CommonLibs

1.1

Generated by Doxygen 1.9.1

Chapter 1

Home

1.1 Introduction

Library contains custom types and utulites (vector, string, etc.) created with minimal dependency from standard libraries. I implement this for educational purposes only.

1.2 Installation

1. Create Visual Studio solution or use existing one
2. Place this folder into your solution. I recommend using git submodules, because it allows to update lib easily:
   (a) `git init`, if you do not have repository yet
   (b) `git submodule add https://github.com/ptrToYuri/CommonLibs.git`
3. Right click on solution in Visual Studio -> Add -> Existing project -> find and select "CommonLibs.vcxproj" in the file explorer
4. Right click on project in which you want to use with this lib -> Add -> Reference -> Select "CommonLibs" -> Ok

1.3 Update

If you downloaded CommonLibs via git submodule, run `git submodule update --recursive --remote`. Otherwise files may be replaced manually.

1.4 Get started

Documentation is available on this website: `Namespaces`, `List of classes`, `List of header files`.
You can use the sidebar and dropdown menu for advanced navigation.

1.5 Reliability

Code is neither enough optimized nor stable. Please, keep in mind that there might be bad pracices and mistakes if you want to learn something from this code.

Chapter 2

Todo List

Class Common::TOptional< T >
: SFINAE for == operator

Class Common::TPair< T1, T2 >
: Placement new

Class Common::TVector< T >
: In case of construction errors, do not decrease capacity unless CapacityRule is set to NeverReserve. Capacity management is not consistent now, especially if move operation throws

: Implement SFINAE to support types without nonparam ctor and types without overloaded == operator

Chapter 3

Bug List

Class Common::TVector< T >
    Move may not be performed

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

Chapter 6

Class Index

6.1 Class List

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

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Chapter 8

Namespace Documentation

8.1 Common Namespace Reference

Namespaces

- Iterators

Classes

- class CException

  Basic exception class. C-style message is required.
- class COutOfRange

  Represents "Out of Range" error. Can hold message, requested and expected indices.
- class CBadAlloc

  Represents allocation failed error (usually rethrown from new)
- class CDoesNotExist

  Represents "Element does not exist" error. Stores message.
- class TOptional

  Represents object that may not exist.
- class TPair

  Container that represents a pair of objects.
- class TVector

  Container representing array that can change its size.
- struct RemoveReference
- struct RemoveReference< T & >
- struct RemoveReference< T && >

Functions

- template<typename T1 , typename T2 >
  TPair< T1, T2 > MakePair (const T1 &First, const T2 &Second) noexcept
- template<typename IteratorType >
  size_t GetIteratorDistance (IteratorType Begin, IteratorType End)

  Counts elements between two iterators. Range: [Begin: End)
- template<typename T >
  void Allocate (size_t NewSize, T ∗&OutBuffer)
- template<typename T >
  void Deallocate (T ∗&OutBuffer) noexcept
- template<typename T >
  void Construct (size_t Index, T ∗OutBuffer, const T &Value)
- template<typename T >
  void Destruct (size_t Index, T ∗OutBuffer) noexcept
- template<typename T >
  void DestructRange (size_t From, size_t To, T ∗OutBuffer) noexcept
- template<typename T >
  void DestructAll (size_t Size, T ∗OutBuffer) noexcept
- template<typename T >
  void SafeMoveBlock (size_t Size, T ∗FromBuffer, T ∗ToBuffer)
- template<typename T >
  void SafeMoveBlockReverse (size_t Size, T ∗FromBuffer, T ∗ToBuffer)
- template<typename T >
  void Reconstruct (size_t CopySize, size_t NewCapacity, T ∗&OutBuffer, size_t &OutCapacity, size_t &Out←
  Size)
- template<typename IteratorType , typename T >
  void SafeBulkConstruct (size_t StartPosition, IteratorType From, IteratorType To, T ∗OutBuffer)
- template<typename T >
  void SafeFillConstruct (size_t StartPosition, size_t EndPosition, T ∗OutBuffer, const T &Value)
- size_t GetRawStringLength (const char ∗NullTermString)

  Calculates length of the C-string.
- size_t GetRawStringLength (const char ∗NullTermString, size_t MaxLength)

  Calculates length of the C-string. Stops when null character is reached or MaxLength elements have been counted. Useful with malformatted input.
- void CopyRawString (const char ∗NullTermStringFrom, char ∗const RawStringTo)

  Does the copy of C-style string (ended with '\0').
- void CopyRawString (const char ∗NullTermStringFrom, char ∗RawStringTo, size_t MaxLength)

  Does the copy of C-style string. Stops when null character is reached or MaxLength elements have been copied. Useful when string should be cut.
- bool AreRawStringsEqual (const char ∗NullTermString1, const char ∗NullTermString2)

  Checks whether two C-style strings are equal.
- bool AreRawStringsEqual (const char ∗NullTermString1, const char ∗NullTermString2, size_t MaxCompare←
  Length)

  Checks whether two C-style strings are equal. Use this to compare N first elements, even if strings are not null-terminated.
- template<typename IteratorType , typename FunctionType >
  void BubbleSort (IteratorType Begin, IteratorType End, FunctionType Comparator)
- template<typename IteratorType , typename FunctionType >
  void SelectionSort (IteratorType Begin, IteratorType End, FunctionType Comparator)
- template<typename IteratorType , typename FunctionType >
  void QuickSort (IteratorType Begin, IteratorType End, FunctionType Comparator)
- template<typename T >
  RemoveReference< T >::Type && Move (T &&Value)
- template<typename T >
  void Swap (T &First, T &Second)

8.1.1 Function Documentation

8.1.1.1 MakePair()

```
template<typename T1 , typename T2 >
TPair<T1, T2> Common::MakePair (
            const T1 & First,
            const T2 & Second ) [inline], [noexcept]
```

8.1.1.2 GetIteratorDistance()

```
template<typename IteratorType >
size_t Common::GetIteratorDistance (
            IteratorType Begin,
            IteratorType End )
```

Counts elements between two iterators. Range: [Begin: End)

Template Parameters

IteratorType	Iterator with implemented ++, != and *

Parameters

Begin	Iterator referring to the first element
End	Iterator referring to the element after last one

Returns

Distance between iterators

Note

Begin must not be greater than end (negative results are not supported)

8.1.1.3 Allocate()

```
template<typename T >
void Common::Allocate (
            size_t NewSize,
            T *& OutBuffer ) [inline]
```

8.1.1.4 Deallocate()

```
template<typename T >
void Common::Deallocate (
            T *& OutBuffer )  [inline], [noexcept]
```

8.1.1.5 Construct()

```
template<typename T >
void Common::Construct (
            size_t Index,
            T * OutBuffer,
            const T & Value )  [inline]
```

8.1.1.6 Destruct()

```
template<typename T >
void Common::Destruct (
            size_t Index,
            T * OutBuffer )  [inline], [noexcept]
```

8.1.1.7 DestructRange()

```
template<typename T >
void Common::DestructRange (
            size_t From,
            size_t To,
            T * OutBuffer )  [inline], [noexcept]
```

8.1.1.8 DestructAll()

```
template<typename T >
void Common::DestructAll (
            size_t Size,
            T * OutBuffer )  [inline], [noexcept]
```

8.1.1.9 SafeMoveBlock()

```
template<typename T >
void Common::SafeMoveBlock (
             size_t Size,
             T * FromBuffer,
             T * ToBuffer )
```

8.1.1.10 SafeMoveBlockReverse()

```
template<typename T >
void Common::SafeMoveBlockReverse (
             size_t Size,
             T * FromBuffer,
             T * ToBuffer )
```

8.1.1.11 Reconstruct()

```
template<typename T >
void Common::Reconstruct (
             size_t CopySize,
             size_t NewCapacity,
             T *& OutBuffer,
             size_t & OutCapacity,
             size_t & OutSize )
```

8.1.1.12 SafeBulkConstruct()

```
template<typename IteratorType , typename T >
void Common::SafeBulkConstruct (
             size_t StartPosition,
             IteratorType From,
             IteratorType To,
             T * OutBuffer )
```

8.1.1.13 SafeFillConstruct()

```
template<typename T >
void Common::SafeFillConstruct (
             size_t StartPosition,
             size_t EndPosition,
             T * OutBuffer,
             const T & Value )
```

8.1.1.14 GetRawStringLength() [1/2]

```
size_t Common::GetRawStringLength (
            const char * NullTermString )
```

Calculates length of the C-string.

Parameters

NullTermString	Char array that ends with '\0'

Returns

Number of actual letters in string ('\0' is not counted)

8.1.1.15 GetRawStringLength() [2/2]

```
size_t Common::GetRawStringLength (
             const char * NullTermString,
             size_t MaxLength )
```

Calculates length of the C-string. Stops when null character is reached or MaxLength elements have been counted. Useful with malformatted input.

Parameters

NullTermString	Char array that ends with '\0' (or not, if you rely on MaxLength and buffer size)
MaxLength	Max amount of characters to count; does not include the trailing '\0'

Returns

Number of actual letters in string ('\0' is not counted)

8.1.1.16 CopyRawString() [1/2]

```
void Common::CopyRawString (
             const char * NullTermStringFrom,
             char *const RawStringTo )
```

Does the copy of C-style string (ended with '\0').

Parameters

NullTermStringFrom	Source: char array that ends with '\0'
NullTermStringTo	Destination: Char array that is large enough to receive copied elements. May not end with '\0'

8.1.1.17 CopyRawString() [2/2]

```
void Common::CopyRawString (
            const char * NullTermStringFrom,
            char * RawStringTo,
            size_t MaxLength )
```

Does the copy of C-style string. Stops when null character is reached or MaxLength elements have been copied. Useful when string should be cut.

