDerived** ()

  Constructor is inherited from TensorBase.

- **TensorDerived** (const std::vector< size_t > shape)

  Constructor is inherited from TensorBase.
- **TensorDerived** (const std::vector< size_t > shape, const T &val)

  Constructor is inherited from TensorBase.
- template<class T2 >
  TensorDerived< T,-1 > & **operator=** (const TensorBase< T2 > &rhs)

  Calls TensorBase< T >::operator= and returns ∗this by reference.
- TensorDerived< T,-1 > & **operator=** (const std::vector< T > &rhs)

  Calls TensorBase< T >::operator= and returns ∗this by reference.

Additional Inherited Members

7.5.1 Detailed Description

template<class T>
class TensorUtils::TensorDerived< T,-1 >

This class specialization defines a tensor with mutable rank and inherits from TensorBase.

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

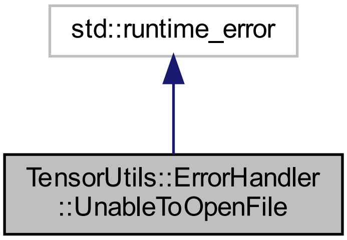
- TensorDerived.hpp

7.6 TensorUtils::ErrorHandler::UnableToOpenFile Class Reference

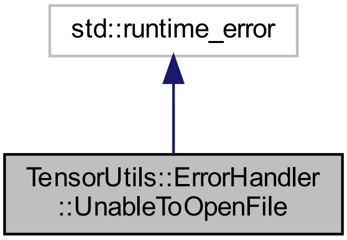
This error is thrown, if a file cannot be opened. Inherits from std::runtime_error.

```
#include <ErrorHandler.hpp>
```

Inheritance diagram for TensorUtils::ErrorHandler::UnableToOpenFile:



Collaboration diagram for TensorUtils::ErrorHandler::UnableToOpenFile:



Public Member Functions

- **UnableToOpenFile** (const std::string &what_arg)
  Constructor inherited from std::runtime_error.

7.6.1 Detailed Description

This error is thrown, if a file cannot be opened. Inherits from std::runtime_error.

See ErrorHandler for details.

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

- ErrorHandler.hpp

Chapter 8

File Documentation

8.1 ErrorHandler.hpp

```
#ifndef ERRORHANDLER_HPP
#define ERRORHANDLER_HPP

#include <stdexcept>
#include <string>

namespace TensorUtils
{
    namespace ErrorHandler
    {

        class UnableToOpenFile : public std::runtime_error
        {
            public:
            explicit UnableToOpenFile (const std::string& what_arg) : std::runtime_error(what_arg) {};
        };


        class ShapeMismatch : public std::runtime_error
        {
            public:
            explicit ShapeMismatch (const std::string& what_arg) : std::runtime_error(what_arg) {};
        };


        class RankMismatch : public std::runtime_error
        {
            public:
            explicit RankMismatch (const std::string& what_arg) : std::runtime_error(what_arg) {};
        };
    }
}

#endif // ERRORHANDLER_HPP
```