Parameters

NullTermStringFrom	Source: char array that ends with '\0' (or not, if you rely on MaxLength and buffer size)
NullTermStringTo	Destination: Char array that is large enough to receive copied elements. May not end with '\0'. After copying it gets '\0' anyway
MaxLength	Max amount of characters to copy; does not include trailing '\0'

8.1.1.18 AreRawStringsEqual() [1/2]

```
bool Common::AreRawStringsEqual (
            const char * NullTermString1,
            const char * NullTermString2 )
```

Checks whether two C-style strings are equal.

Parameters

NullTermString1	First null-terminated string
NullTermString2	Second null-terminated string

Returns

true if characters before '\0' are the same false otherwise

8.1.1.19 AreRawStringsEqual() [2/2]

```
bool Common::AreRawStringsEqual (
            const char * NullTermString1,
            const char * NullTermString2,
            size_t MaxCompareLength )
```

Checks whether two C-style strings are equal. Use this to compare N first elements, even if strings are not null-terminated.

Parameters

NullTermString1	First string (whether null- terminated or limited with MaxCompareLength)
NullTermString2	Second string (whether null- terminated or limited with MaxCompareLength)
MaxComparedLength	Max amount of characters to compare; does not include trailing '\0'

Returns

true if characters before '\0' are the same false otherwise

8.1.1.20 BubbleSort()

```
template<typename IteratorType , typename FunctionType >
void Common::BubbleSort (
             IteratorType Begin,
             IteratorType End,
             FunctionType Comparator )
```

8.1.1.21 SelectionSort()

```
template<typename IteratorType , typename FunctionType >
void Common::SelectionSort (
             IteratorType Begin,
             IteratorType End,
             FunctionType Comparator )
```

8.1.1.22 QuickSort()

```
template<typename IteratorType , typename FunctionType >
void Common::QuickSort (
             IteratorType Begin,
             IteratorType End,
             FunctionType Comparator )
```

8.1.1.23 Move()

```
template<typename T >
RemoveReference<T>::Type&& Common::Move (
             T && Value )
```

8.1.1.24 Swap()

```
template<typename T >
void Common::Swap (
            T & First,
            T & Second )
```

8.2 Common::Iterators Namespace Reference

Classes

- class TBlockIterator
- class TReverseBlockIterator
- class TSafeBlockIterator
- class TSafeReverseBlockIterator

Chapter 9

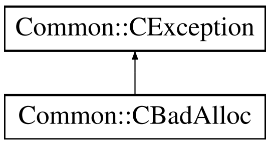
Class Documentation

9.1 Common::CBadAlloc Class Reference

Represents allocation failed error (usually rethrown from new)

`#include "CommonTypes/Exception.h"`

Inheritance diagram for Common::CBadAlloc:



Public Member Functions

- CBadAlloc (const char ∗Message) noexcept

  Pass only message, if other properties cannot be specified.
- CBadAlloc (const char ∗Message, size_t RequestedAllocSize) noexcept

  Also specify requested alloc size.
- size_t GetRequestedAllocSize ()

  Size in bytes that was intended to be allocated.

Protected Attributes

- const size_t RequestedAllocSize = 0

9.1.1 Detailed Description

Represents allocation failed error (usually rethrown from new)

9.1.2 Constructor & Destructor Documentation

9.1.2.1 CBadAlloc() [1/2]

```
Common::CBadAlloc::CBadAlloc (
            const char * Message )  [inline], [noexcept]
```

Pass only message, if other properties cannot be specified.

Parameters

Message	Description. Will be copied to the inner buffer

Note

If length of message > 47, first 47 symbols will be saved.

9.1.2.2 CBadAlloc() [2/2]

```
Common::CBadAlloc::CBadAlloc (
            const char * Message,
            size_t RequestedAllocSize )  [inline], [noexcept]
```

Also specify requested alloc size.

Parameters

Message	Description. Will be copied to the inner buffer
RequestedAllocSize	Requested allocation size

Returns

9.1.3 Member Function Documentation

9.1.3.1 GetRequestedAllocSize()

```
size_t Common::CBadAlloc::GetRequestedAllocSize ( )  [inline]
```

Size in bytes that was intended to be allocated.

Returns

Requested allocation size, that failed

Note

Returns 0 if constructed only with message.

9.1.4 Member Data Documentation

9.1.4.1 RequestedAllocSize

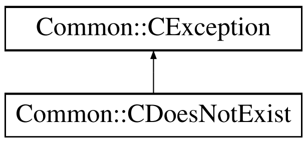
```
const size_t Common::CBadAlloc::RequestedAllocSize = 0  [protected]
```

9.2 Common::CDoesNotExist Class Reference

Represents "Element does not exist" error. Stores message.

`#include "CommonTypes/Exception.h"`

Inheritance diagram for Common::CDoesNotExist:



Public Member Functions

- CDoesNotExist (const char ∗Message) noexcept
  Pass only message, if other properties cannot be specified.

Additional Inherited Members

9.2.1 Detailed Description

Represents "Element does not exist" error. Stores message.

9.2.2 Constructor & Destructor Documentation

9.2.2.1 CDoesNotExist()

```
Common::CDoesNotExist::CDoesNotExist (
            const char * Message )  [inline], [noexcept]
```

Pass only message, if other properties cannot be specified.

Parameters

Message	Description. Will be copied to the inner buffer

Note

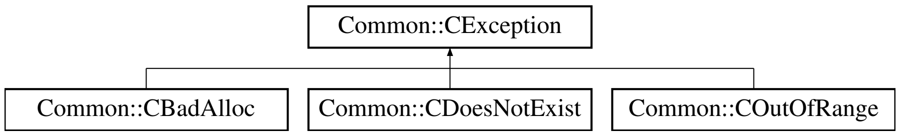
If length of message $>$ 47, first 47 symbols will be saved.

9.3 Common::CException Class Reference

Basic exception class. C-style message is required.

```
#include "CommonTypes/Exception.h"
```

Inheritance diagram for Common::CException:



Public Member Functions

- CException (const char ∗Message) noexcept

  All exceptions must provide the message.
- virtual ∼CException ()
- virtual const char ∗ GetMessage () const noexcept

  Error message.

Protected Attributes

- char Message [48]

9.3.1 Detailed Description

Basic exception class. C-style message is required.

9.3.2 Constructor & Destructor Documentation

9.3.2.1 CException()

```
Common::CException::CException (
            const char ∗ Message )  [inline], [noexcept]
```

All exceptions must provide the message.

Parameters

Message	Error description. Will be copied to an inner buffer

Note

If length of message $>$ 47, first 47 symbols will be saved.

9.3.2.2 ∼CException()

```
virtual Common::CException::∼CException ( )  [inline], [virtual]
```

9.3.3 Member Function Documentation

9.3.3.1 GetMessage()

```
virtual const char* Common::CException::GetMessage ( ) const  [inline], [virtual], [noexcept]
```

Error message.

Returns

C-style string with error description

9.3.4 Member Data Documentation

9.3.4.1 Message

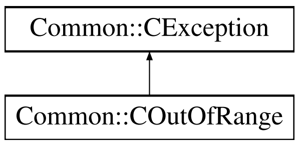
```
char Common::CException::Message[48]  [protected]
```

9.4 Common::COutOfRange Class Reference

Represents "Out of Range" error. Can hold message, requested and expected indices.

```
#include "CommonTypes/Exception.h"
```

Inheritance diagram for Common::COutOfRange:



Public Member Functions

- COutOfRange (const char ∗Message) noexcept

    Pass only message, if other properties cannot be specified.
- COutOfRange (const char ∗Message, int RequestedIndex, const TPair< size_t, size_t > &ExpectedIndex) noexcept

    Describes valid range and errored value. Contains message.
- const TPair< size_t, size_t > & GetExpectedRange () const noexcept

    Specifies valid range.
- int GetRequestedIndex () const noexcept

    Index that caused this exception.

Protected Attributes

- const int RequestedIndex = 0
- const TPair< size_t, size_t > ExpectedIndex = { 0,0 }

9.4.1 Detailed Description

Represents "Out of Range" error. Can hold message, requested and expected indices.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 COutOfRange() [1/2]

```
Common::COutOfRange::COutOfRange (
            const char * Message )  [inline], [noexcept]
```

Pass only message, if other properties cannot be specified.

Parameters

Message	Description. Will be copied to an inner buffer

Note

Range will be set to empty [0: 0), requested index to 0.

If length of message > 47, first 47 symbols will be saved.

9.4.2.2 COutOfRange() [2/2]

```
Common::COutOfRange::COutOfRange (
            const char * Message,
            int RequestedIndex,
            const TPair< size_t, size_t > & ExpectedIndex )  [inline], [noexcept]
```

Describes valid range and errored value. Contains message.

Parameters

Message	Description. Will be copied to an inner buffer
RequestedIndex	Errored index (out of range).
ExpectedIndex	Pair of Min and Max+1 indexes, that were available. Range: [First: Second)

Note

If length of message $>$ 47, first 47 symbols will be saved.

9.4.3 Member Function Documentation

9.4.3.1 GetExpectedRange()

```
const TPair<size_t, size_t>& Common::COutOfRange::GetExpectedRange ( ) const  [inline], [noexcept]
```

Specifies valid range.

Returns

Pair of Min and Max+1 indexes, that were available. Range: [First: Second)

Note

Returns 0 Index with [0: 0) range if constructed only with message.

9.4.3.2 GetRequestedIndex()

```
int Common::COutOfRange::GetRequestedIndex ( ) const  [inline], [noexcept]
```

Index that caused this exception.

Returns

Value of index, that is not in expected range

Note

Returns 0 Index with [0: 0) range if constructed only with message.

9.4.4 Member Data Documentation

9.4.4.1 RequestedIndex

```
const int Common::COutOfRange::RequestedIndex = 0  [protected]
```

9.4.4.2 ExpectedIndex

```
const TPair<size_t, size_t> Common::COutOfRange::ExpectedIndex = { 0,0 }  [protected]
```

9.5 Common::RemoveReference< T > Struct Template Reference

```
#include "CommonUtils/TypeOperations.h"
```

Public Types

- typedef T Type

9.5.1 Member Typedef Documentation

9.5.1.1 Type

```
template<typename T >
typedef T Common::RemoveReference< T >::Type
```

9.6 Common::RemoveReference< T & > Struct Template Reference

```
#include "CommonUtils/TypeOperations.h"
```

Public Types

- typedef T Type

9.6.1 Member Typedef Documentation

9.6.1.1 Type

```
template<typename T >
typedef T Common::RemoveReference< T & >::Type
```

9.7 Common::RemoveReference< T && > Struct Template Reference

```
#include "CommonUtils/TypeOperations.h"
```

Public Types

- typedef T Type

9.7.1 Member Typedef Documentation

9.7.1.1 Type

```
template<typename T >
typedef T Common::RemoveReference< T && >::Type
```

9.8 Common::Iterators::TBlockIterator< PtrType, RefType > Class Template Reference

```
#include "CommonTypes/Iterators/Block.h"
```

Public Member Functions

- TBlockIterator ()
- TBlockIterator (PtrType InitialPosition)
- const TBlockIterator & operator++ ()
- TBlockIterator operator+ (size_t Offset)
- const TBlockIterator & operator+= (size_t Offset)
- const TBlockIterator & operator-- ()
- TBlockIterator operator- (size_t Offset)
- const TBlockIterator & operator-= (size_t Offset)
- bool operator== (const TBlockIterator &Other)
- bool operator!= (const TBlockIterator &Other)
- RefType operator* ()

9.8.1 Constructor & Destructor Documentation

9.8.1.1 TBlockIterator() [1/2]

```
template<typename PtrType , typename RefType >
Common::Iterators::TBlockIterator< PtrType, RefType >::TBlockIterator ( )
```

9.8.1.2 TBlockIterator() [2/2]

```
template<typename PtrType , typename RefType >
Common::Iterators::TBlockIterator< PtrType, RefType >::TBlockIterator (
            PtrType InitialPosition )
```

9.8.2 Member Function Documentation

9.8.2.1 operator++()

```
template<typename PtrType , typename RefType >
const TBlockIterator& Common::Iterators::TBlockIterator< PtrType, RefType >::operator++ ( )
```

9.8.2.2 operator+()

```
template<typename PtrType , typename RefType >
TBlockIterator Common::Iterators::TBlockIterator< PtrType, RefType >::operator+ (
            size_t Offset )
```

9.8.2.3 operator+=()

```
template<typename PtrType , typename RefType >
const TBlockIterator& Common::Iterators::TBlockIterator< PtrType, RefType >::operator+= (
            size_t Offset )
```

9.8.2.4 operator--()

```
template<typename PtrType , typename RefType >
const TBlockIterator& Common::Iterators::TBlockIterator< PtrType, RefType >::operator-- ( )
```

9.8.2.5 operator-()

```
template<typename PtrType , typename RefType >
TBlockIterator Common::Iterators::TBlockIterator< PtrType, RefType >::operator- (
            size_t Offset )
```

9.8.2.6 operator-=()

```
template<typename PtrType , typename RefType >
const TBlockIterator& Common::Iterators::TBlockIterator< PtrType, RefType >::operator-= (
            size_t Offset )
```

9.8.2.7 operator==()

```
template<typename PtrType , typename RefType >
bool Common::Iterators::TBlockIterator< PtrType, RefType >::operator== (
            const TBlockIterator< PtrType, RefType > & Other )
```

9.8.2.8 operator"!=()

```
template<typename PtrType , typename RefType >
bool Common::Iterators::TBlockIterator< PtrType, RefType >::operator!= (
            const TBlockIterator< PtrType, RefType > & Other )
```

9.8.2.9 operator*()

```
template<typename PtrType , typename RefType >
RefType Common::Iterators::TBlockIterator< PtrType, RefType >::operator* ( )
```

9.9 Common::TOptional< T > Class Template Reference

Represents object that may not exist.

```
#include "CommonTypes/Optional.h"
```

Public Member Functions

- TOptional ()=default

  No object by default.
- TOptional (const T &Value)

  Initialize optional with existing value (copy).
- TOptional (const TOptional< T > &Other)

  Initialize by copying another TOptional.
- TOptional (TOptional< T > &&Other) noexcept

  Move constructor.
- ∼TOptional ()
- void SetValue (const T &Value)

  Set value to optional (copy).
- void Clear () noexcept

  Remove value from optional.
- TOptional< T > & operator= (const T &Value)

  Assign value to the optional (copy).
- TOptional< T > & operator= (const TOptional< T > &Other)

  Assign from another optional (copy).
- TOptional< T > & operator= (TOptional< T > &&Other) noexcept

  Move assignment.
- void Swap (TOptional< T > &Other)

  Swaps two optionals without reconstructing values.
- bool operator== (const TOptional< T > &Other)

  Checks if two optionals contain the same values.
- bool DoesValueExist () const noexcept

  Check if optional contains value.
- const T & GetValue () const

  Gets value if it exists or throws an exception.
- const T & GetValueOr (const T &OtherVariant) const noexcept

  Get value or passed value (if not possible).

9.9.1 Detailed Description

template< typename T >
class Common::TOptional< T >

Represents object that may not exist.

Todo SFINAE for == operator

9.9.2 Constructor & Destructor Documentation

9.9.2.1 TOptional() [1/4]

```
template<typename T >
Common::TOptional< T >::TOptional ( )  [default]
```

No object by default.

9.9.2.2 TOptional() [2/4]

```
template<typename T >
Common::TOptional< T >::TOptional (
            const T & Value )
```

Initialize optional with existing value (copy).

Parameters

Value	Object to create copy from

9.9.2.3 TOptional() [3/4]

```
template<typename T >
Common::TOptional< T >::TOptional (
            const TOptional< T > & Other )
```

Initialize by copying another TOptional.

Parameters

Other	Optional to create copy from

9.9.2.4 TOptional() [4/4]

```
template<typename T >
Common::TOptional< T >::TOptional (
            TOptional< T > && Other )  [noexcept]
```

Move constructor.

Parameters

Other	Temporary object to get data from

9.9.2.5 ∼TOptional()

```
template<typename T >
Common::TOptional< T >::∼TOptional ( )
```

9.9.3 Member Function Documentation

9.9.3.1 SetValue()

```
template<typename T >
void Common::TOptional< T >::SetValue (
             const T & Value )
```

Set value to optional (copy).

Parameters

Value	Object to create copy from

9.9.3.2 Clear()

```
template<typename T >
void Common::TOptional< T >::Clear ( )  [noexcept]
```

Remove value from optional.

Note

This will call destructor on internal object.

9.9.3.3 operator=() [1/3]

```
template<typename T >
TOptional<T>& Common::TOptional< T >::operator= (
             const T & Value )
```

Assign value to the optional (copy).

Parameters

Value	Object to create copy from

Returns

Reference to this optional

9.9.3.4 operator=() [2/3]

```
template<typename T >
TOptional<T>& Common::TOptional< T >::operator= (
            const TOptional< T > & Other )
```

Assign from another optional (copy).

Parameters

Other	Optional to create copy from

Returns

Reference to this optional

9.9.3.5 operator=() [3/3]

```
template<typename T >
TOptional<T>& Common::TOptional< T >::operator= (
            TOptional< T > && Other )  [noexcept]
```

Move assignment.

Parameters

Other	Temporary object to get data from

Returns

Reference to this optional

9.9.3.6 Swap()

```
template<typename T >
void Common::TOptional< T >::Swap (
             TOptional< T > & Other )
```

Swaps two optionals without reconstructing values.

Parameters

Other	Object to swap resources with

9.9.3.7 operator==()

```
template<typename T >
bool Common::TOptional< T >::operator== (
             const TOptional< T > & Other )
```

Checks if two optionals contain the same values.

Parameters

Other	Other optional to compare

Returns

True if sizes and values are equal, false otherwise

Note

Containing element must implement == operator.

9.9.3.8 DoesValueExist()

```
template<typename T >
bool Common::TOptional< T >::DoesValueExist ( ) const  [noexcept]
```

Check if optional contains value.

Returns

True if value exists, false otherwise

9.9.3.9 GetValue()

```
template<typename T >
const T& Common::TOptional< T >::GetValue ( ) const
```

Gets value if it exists or throws an exception.

Returns

Optional's value

9.9.3.10 GetValueOr()

```
template<typename T >
const T& Common::TOptional< T >::GetValueOr (
             const T & OtherVariant ) const  [noexcept]
```

Get value or passed value (if not possible).

Parameters

OtherVariant	Returned if optional is empty

Returns

Optional internal value or provided value

9.10 Common::TPair< T1, T2 > Class Template Reference

Container that represents a pair of objects.

`#include "CommonTypes/Pair.h"`

Public Member Functions

- TPair ()=default

  Initialize pair with type default values.
- TPair (const T1 &First, const T2 &Second) noexcept

  Creates a pair copying passed values.
- void MakePair (T1 First, T2 Second) noexcept

  Assigns two values to pair at once (copy).

Public Attributes

- T1 First = T1{}

  First value in pair.
- T2 Second = T2{}

  Second value in pair.

9.10.1 Detailed Description

template<typename T1, typename T2>
class Common::TPair< T1, T2 >

Container that represents a pair of objects.

Todo Placement new

9.10.2 Constructor & Destructor Documentation

9.10.2.1 TPair() [1/2]

```
template<typename T1 , typename T2 >
Common::TPair< T1, T2 >::TPair ( )  [default]
```

Initialize pair with type default values.

9.10.2.2 TPair() [2/2]

```
template<typename T1 , typename T2 >
Common::TPair< T1, T2 >::TPair (
             const T1 & First,
             const T2 & Second )  [inline], [noexcept]
```

Creates a pair copying passed values.