8.2 TensorBase.hpp

```
#ifndef TENSORBASE_HPP
#define TENSORBASE_HPP

#include <vector>
#include <string>

namespace TensorUtils
{
    template<class T>
    class TensorBase : public std::vector<T>
    {
        public:

            // avoid name hiding of base class constructors
            using std::vector<T>::vector;

            TensorBase();

            TensorBase(const std::vector<size_t> &shape);

            TensorBase(const std::vector<size_t> &shape, const T& val);

            virtual ~TensorBase();

             void alloc(const std::vector<size_t> &shape);

             void alloc(const std::vector<size_t> &shape, const T& val);

             void init(const T& val);

             void arange(T val=0);

             void clear();

             void print();

             void read(std::string path);

             void write(std::string oname, std::string folder);

             void write(std::string oname, std::string folder, int precision);

             TensorBase<T> transpose(const std::vector<unsigned> &axes);

             TensorBase<T> slice(const std::vector<size_t> &idx_at);

             TensorBase<T>& reshape(const std::vector<size_t> &shape);

             template<class T2>
             TensorBase<T> dot(
                 TensorBase<T2>&             rhs,
                 const std::vector<int>      &idx_lhs,
                 const std::vector<int>      &idx_rhs,
                 const std::vector<size_t>   &idx_at={});

             TensorBase<T>& operator= (const std::vector<T>& rhs);

             template<class T2> TensorBase<T>&   operator=   (const TensorBase<T2>& rhs);

             template<class T2> TensorBase<T>&   operator+=  (const TensorBase<T2>& rhs);

             template<class T2> TensorBase<T>    operator+   (const TensorBase<T2>& rhs);

             template<class T2> TensorBase<T>&   operator-=  (const TensorBase<T2>& rhs);

             template<class T2> TensorBase<T>    operator-   (const TensorBase<T2>& rhs);

             TensorBase<T>&                      operator*=  (const T& rhs);

             TensorBase<T>                       operator*   (const T& rhs);

             TensorBase<T>&                      operator/=  (const T& rhs);

             TensorBase<T>                       operator/   (const T& rhs);

             template<class T2> TensorBase<T>&   operator«  (T2& rhs);

             template<class T2>  T2&             operator»  (T2& rhs);

             friend TensorBase<T> operator* (const T& lhs, TensorBase<T> rhs) { rhs*=lhs; return rhs; };

             template<class T2>
             TensorBase<T>& assign(
                 TensorBase<T2>              &rhs,
                 const std::vector<size_t>   &at_lhs={},
                 const std::vector<size_t>   &at_rhs={});

             template<class T2>
             TensorBase<T>& add(
                 TensorBase<T2>              &rhs,
                 const std::vector<size_t>   &at_lhs={},
                 const std::vector<size_t>   &at_rhs={});

             template<class T2>
             TensorBase<T>& substract(
                 TensorBase<T2>              &rhs,
                 const std::vector<size_t>   &at_lhs={},
                 const std::vector<size_t>   &at_rhs={});

             TensorBase<T>& multiply(
                 const T                     &rhs,
                 const std::vector<size_t>   &at_lhs={});
```

```

            TensorBase<T>& divide(
                const T                     &rhs,
                const std::vector<size_t>   &at_lhs={});

            template<class T2>
            TensorBase<T> plus(
                TensorBase<T2>              &rhs,
                const std::vector<size_t>   &at_lhs={},
                const std::vector<size_t>   &at_rhs={});

            template<class T2>
            TensorBase<T> minus(
                TensorBase<T2>              &rhs,
                const std::vector<size_t>   &at_lhs={},
                const std::vector<size_t>   &at_rhs={});

            TensorBase<T> product(
                const T                     &rhs,
                const std::vector<size_t>   &at_lhs={});

            TensorBase<T> quotient(
                const T                     &rhs,
                const std::vector<size_t>   &at_lhs={});

            T& operator()(const std::vector<size_t> &indices);

            T& operator()(const std::vector<size_t*> &indices);

            T& operator()();
            T& operator()(size_t n0);
            T& operator()(size_t n0, size_t n1);
            T& operator()(size_t n0, size_t n1, size_t n2);
            T& operator()(size_t n0, size_t n1, size_t n2, size_t n3);
            T& operator()(size_t n0, size_t n1, size_t n2, size_t n3, size_t n4);
            T& operator()(size_t n0, size_t n1, size_t n2, size_t n3, size_t n4, size_t n5);
            T& operator()(size_t n0, size_t n1, size_t n2, size_t n3, size_t n4, size_t n5, size_t n6);
            T& operator()(  size_t n0, size_t n1, size_t n2, size_t n3,
                            size_t n4, size_t n5, size_t n6, size_t n7);

            std::vector<size_t> shape;

            std::vector<size_t> incr;

        protected:
            void read_txt_helper(std::string path);
            void write_txt(std::string oname, std::string folder);
            template<class BUFFER_TYPE> void print_helper();
            template<class BUFFER_TYPE> void read_txt(std::string path);
            template<class BUFFER_TYPE> void read_bin(std::string path);
            template<class BUFFER_TYPE> void write_bin(std::string basename, std::string folder);
            template<class BUFFER_TYPE> void write_txt(std::string oname, std::string folder, int precision);
    };
}


#endif // TENSORBASE_HPP
```