Parameters

First	First value in pair
Second	Second value in pair

9.10.3 Member Function Documentation

9.10.3.1 MakePair()

```
template<typename T1 , typename T2 >
void Common::TPair< T1, T2 >::MakePair (
             T1 First,
             T2 Second )  [inline], [noexcept]
```

Assigns two values to pair at once (copy).

Parameters

First	First value in pair
Second	Second value in pair

9.10.4 Member Data Documentation

9.10.4.1 First

```
template<typename T1 , typename T2 >
T1 Common::TPair< T1, T2 >::First = T1{}
```

First value in pair.

9.10.4.2 Second

```
template<typename T1 , typename T2 >
T2 Common::TPair< T1, T2 >::Second = T2{}
```

Second value in pair.

9.11 Common::Iterators::TReverseBlockIterator< PtrType, RefType > Class Template Reference

```
#include "CommonTypes/Iterators/Block.h"
```

Public Member Functions

- TReverseBlockIterator ()
- TReverseBlockIterator (PtrType InitialPosition)
- const TReverseBlockIterator & operator++ ()
- TReverseBlockIterator operator+ (size_t Offset)
- const TReverseBlockIterator & operator+= (size_t Offset)
- const TReverseBlockIterator & operator-- ()
- TReverseBlockIterator operator- (size_t Offset)
- const TReverseBlockIterator & operator-= (size_t Offset)
- bool operator== (const TReverseBlockIterator &Other)
- bool operator!= (const TReverseBlockIterator &Other)
- RefType operator∗ ()

9.11.1 Constructor & Destructor Documentation

9.11.1.1 TReverseBlockIterator() [1/2]

```
template<typename PtrType , typename RefType >
Common::Iterators::TReverseBlockIterator< PtrType, RefType >::TReverseBlockIterator ( )
```

9.11.1.2 TReverseBlockIterator() [2/2]

```
template<typename PtrType , typename RefType >
Common::Iterators::TReverseBlockIterator< PtrType, RefType >::TReverseBlockIterator (
             PtrType InitialPosition )
```

9.11.2 Member Function Documentation

9.11.2.1 operator++()

```
template<typename PtrType , typename RefType >
const TReverseBlockIterator& Common::Iterators::TReverseBlockIterator< PtrType, RefType >↩
::operator++ ( )
```

9.11.2.2 operator+()

```
template<typename PtrType , typename RefType >
TReverseBlockIterator Common::Iterators::TReverseBlockIterator< PtrType, RefType >::operator+
(
             size_t Offset )
```

9.11.2.3 operator+=()

```
template<typename PtrType , typename RefType >
const TReverseBlockIterator& Common::Iterators::TReverseBlockIterator< PtrType, RefType >↩
::operator+= (
             size_t Offset )
```

9.11.2.4 operator--()

```
template<typename PtrType , typename RefType >
const TReverseBlockIterator& Common::Iterators::TReverseBlockIterator< PtrType, RefType >↩
::operator-- ( )
```

9.11.2.5 operator-()

```
template<typename PtrType , typename RefType >
TReverseBlockIterator Common::Iterators::TReverseBlockIterator< PtrType, RefType >::operator-
(
             size_t Offset )
```

9.11.2.6 operator-=()

```
template<typename PtrType , typename RefType >
const TReverseBlockIterator& Common::Iterators::TReverseBlockIterator< PtrType, RefType >↩
::operator-= (
             size_t Offset )
```

9.11.2.7 operator==()

```
template<typename PtrType , typename RefType >
bool Common::Iterators::TReverseBlockIterator< PtrType, RefType >::operator== (
             const TReverseBlockIterator< PtrType, RefType > & Other )
```

9.11.2.8 operator"!=()

```
template<typename PtrType , typename RefType >
bool Common::Iterators::TReverseBlockIterator< PtrType, RefType >::operator!= (
             const TReverseBlockIterator< PtrType, RefType > & Other )
```

9.11.2.9 operator∗()

```
template<typename PtrType , typename RefType >
RefType Common::Iterators::TReverseBlockIterator< PtrType, RefType >::operator* ( )
```

9.12 Common::Iterators::TSafeBlockIterator< PtrType, RefType, ContType > Class Template Reference

```
#include "CommonTypes/Iterators/Block.h"
```

Public Member Functions

- TSafeBlockIterator (PtrType InitialPosition, const ContType Owner)
- const TSafeBlockIterator & operator++ ()
- TSafeBlockIterator operator+ (size_t Offset)
- const TSafeBlockIterator & operator+= (size_t Offset)
- const TSafeBlockIterator & operator-- ()
- TSafeBlockIterator operator- (size_t Offset)
- const TSafeBlockIterator & operator-= (size_t Offset)
- bool operator== (const TSafeBlockIterator &Other)
- bool operator!= (const TSafeBlockIterator &Other)
- RefType operator* ()

9.12.1 Constructor & Destructor Documentation

9.12.1.1 TSafeBlockIterator()

```
template<typename PtrType , typename RefType , typename ContType >
Common::Iterators::TSafeBlockIterator< PtrType, RefType, ContType >::TSafeBlockIterator (
             PtrType InitialPosition,
             const ContType Owner )  [inline]
```

9.12.2 Member Function Documentation

9.12.2.1 operator++()

```
template<typename PtrType , typename RefType , typename ContType >
const TSafeBlockIterator& Common::Iterators::TSafeBlockIterator< PtrType, RefType, ContType
>::operator++ ( )
```

9.12.2.2 operator+()

```
template<typename PtrType , typename RefType , typename ContType >
TSafeBlockIterator Common::Iterators::TSafeBlockIterator< PtrType, RefType, ContType >::operator+
(
             size_t Offset )
```

9.12.2.3 operator+=()

```
template<typename PtrType , typename RefType , typename ContType >
const TSafeBlockIterator& Common::Iterators::TSafeBlockIterator< PtrType, RefType, ContType
>::operator+= (
             size_t Offset )
```

9.12.2.4 operator--()

```
template<typename PtrType , typename RefType , typename ContType >
const TSafeBlockIterator& Common::Iterators::TSafeBlockIterator< PtrType, RefType, ContType
>::operator-- ( )
```

9.12.2.5 operator-()

```
template<typename PtrType , typename RefType , typename ContType >
TSafeBlockIterator Common::Iterators::TSafeBlockIterator< PtrType, RefType, ContType >::operator-
(
             size_t Offset )
```

9.12.2.6 operator-=()

```
template<typename PtrType , typename RefType , typename ContType >
const TSafeBlockIterator& Common::Iterators::TSafeBlockIterator< PtrType, RefType, ContType
>::operator-= (
             size_t Offset )
```

9.12.2.7 operator==()

```
template<typename PtrType , typename RefType , typename ContType >
bool Common::Iterators::TSafeBlockIterator< PtrType, RefType, ContType >::operator== (
             const TSafeBlockIterator< PtrType, RefType, ContType > & Other )
```

9.12.2.8 operator"!=()

```
template<typename PtrType , typename RefType , typename ContType >
bool Common::Iterators::TSafeBlockIterator< PtrType, RefType, ContType >::operator!= (
             const TSafeBlockIterator< PtrType, RefType, ContType > & Other )
```

9.12.2.9 operator∗()

```
template<typename PtrType , typename RefType , typename ContType >
RefType Common::Iterators::TSafeBlockIterator< PtrType, RefType, ContType >::operator* ( )
```

9.13 Common::Iterators::TSafeReverseBlockIterator< PtrType, RefType, ContType > Class Template Reference

```
#include "CommonTypes/Iterators/Block.h"
```

Public Member Functions

- TSafeReverseBlockIterator (PtrType InitialPosition, const ContType Owner)
- const TSafeReverseBlockIterator & operator++ ()
- TSafeReverseBlockIterator operator+ (size_t Offset)
- const TSafeReverseBlockIterator & operator+= (size_t Offset)
- const TSafeReverseBlockIterator & operator-- ()
- TSafeReverseBlockIterator operator- (size_t Offset)
- const TSafeReverseBlockIterator & operator-= (size_t Offset)
- bool operator== (const TSafeReverseBlockIterator &Other)
- bool operator!= (const TSafeReverseBlockIterator &Other)
- RefType operator∗ ()

9.13.1 Constructor & Destructor Documentation

9.13.1.1 TSafeReverseBlockIterator()

```
template<typename PtrType , typename RefType , class ContType >
Common::Iterators::TSafeReverseBlockIterator< PtrType, RefType, ContType >::TSafeReverseBlockIterator
(
            PtrType InitialPosition,
            const ContType Owner )
```

9.13.2 Member Function Documentation

9.13.2.1 operator++()

```
template<typename PtrType , typename RefType , class ContType >
const TSafeReverseBlockIterator& Common::Iterators::TSafeReverseBlockIterator< PtrType, Ref←
Type, ContType >::operator++ ( )
```

9.13.2.2 operator+()

```
template<typename PtrType , typename RefType , class ContType >
TSafeReverseBlockIterator Common::Iterators::TSafeReverseBlockIterator< PtrType, RefType,
ContType >::operator+ (
            size_t Offset )
```

9.13.2.3 operator+=()

```
template<typename PtrType , typename RefType , class ContType >
const TSafeReverseBlockIterator& Common::Iterators::TSafeReverseBlockIterator< PtrType, Ref↩
Type, ContType >::operator+= (
            size_t Offset )
```

9.13.2.4 operator--()

```
template<typename PtrType , typename RefType , class ContType >
const TSafeReverseBlockIterator& Common::Iterators::TSafeReverseBlockIterator< PtrType, Ref↩
Type, ContType >::operator-- ( )
```

9.13.2.5 operator-()

```
template<typename PtrType , typename RefType , class ContType >
TSafeReverseBlockIterator Common::Iterators::TSafeReverseBlockIterator< PtrType, RefType,
ContType >::operator- (
            size_t Offset )
```

9.13.2.6 operator-=()

```
template<typename PtrType , typename RefType , class ContType >
const TSafeReverseBlockIterator& Common::Iterators::TSafeReverseBlockIterator< PtrType, Ref↩
Type, ContType >::operator-= (
            size_t Offset )
```

9.13.2.7 operator==()

```
template<typename PtrType , typename RefType , class ContType >
bool Common::Iterators::TSafeReverseBlockIterator< PtrType, RefType, ContType >::operator== (
            const TSafeReverseBlockIterator< PtrType, RefType, ContType > & Other )
```

9.13.2.8 operator"!=()

```
template<typename PtrType , typename RefType , class ContType >
bool Common::Iterators::TSafeReverseBlockIterator< PtrType, RefType, ContType >::operator!= (
             const TSafeReverseBlockIterator< PtrType, RefType, ContType > & Other )
```

9.13.2.9 operator*()

```
template<typename PtrType , typename RefType , class ContType >
RefType Common::Iterators::TSafeReverseBlockIterator< PtrType, RefType, ContType >::operator*
( )
```

9.14 Common::TVector< T > Class Template Reference

Container representing array that can change its size.

```
#include "CommonTypes/Vector.h"
```

Public Types

- enum class EReservedCapacityRule : uint8_t { Exponential , Linear , NeverReserve }

  Defines how extra Capacity is reserved.
- enum class EShrinkBehavior { Require , Allow , Deny }

  Overrides CapacityRule for specific element removal case.
- typedef T value_type
- typedef Iterators::TBlockIterator< T *, T & > CIterator

  Iterator. Implemented op-s: ++, +=, +, --, -=, -, ==, !=, =.
- typedef Iterators::TBlockIterator< const T *, const T & > CConstIterator

  Version of CIterator for const values.
- typedef Iterators::TReverseBlockIterator< T *, T & > CReverseIterator

  Reverse iterator. Increment is actually decrement, etc.
- typedef Iterators::TReverseBlockIterator< const T *, const T & > CConstReverseIterator

  Version of TReverseIterator for const values.
- typedef Iterators::TSafeBlockIterator< T *, T &, TVector< T > * > CSafeIterator

  Iterator that does bounds checking and throws OutOfRange().
- typedef Iterators::TSafeBlockIterator< const T *, const T &, const TVector< T > * > CSafeConstIterator

  Version of TSafeIterator for const values.
- typedef Iterators::TSafeReverseBlockIterator< T *, T &, TVector< T > * > CSafeReverseIterator

  Reverse iterator that can throw OutOfRange().
- typedef Iterators::TSafeReverseBlockIterator< const T *, const T &, const TVector< T > * > CSafeConstReverseIterator

  Version of TSafeReverseIterator for const values.

Public Member Functions

- TVector (EReservedCapacityRule CapacityRule=EReservedCapacityRule::Exponential) noexcept

  Creates empty vector with Capacity preset predefined.
- TVector (size_t Size, const T &DefaultValue={}, EReservedCapacityRule CapacityRule=EReservedCapacityRule::Exponential)

  Vector with pre-created elements.
- TVector (size_t Size, const T ∗const Array, EReservedCapacityRule CapacityRule=EReservedCapacityRule::Exponential)

  Vector constructed from raw dynamic array (copy).
- TVector (const std::initializer_list< T > &ValuesList)

  Modern C++ initialization syntax: name = {...}.
- template<typename IteratorType >
  TVector (IteratorType Begin, IteratorType End, EReservedCapacityRule CapacityRule=EReservedCapacityRule::Exponential, typename std::enable_if<!std::is_integral< IteratorType >::value >::type ∗=0)

  Constructor to get values from another container.
- TVector (const TVector< T > &Other)

  Initialize by copying another TVector.
- TVector (TVector< T > &&Other) noexcept

  Move constructor.
- ∼TVector ()
- template<typename IteratorType >
  void Assign (IteratorType Begin, IteratorType End, EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  Allows to copy values from another container.
- TVector< T > & operator= (const std::initializer_list< T > &ValuesList)

  Allows assignment with = {...} style.
- TVector< T > & operator= (const TVector< T > &Other)

  Makes a copy of another vector.
- TVector< T > & operator= (TVector< T > &&Other) noexcept

  Move assignment.
- T & operator[ ] (size_t Index)

  Index operator that prodives access to the element.
- const T & operator[ ] (size_t Index) const

  Index operator for const vectors.
- T & SafeAt (size_t Index)

  [] with range check.
- const T & SafeAt (size_t Index) const

  SafeAt() for const vectors.
- T & AutoAt (size_t Index, const T &DefaultValue={})

  Provides access to the element. If range check fails, will auto fill vector up to Index with DefaultValue.
- T ∗ RawData () noexcept

  Provides access to the internal buffer.
- const T ∗ RawData () const noexcept

  RawData() for const vectors.
- bool operator== (const TVector< T > &Other) const noexcept

  Checks if two vectors contain the same values by calling equality operator recursively.
- bool operator!= (const TVector< T > &Other) const noexcept

  Opposite to operator ==.
- TVector< T > & operator+= (const TVector< T > &Other)

  Concatenates vectors (push 1 with 2)
- TVector< T > operator+ (const TVector< T > &Other) const

  Concatenates vectors (push 1 with 2)
- void Push (const T &Value)

Adds one element to the end of vector.

- template<typename IteratorType >
  void Push (IteratorType Begin, IteratorType End)

  Adds multiple elements to the end via iterators.
- void Insert (size_t Position, const T &Value)

  Inserts one element to the specified position.
- void SafeInsert (size_t Position, const T &Value)

  Insert() with range check.
- void AutoInsert (size_t Position, const T &Value, const T &DefaultValue={})

  Inserts element, extends vector if range check failed.
- template<typename IteratorType >
  void Insert (size_t Position, IteratorType Begin, IteratorType End)

  Inserts range of elements, starting at Position.
- template<typename IteratorType >
  void SafeInsert (size_t Position, IteratorType Begin, IteratorType End)

  Insert() with range check.
- template<typename IteratorType >
  void AutoInsert (size_t Position, IteratorType Begin, IteratorType End, const T &DefaultValue={})

  Inserts range of elements, starting at Position. Extends vector if range check failed.
- void Pop (EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  Removes one element from the end of vector.
- void SafePop (EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  Pop() with range check.
- T SafePopGet (EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  SafePop() that returns removed value.
- void PopMultiple (size_t ElementsCount, EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  Removes N elements from the end of vector.
- void Shift (EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  Removes one element from the beginning of vector.
- void SafeShift (EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  Shift() with range check.
- T SafeShiftGet (EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  SafeShift() that returns removed value.
- void ShiftMultiple (size_t ElementsToShift, EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  Removes N elements from the beginning of vector.
- void Erase (size_t Position, EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  Removes element with specified position.
- void SafeErase (size_t Position, EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  Erase() with range check.
- T SafeEraseGet (size_t Position, EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  SafeErase() that returns removed value.
- void EraseMultiple (size_t PositionFrom, size_t PositionTo, EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  Removes range of elements from vector.
- void Reserve (size_t NewCapacity)

  Allocates elements internally for the future use.
- void Resize (size_t NewSize, const T &DefaultValue={}, EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  Changes size of vector.
- void Swap (TVector< T > &Other) noexcept

  Swaps two vectors internally without deep copy.
- void ShrinkToFit ()

  Clears memory that was reserved for future use.

- void Clear (EShrinkBehavior ShrinkBehavior=EShrinkBehavior::Allow)

  Removes all elements from the vector.
- size_t GetSize () const noexcept

  Size is number of elements that you can use.
- size_t GetCapacity () const noexcept

  Capacity is Size + reserved space for the future use.
- bool IsEmpty () const noexcept

  Simple check if size of this vector equals 0.
- void SetCapacityRule (EReservedCapacityRule CapacityRule) noexcept

  Set enum value that will describe how re-allocation works when elements are added / removed from vertor. Reallocation on removal happens only if bool bAllowAutoShrink was passed with supported operation.
- EReservedCapacityRule GetCapacityRule () const noexcept

  Returns Capacity rule that is currently applied. It affects how elements are allocated & destructed.
- T & Front ()

  Provides access to the first element.
- const T & Front () const

  Front() for const vectors.
- T & SafeFront ()

  Front() with range check.
- const T & SafeFront () const

  SafeFront() for const vectors.
- T & Back ()

  Provides access to the last element.
- const T & Back () const

  Back() for const vectors.
- T & SafeBack ()

  Back() with range check.
- const T & SafeBack () const

  SafeBack() for const vectors.
- CIterator Begin ()

  Iterator pointing to the first elem.
- CConstIterator ConstBegin () const

  Iterator pointing to the first elem (const).
- CReverseIterator ReverseBegin ()

  Reverse iterator pointing to the first elem.
- CConstReverseIterator ConstReverseBegin () const

  Reverse iterator pointing to the first elem (const).
- CSafeIterator SafeBegin ()

  Safe iterator pointing to the first elem.
- CSafeConstIterator SafeConstBegin () const

  Safe iterator pointing to the first elem (const).
- CSafeReverseIterator SafeReverseBegin ()

  Safe reverse iterator pointing to the first elem.
- CSafeConstReverseIterator SafeConstReverseBegin () const

  Safe reverse iterator pointing to the first elem (const).
- CIterator End ()

  Iterator pointing to the elem after last.
- CConstIterator ConstEnd () const

  Iterator pointing to the elem after last (const).
- CReverseIterator ReverseEnd ()

  Reverse iterator pointing to the elem after last.