8.3 TensorDerived.hpp

```

#ifndef TENSORDERIVED_HPP
#define TENSORDERIVED_HPP

#include "TensorBase.hpp"

namespace TensorUtils
{

    template<class T, int N>
    class TensorDerived : public TensorBase<T>
    {
        public:
            TensorDerived();

            TensorDerived(const std::vector<size_t> shape);

            TensorDerived(const std::vector<size_t> shape, const T& val);

            void alloc(const std::vector<size_t> shape);

            void alloc(const std::vector<size_t> shape, const T &val);
            void clear();

            template<class T2> TensorDerived<T,N>& operator= (const TensorBase<T2> &rhs);

            TensorDerived<T,N>& operator= (const std::vector<T> &rhs);
    };

    template<class T>
    class TensorDerived<T,-1> : public TensorBase<T>
    {
        public:
            TensorDerived() : TensorBase<T>() {};

            TensorDerived(const std::vector<size_t> shape)  : TensorBase<T>(shape) {};

            TensorDerived(const std::vector<size_t> shape, const T& val) : TensorBase<T>(shape, val) {};

            template<class T2> TensorDerived<T,-1>& operator= (const TensorBase<T2> &rhs);

            TensorDerived<T,-1>& operator= (const std::vector<T> &rhs);
    };
}

#endif // TENSORDERIVED_HPP

```

8.4 TensorUtils.hpp

```

#ifndef TENSORUTILS_HPP
#define TENSORUTILS_HPP

#include "ErrorHandler.hpp"
#include "TensorDerived.hpp"

namespace TensorUtils
{
    template<class T, int N=-1> using tensor = TensorDerived<T,N>;
}
#endif // TENSORUTILS_HPP

```

Index

add
    TensorUtils::TensorBase< T >, 26
alloc
    TensorUtils::TensorBase< T >, 26, 27
arange
    TensorUtils::TensorBase< T >, 27
assign
    TensorUtils::TensorBase< T >, 28

clear
    TensorUtils::TensorBase< T >, 28

divide
    TensorUtils::TensorBase< T >, 28
dot
    TensorUtils::TensorBase< T >, 29

ErrorHandler, 18
ErrorHandler.hpp, 49

incr
    TensorUtils::TensorBase< T >, 42
init
    TensorUtils::TensorBase< T >, 29

minus
    TensorUtils::TensorBase< T >, 30
multiply
    TensorUtils::TensorBase< T >, 30

operator<<
    TensorUtils::TensorBase< T >, 34
operator>>
    TensorUtils::TensorBase< T >, 35
operator*
    TensorUtils::TensorBase< T >, 31, 41
operator*=
    TensorUtils::TensorBase< T >, 31
operator()
    TensorUtils::TensorBase< T >, 31
operator+
    TensorUtils::TensorBase< T >, 32
operator+=
    TensorUtils::TensorBase< T >, 32
operator-
    TensorUtils::TensorBase< T >, 32
operator-=
    TensorUtils::TensorBase< T >, 33
operator/
    TensorUtils::TensorBase< T >, 33
operator/=
    TensorUtils::TensorBase< T >, 33
operator=
    TensorUtils::TensorBase< T >, 34

plus
    TensorUtils::TensorBase< T >, 35
print
    TensorUtils::TensorBase< T >, 36
product
    TensorUtils::TensorBase< T >, 36

quotient
    TensorUtils::TensorBase< T >, 36

read
    TensorUtils::TensorBase< T >, 37
reshape
    TensorUtils::TensorBase< T >, 38

shape
    TensorUtils::TensorBase< T >, 42
slice
    TensorUtils::TensorBase< T >, 38
substract
    TensorUtils::TensorBase< T >, 39

tensor
    TensorUtils, 18
TensorBase
    TensorUtils::TensorBase< T >, 24
TensorBase.hpp, 49
TensorDerived.hpp, 51
TensorUtils, 17
    tensor, 18
TensorUtils.hpp, 52
TensorUtils::ErrorHandler::RankMismatch, 19
TensorUtils::ErrorHandler::ShapeMismatch, 20
TensorUtils::ErrorHandler::UnableToOpenFile, 46
TensorUtils::TensorBase< T >, 21
    add, 26
    alloc, 26, 27
    arange, 27
    assign, 28
    clear, 28
    divide, 28
    dot, 29
    incr, 42
    init, 29
    minus, 30
    multiply, 30
    operator<<, 34

operator>>, 35
operator*, 31, 41
operator*=, 31
operator(), 31
operator+, 32
operator+=, 32
operator-, 32
operator-=, 33
operator/, 33
operator/=, 33
operator=, 34
plus, 35
print, 36
product, 36
quotient, 36
read, 37
reshape, 38
shape, 42
slice, 38
substract, 39
TensorBase, 24
transpose, 39
write, 40, 41

TensorUtils::TensorDerived< T, N >, 43
TensorUtils::TensorDerived< T,-1 >, 45
transpose
TensorUtils::TensorBase< T >, 39

write
TensorUtils::TensorBase< T >, 40, 41