- CConstReverseIterator ConstReverseEnd () const

  Reverse iterator pointing to the elem after last (const).
- CSafeIterator SafeEnd ()

  Safe iterator pointing to the elem after last.
- CSafeConstIterator SafeConstEnd () const

  Safe iterator pointing to the elem after last (const).
- CSafeReverseIterator SafeReverseEnd ()

  Safe reverse iterator pointing to the elem after last.
- CSafeConstReverseIterator SafeConstReverseEnd () const

  Safe reverse it. pointing to elem after last (const).
- CIterator begin ()

  Begin() alias (for compatibity)
- CConstIterator begin () const

  ConstBegin() alias (overloaded, for compatibity)
- CConstIterator cbegin () const

  ConstBegin() alias (for compatibity)
- CReverseIterator rbegin ()

  ReverseBegin() alias (for compatibity)
- CConstReverseIterator rbegin () const

  ConstReverseBegin() alias (overloaded, for compatibity)
- CConstReverseIterator crbegin () const

  ConstReverseBegin() alias (for compatibity)
- CIterator end ()

  End() alias (for compatibity)
- CConstIterator end () const

  ConstEnd() alias (overloaded, for compatibity)
- CConstIterator cend () const

  ConstEnd() alias (for compatibity)
- CReverseIterator rend ()

  ReverseEnd() alias (for compatibity)
- CConstReverseIterator rend () const

  ConstReverseEnd() alias (overloaded, for compatibity)
- CConstReverseIterator crend () const

  ConstReverseEnd() alias (for compatibity)

9.14.1 Detailed Description

template<typename T>
class Common::TVector< T >

Container representing array that can change its size.

Fast in accessing elements as offsets are used, but not very effective in terms of adding and removing elements. To minimize those drawbacks, there are memory reserving rules that you can manage.

Unlike std::vector, this has utilities to manage capacity outside. EReservedCapacityRule describes how extra capacity is reserved. This value is class member (get/set are possible). EShrinkBehavior can be passed to every function that is supposed to decrease TVector capacity, overriding EReservedCapacityRule in terms of removing elements in this specific case.

Exception policy: TVector stays in the previous state if construction fails. TVector is cleared if move construction of the underlying object failed. TVector is in the broken state if exception occured from TVector constructor, but destruction is handled correctly. Rethrows original exception. Capacity is not changed if CapacityRule is NOT NeverReserve.

Note

If the logical result of operation may not be determined, there are two methods: one that throws an exception ("Safe" prefix) and another one that does assertion in debug mode. Custom exception is COutOfRange, derived from CException.

Underlying object must be copy and move constructible.

Todo In case of construction errors, do not decrease capacity unless CapacityRule is set to NeverReserve. Capacity management is not consistent now, especially if move operation throws

Todo Implement SFINAE to support types without nonparam ctor and types without overloaded == operator

Bug Move may not be performed

9.14.2 Member Typedef Documentation

9.14.2.1 value_type

```
template<typename T >
typedef T Common::TVector< T >::value_type
```

9.14.2.2 CIterator

```
template<typename T >
typedef Iterators::TBlockIterator<T*, T&> Common::TVector< T >::CIterator
```

Iterator. Implemented op-s: ++, +=, +, --, -=, -, ==, !=, =.

9.14.2.3 CConstIterator

```
template<typename T >
typedef Iterators::TBlockIterator<const T*, const T&> Common::TVector< T >::CConstIterator
```

Version of CIterator for const values.

9.14.2.4 CReverseIterator

```
template<typename T >
typedef Iterators::TReverseBlockIterator<T*, T&> Common::TVector< T >::CReverseIterator
```

Reverse iterator. Increment is actually decrement, etc.

9.14.2.5 CConstReverseIterator

```
template<typename T >
typedef Iterators::TReverseBlockIterator<const T*, const T&> Common::TVector< T >::CConstReverseIterator
```

Version of TReverseIterator for const values.

9.14.2.6 CSafeIterator

```
template<typename T >
typedef Iterators::TSafeBlockIterator<T*, T&, TVector<T>*> Common::TVector< T >::CSafeIterator
```

Iterator that does bounds checking and throws OutOfRange().

9.14.2.7 CSafeConstIterator

```
template<typename T >
typedef Iterators::TSafeBlockIterator<const T*, const T&, const TVector<T>*> Common::TVector<
T >::CSafeConstIterator
```

Version of TSafeIterator for const values.

9.14.2.8 CSafeReverseIterator

```
template<typename T >
typedef Iterators::TSafeReverseBlockIterator<T*, T&, TVector<T>*> Common::TVector< T >↩
::CSafeReverseIterator
```

Reverse iterator that can throw OutOfRange().

9.14.2.9 CSafeConstReverseIterator

```
template<typename T >
typedef Iterators::TSafeReverseBlockIterator<const T*, const T&, const TVector<T>*> Common::TVector<
T >::CSafeConstReverseIterator
```

Version of TSafeReverseIterator for const values.

9.14.3 Member Enumeration Documentation

9.14.3.1 EReservedCapacityRule

```
template<typename T >
enum Common::TVector::EReservedCapacityRule :  uint8_t  [strong]
```

Defines how extra Capacity is reserved.

Enumerator

Exponential	[ADD] allocates (NewSize-1)∗2 if capacity exceed and vector is not small enough [DEL] allocates 2∗Size if Capacity > 4 ∗ Size and vector is not small enough
Linear	[ADD] allocates NewSize + 3 + 32 / sizeof(T) if capacity exceed [DEL] allocates Size + 4 + 32 / sizeof(T) if size exceeds capacity by more than 2∗(8 + 64 / sizeof(T))
NeverReserve	[ADD, DEL] memory is never reserved automatically; Decrease of vector size causes ShrinkToFit() immediately

9.14.3.2 EShrinkBehavior

```
template<typename T >
enum Common::TVector::EShrinkBehavior  [strong]
```

Overrides CapacityRule for specific element removal case.

Enumerator

Require	Force reallocation if possible. Same as calling ShrinkToFit() after each operation or setting EReservedCapacityRule to NeverReserve
Allow	Used as default value. Inherits behavior from EReservedCapacityRule. **See also** EReservedCapacityRule for more info about presets.
Deny	Do not deallocate memory, even if EReservedCapacityRule prescribes that.

9.14.4 Constructor & Destructor Documentation

9.14.4.1 TVector() [1/7]

```
template<typename T >
Common::TVector< T >::TVector (
            EReservedCapacityRule CapacityRule = EReservedCapacityRule::Exponential )  [noexcept]
```

Creates empty vector with Capacity preset predefined.

Parameters

CapacityRule	Optional. Describes how memory is reserved

See also

EReservedCapacityRule for more info about presets.

9.14.4.2 TVector() [2/7]

```
template<typename T >
Common::TVector< T >::TVector (
             size_t Size,
             const T & DefaultValue = {},
             EReservedCapacityRule CapacityRule = EReservedCapacityRule::Exponential )
```

Vector with pre-created elements.

Parameters

Size	Number of elements to allocate
DefaultValue	Value to initialize with
CapacityRule	Optional. Describes how memory is reserved

See also

EReservedCapacityRule for more info about presets.

9.14.4.3 TVector() [3/7]

```
template<typename T >
Common::TVector< T >::TVector (
             size_t Size,
             const T *const Array,
             EReservedCapacityRule CapacityRule = EReservedCapacityRule::Exponential )
```

Vector constructed from raw dynamic array (copy).

Parameters

Size	Number of elements in original array
Array	Pointer to heap with C-style array
CapacityRule	Optional. Describes how memory is reserved

Note

Array[0] to Array[Size-1] must exist and have the same type as vector value_type.

See also

EReservedCapacityRule for more info about presets.

9.14.4.4 TVector() `[4/7]`

```
template<typename T >
Common::TVector< T >::TVector (
            const std::initializer_list< T > & ValuesList )
```

Modern C++ initialization syntax: name = {...}.

Parameters

ValuesList	Initializer list

9.14.4.5 TVector() `[5/7]`

```
template<typename T >
template<typename IteratorType >
Common::TVector< T >::TVector (
            IteratorType Begin,
            IteratorType End,
            EReservedCapacityRule CapacityRule = EReservedCapacityRule::Exponential,
            typename std::enable_if<!std::is_integral< IteratorType >::value >::type *  = 0
)
```

Constructor to get values from another container.

Template Parameters

IteratorType	Iterator that implements ++, != and *

Parameters

Begin	Iterator referring to the beginning of container
End	Iterator referring to the end of container
CapacityRule	Optional. Describes how memory is reserved

See also

EReservedCapacityRule for more info about presets.

9.14.4.6 TVector() [6/7]

```
template<typename T >
Common::TVector< T >::TVector (
            const TVector< T > & Other )
```

Initialize by copying another TVector.

Parameters

Other	vector to copy

9.14.4.7 TVector() [7/7]

```
template<typename T >
Common::TVector< T >::TVector (
            TVector< T > && Other )  [noexcept]
```

Move constructor.

Parameters

Other	Temporary object to get data from

9.14.4.8 ∼TVector()

```
template<typename T >
Common::TVector< T >::∼TVector ( )
```

9.14.5 Member Function Documentation

9.14.5.1 Assign()

```
template<typename T >
template<typename IteratorType >
void Common::TVector< T >::Assign (
            IteratorType Begin,
            IteratorType End,
            EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

Allows to copy values from another container.

Template Parameters

IteratorType	Iterator that implements ++, != and ∗

Parameters

Begin	Iterator referring to the beginning of container
End	Iterator referring to the end of container
ShrinkBehavior	Optional. Describes how memory is freed

See also

ShrinkBehavior for more info about patterns.

9.14.5.2 operator=() [1/3]

```
template<typename T >
TVector<T>& Common::TVector< T >::operator= (
             const std::initializer_list< T > & ValuesList )
```

Allows assignment with = {...} style.

Parameters

ValuesList	Initializer list

Returns

Reference to this vector

9.14.5.3 operator=() [2/3]

```
template<typename T >
TVector<T>& Common::TVector< T >::operator= (
             const TVector< T > & Other )
```

Makes a copy of another vector.

Parameters

Other	vector

Returns

Reference to this vector

9.14.5.4 operator=() [3/3]

```
template<typename T >
TVector<T>& Common::TVector< T >::operator= (
            TVector< T > && Other )  [noexcept]
```

Move assignment.

Parameters

Other	Temporary object to get data from

Returns

Reference to this vector

9.14.5.5 operator[]() [1/2]

```
template<typename T >
T& Common::TVector< T >::operator[] (
            size_t Index )
```

Index operator that prodives access to the element.

Parameters

Index	aka offset value

Returns

Reference to the requested element

Note

Element with requested index must exist in vector.

9.14.5.6 operator[]() [2/2]

```
template<typename T >
const T& Common::TVector< T >::operator[] (
             size_t Index ) const  [inline]
```

Index operator for const vectors.

9.14.5.7 SafeAt() [1/2]

```
template<typename T >
T& Common::TVector< T >::SafeAt (
             size_t Index )
```

[] with range check.

9.14.5.8 SafeAt() [2/2]

```
template<typename T >
const T& Common::TVector< T >::SafeAt (
             size_t Index ) const  [inline]
```

SafeAt() for const vectors.

9.14.5.9 AutoAt()

```
template<typename T >
T& Common::TVector< T >::AutoAt (
             size_t Index,
             const T & DefaultValue = {} )
```

Provides access to the element. If range check fails, will auto fill vector up to Index with DefaultValue.

Parameters

Index	Element index
DefaultValue	Value to initialize added elements

Returns

Reference to the requested element

See also

Use operator [] if you are sure that element exists

9.14.5.10 RawData() [1/2]

```
template<typename T >
T* Common::TVector< T >::RawData ( )  [noexcept]
```

Provides access to the internal buffer.

Returns

Pointer to c-style heap array

9.14.5.11 RawData() [2/2]

```
template<typename T >
const T* Common::TVector< T >::RawData ( ) const  [inline], [noexcept]
```

RawData() for const vectors.

9.14.5.12 operator==()

```
template<typename T >
bool Common::TVector< T >::operator== (
             const TVector< T > & Other ) const  [noexcept]
```

Checks if two vectors contain the same values by calling equality operator recursively.

Parameters

Other	Other vector to compare

Returns

True if sizes and values are equal, false otherwise

Note

Containing elements must implement == operator

9.14.5.13 operator"!=()

```
template<typename T >
bool Common::TVector< T >::operator!= (
             const TVector< T > & Other ) const  [noexcept]
```

Opposite to operator ==.

9.14.5.14 operator+=()

```
template<typename T >
TVector<T>& Common::TVector< T >::operator+= (
             const TVector< T > & Other )
```

Concatenates vectors (push 1 with 2)

Parameters

Other	Other vector to copy values from

Returns

Reference to this vector

9.14.5.15 operator+()

```
template<typename T >
TVector<T> Common::TVector< T >::operator+ (
             const TVector< T > & Other ) const
```

Concatenates vectors (push 1 with 2)

Parameters

Other	Other vector to copy values from

Returns

New vector, containing elements from both vectors

9.14.5.16 Push() [1/2]

```
template<typename T >
void Common::TVector< T >::Push (
             const T & Value )
```

Adds one element to the end of vector.

Parameters

Value	Element to add

See also

Call ShrinkToFit() to clear reserved memory, Reserve() to increase its amount

9.14.5.17 Push() [2/2]

```
template<typename T >
template<typename IteratorType >
void Common::TVector< T >::Push (
             IteratorType Begin,
             IteratorType End )
```

Adds multiple elements to the end via iterators.

Template Parameters

IteratorType	Iterator with implemented ++, != and ∗

Parameters

Begin	Iterator referring to the first element
End	Iterator referring to the element after last one

9.14.5.18 Insert() [1/2]

```
template<typename T >
void Common::TVector< T >::Insert (
             size_t Position,
             const T & Value )
```

Inserts one element to the specified position.

Parameters

Position	Index where to insert
Value	Value to insert

Note

Position must not exceed Size

9.14.5.19 SafeInsert() [1/2]

```
template<typename T >
void Common::TVector< T >::SafeInsert (
             size_t Position,
             const T & Value )
```

Insert() with range check.

9.14.5.20 AutoInsert() [1/2]

```
template<typename T >
void Common::TVector< T >::AutoInsert (
             size_t Position,
             const T & Value,
             const T & DefaultValue = {} )
```

Inserts element, extends vector if range check failed.

Parameters

Position	
Value	Value to insert
DefaultValue	Value to fill with if Position > Size

See also

Insert() if you are sure that Position <= Size

9.14.5.21 Insert() [2/2]

```
template<typename T >
template<typename IteratorType >
void Common::TVector< T >::Insert (
             size_t Position,
             IteratorType Begin,
             IteratorType End )
```

Inserts range of elements, starting at Position.

Template Parameters

IteratorType	Iterator with implemented ++, != and ∗

Parameters

Position	Index of the first inserted element
Begin	Iterator referring to the first element
End	Iterator referring to the element after last one

Note

Position must not exceed Size

9.14.5.22 SafeInsert() [2/2]

```
template<typename T >
template<typename IteratorType >
void Common::TVector< T >::SafeInsert (
            size_t Position,
            IteratorType Begin,
            IteratorType End )
```

Insert() with range check.

9.14.5.23 AutoInsert() [2/2]

```
template<typename T >
template<typename IteratorType >
void Common::TVector< T >::AutoInsert (
            size_t Position,
            IteratorType Begin,
            IteratorType End,
            const T & DefaultValue = {} )
```

Inserts range of elements, starting at Position. Extends vector if range check failed.

Template Parameters

IteratorType	Iterator with implemented ++, != and ∗

Parameters

Position	Index of the first inserted element
Begin	Iterator referring to the first element

Parameters

End	Iterator referring to the element after last one
DefaultValue	Value to fill with if Position > Size

See also

Insert() if you are sure that Position <= Size

9.14.5.24 Pop()

```
template<typename T >
void Common::TVector< T >::Pop (
            EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

Removes one element from the end of vector.

Parameters

ShrinkBehavior	Optional. Describes how memory is freed

Note

Vector must not be empty.

9.14.5.25 SafePop()

```
template<typename T >
void Common::TVector< T >::SafePop (
            EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

Pop() with range check.

9.14.5.26 SafePopGet()

```
template<typename T >
T Common::TVector< T >::SafePopGet (
            EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

SafePop() that returns removed value.

9.14.5.27 PopMultiple()

```
template<typename T >
void Common::TVector< T >::PopMultiple (
             size_t ElementsCount,
             EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

Removes N elements from the end of vector.

Parameters

ElementsCount	Number of elements to be removed
ShrinkBehavior	Optional. Describes how memory is freed

Note

If ElementsCount >= Size, clears vector

9.14.5.28 Shift()

```
template<typename T >
void Common::TVector< T >::Shift (
             EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

Removes one element from the beginning of vector.

Parameters

ShrinkBehavior	Optional. Describes how memory is freed

Note

Vector must not be empty.

9.14.5.29 SafeShift()

```
template<typename T >
void Common::TVector< T >::SafeShift (
             EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

Shift() with range check.

9.14.5.30 SafeShiftGet()

```
template<typename T >
T Common::TVector< T >::SafeShiftGet (
            EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

SafeShift() that returns removed value.

9.14.5.31 ShiftMultiple()

```
template<typename T >
void Common::TVector< T >::ShiftMultiple (
            size_t ElementsToShift,
            EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

Removes N elements from the beginning of vector.

Parameters

ElementsCount	Number of elements to be removed
ShrinkBehavior	Optional. Describes how memory is freed

Note

If ElementsCount >= Size, clears vector

9.14.5.32 Erase()

```
template<typename T >
void Common::TVector< T >::Erase (
            size_t Position,
            EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

Removes element with specified position.

Parameters

Position	Position of element to be removed
ShrinkBehavior	Optional. Describes how memory is freed

Attention

This method removes one element. To remove multiple, use EraseMultiple(). Your code with such a mistake may be compiled because of optional param.

9.14.5.33 SafeErase()

```
template<typename T >
void Common::TVector< T >::SafeErase (
             size_t Position,
             EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

Erase() with range check.

9.14.5.34 SafeEraseGet()

```
template<typename T >
T Common::TVector< T >::SafeEraseGet (
             size_t Position,
             EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

SafeErase() that returns removed value.

9.14.5.35 EraseMultiple()

```
template<typename T >
void Common::TVector< T >::EraseMultiple (
             size_t PositionFrom,
             size_t PositionTo,
             EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

Removes range of elements from vector.

Parameters

PositionFrom	Starting index for erase
PositionTo	End point for erase (after the last element)
ShrinkBehavior	Optional. Describes how memory is freed

Note

Ignores elements at unavailable positions.

Attention

This method removes multiple elements. To remove one, use Erase(). Your code with such a mistake may be compiled because of optional param.

9.14.5.36 Reserve()

```
template<typename T >
void Common::TVector< T >::Reserve (
             size_t NewCapacity )
```

Allocates elements internally for the future use.

Parameters

NewCapacity	If greater than size, will update internal capacity

See also

Call ShrinkToFit() to clear reserved memory.

Attention

Upon elements removal, vector can be shrinked if CapacityRule and ShrinkBehavior allow that

9.14.5.37 Resize()

```
template<typename T >
void Common::TVector< T >::Resize (
             size_t NewSize,
             const T & DefaultValue = {},
             EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

Changes size of vector.

Parameters

NewSize	New size of vector. If NewSize $<$ Size, deletes last elements. Otherwise, creates new with passed value.
DefaultValue	Value to initialize added elements
ShrinkBehavior	Optional. Describes how memory is freed

9.14.5.38 Swap()

```
template<typename T >
void Common::TVector< T >::Swap (
             TVector< T > & Other )  [noexcept]
```

Swaps two vectors internally without deep copy.

Parameters

Other	Object to swap resources with

9.14.5.39 ShrinkToFit()

```
template<typename T >
void Common::TVector< T >::ShrinkToFit ( )
```

Clears memory that was reserved for future use.

See also

Methods that can reserve memory: Push(), Reserve(), etc.

9.14.5.40 Clear()

```
template<typename T >
void Common::TVector< T >::Clear (
             EShrinkBehavior ShrinkBehavior = EShrinkBehavior::Allow )
```

Removes all elements from the vector.

Parameters

ShrinkBehavior	Optional. Describes how memory is freed

9.14.5.41 GetSize()

```
template<typename T >
size_t Common::TVector< T >::GetSize ( ) const [noexcept]
```

Size is number of elements that you can use.

Returns

Size of vector

9.14.5.42 GetCapacity()

```
template<typename T >
size_t Common::TVector< T >::GetCapacity ( ) const  [noexcept]
```

Capacity is Size + reserved space for the future use.

Returns

Capacity of vector

9.14.5.43 IsEmpty()

```
template<typename T >
bool Common::TVector< T >::IsEmpty ( ) const  [noexcept]
```

Simple check if size of this vector equals 0.

Returns

True if empty, false if not

9.14.5.44 SetCapacityRule()

```
template<typename T >
void Common::TVector< T >::SetCapacityRule (
             EReservedCapacityRule CapacityRule )  [noexcept]
```

Set enum value that will describe how re-allocation works when elements are added / removed from vertor. Reallocation on removal happens only if bool bAllowAutoShrink was passed with supported operation.

Parameters

CapacityRule	Preset value from EReservedCapacityRule

Note

If bAllowAutoShrink was passed with operation, size that you have manually reserved may also be deallocated

9.14.5.45 GetCapacityRule()

```
template<typename T >
EReservedCapacityRule Common::TVector< T >::GetCapacityRule ( ) const  [noexcept]
```

Returns Capacity rule that is currently applied. It affects how elements are allocated & destructed.

Returns

EReservedCapacityRule Current capacity rule

See also

EReservedCapacityRule for more info about presets.

9.14.5.46 Front() [1/2]

```
template<typename T >
T& Common::TVector< T >::Front ( )
```

Provides access to the first element.

Returns

Reference to the first element

Note

Vector must not be empty.

9.14.5.47 Front() [2/2]

```
template<typename T >
const T& Common::TVector< T >::Front ( ) const  [inline]
```

Front() for const vectors.

9.14.5.48 SafeFront() [1/2]

```
template<typename T >
T& Common::TVector< T >::SafeFront ( )
```

Front() with range check.

9.14.5.49 SafeFront() [2/2]

```
template<typename T >
const T& Common::TVector< T >::SafeFront ( ) const  [inline]
```

SafeFront() for const vectors.

9.14.5.50 Back() [1/2]

```
template<typename T >
T& Common::TVector< T >::Back ( )
```

Provides access to the last element.

Returns

Reference to the last element

Note

Vector must not be empty.

9.14.5.51 Back() [2/2]

```
template<typename T >
const T& Common::TVector< T >::Back ( ) const  [inline]
```

Back() for const vectors.

9.14.5.52 SafeBack() [1/2]

```
template<typename T >
T& Common::TVector< T >::SafeBack ( )
```

Back() with range check.

9.14.5.53 SafeBack() [2/2]

```
template<typename T >
const T& Common::TVector< T >::SafeBack ( ) const  [inline]
```

SafeBack() for const vectors.

9.14.5.54 Begin()

```
template<typename T >
CIterator Common::TVector< T >::Begin ( )
```

Iterator pointing to the first elem.

Returns

CIterator iterator

9.14.5.55 ConstBegin()

```
template<typename T >
CConstIterator Common::TVector< T >::ConstBegin ( ) const
```

Iterator pointing to the first elem (const).

Returns

CConstIterator iterator

9.14.5.56 ReverseBegin()

```
template<typename T >
CReverseIterator Common::TVector< T >::ReverseBegin ( )
```

Reverse iterator pointing to the first elem.

Returns

CReverseIterator iterator

9.14.5.57 ConstReverseBegin()

```
template<typename T >
CConstReverseIterator Common::TVector< T >::ConstReverseBegin ( ) const
```

Reverse iterator pointing to the first elem (const).

Returns

CConstReverseIterator iterator

9.14.5.58 SafeBegin()

```
template<typename T >
CSafeIterator Common::TVector< T >::SafeBegin ( )
```

Safe iterator pointing to the first elem.

Returns

CSafeIterator iterator

9.14.5.59 SafeConstBegin()

```
template<typename T >
CSafeConstIterator Common::TVector< T >::SafeConstBegin ( ) const
```

Safe iterator pointing to the first elem (const).

Returns

CSafeConstIterator iterator

9.14.5.60 SafeReverseBegin()

```
template<typename T >
CSafeReverseIterator Common::TVector< T >::SafeReverseBegin ( )
```

Safe reverse iterator pointing to the first elem.

Returns

CSafeReverseIterator iterator

9.14.5.61 SafeConstReverseBegin()

```
template<typename T >
CSafeConstReverseIterator Common::TVector< T >::SafeConstReverseBegin ( ) const
```

Safe reverse iterator pointing to the first elem (const).

Returns

CSafeConstReverseIterator

9.14.5.62 End()

```
template<typename T >
CIterator Common::TVector< T >::End ( )
```

Iterator pointing to the elem after last.

Returns

CIterator iterator

9.14.5.63 ConstEnd()

```
template<typename T >
CConstIterator Common::TVector< T >::ConstEnd ( ) const
```

Iterator pointing to the elem after last (const).

Returns

CConstIterator iterator

9.14.5.64 ReverseEnd()

```
template<typename T >
CReverseIterator Common::TVector< T >::ReverseEnd ( )
```

Reverse iterator pointing to the elem after last.

Returns

CReverseIterator iterator

9.14.5.65 ConstReverseEnd()

```
template<typename T >
CConstReverseIterator Common::TVector< T >::ConstReverseEnd ( ) const
```

Reverse iterator pointing to the elem after last (const).

Returns

CConstReverseIterator iterator

9.14.5.66 SafeEnd()

```
template<typename T >
CSafeIterator Common::TVector< T >::SafeEnd ( )
```

Safe iterator pointing to the elem after last.

Returns

CSafeIterator iterator

9.14.5.67 SafeConstEnd()

```
template<typename T >
CSafeConstIterator Common::TVector< T >::SafeConstEnd ( ) const
```

Safe iterator pointing to the elem after last (const).

Returns

CSafeConstIterator iterator

9.14.5.68 SafeReverseEnd()

```
template<typename T >
CSafeReverseIterator Common::TVector< T >::SafeReverseEnd ( )
```

Safe reverse iterator pointing to the elem after last.

Returns

CSafeReverseIterator iterator

9.14.5.69 SafeConstReverseEnd()

```
template<typename T >
CSafeConstReverseIterator Common::TVector< T >::SafeConstReverseEnd ( ) const
```

Safe reverse it. pointing to elem after last (const).

Returns

CSafeConstReverseIterator

9.14.5.70 begin() [1/2]

```
template<typename T >
CIterator Common::TVector< T >::begin ( )  [inline]
```

Begin() alias (for compatibity)

9.14.5.71 begin() [2/2]

```
template<typename T >
CConstIterator Common::TVector< T >::begin ( ) const  [inline]
```

ConstBegin() alias (overloaded, for compatibity)

9.14.5.72 cbegin()

```
template<typename T >
CConstIterator Common::TVector< T >::cbegin ( ) const  [inline]
```

ConstBegin() alias (for compatibity)

9.14.5.73 rbegin() [1/2]

```
template<typename T >
CReverseIterator Common::TVector< T >::rbegin ( )  [inline]
```

ReverseBegin() alias (for compatibity)

9.14.5.74 rbegin() [2/2]

```
template<typename T >
CConstReverseIterator Common::TVector< T >::rbegin ( ) const  [inline]
```

ConstReverseBegin() alias (overloaded, for compatibity)

9.14.5.75 crbegin()

```
template<typename T >
CConstReverseIterator Common::TVector< T >::crbegin ( ) const  [inline]
```

ConstReverseBegin() alias (for compatibity)

9.14.5.76 end() [1/2]

```
template<typename T >
CIterator Common::TVector< T >::end ( )  [inline]
```

End() alias (for compatibity)

9.14.5.77 end() [2/2]

```
template<typename T >
CConstIterator Common::TVector< T >::end ( ) const  [inline]
```

ConstEnd() alias (overloaded, for compatibity)

9.14.5.78 cend()

```
template<typename T >
CConstIterator Common::TVector< T >::cend ( ) const  [inline]
```

ConstEnd() alias (for compatibity)

9.14.5.79 rend() [1/2]

```
template<typename T >
CReverseIterator Common::TVector< T >::rend ( )  [inline]
```

ReverseEnd() alias (for compatibity)

9.14.5.80 rend() [2/2]

```
template<typename T >
CConstReverseIterator Common::TVector< T >::rend ( ) const  [inline]
```

ConstReverseEnd() alias (overloaded, for compatibity)

9.14.5.81 crend()

```
template<typename T >
CConstReverseIterator Common::TVector< T >::crend ( ) const  [inline]
```

ConstReverseEnd() alias (for compatibity)

Chapter 10

File Documentation

10.1 CommonTypes/Exception.h File Reference

```
#include "Pair.h"
#include "../CommonUtils/RawString.h"
```

Classes

- class Common::CException

  Basic exception class. C-style message is required.
- class Common::COutOfRange

  Represents "Out of Range" error. Can hold message, requested and expected indices.
- class Common::CBadAlloc

  Represents allocation failed error (usually rethrown from new)
- class Common::CDoesNotExist

  Represents "Element does not exist" error. Stores message.

Namespaces

- Common

10.2 CommonTypes/Iterators/Block.h File Reference

```
#include "../Private/Iterators/Block.tpp"
```

Classes

- class Common::Iterators::TBlockIterator< PtrType, RefType >
- class Common::Iterators::TReverseBlockIterator< PtrType, RefType >
- class Common::Iterators::TSafeBlockIterator< PtrType, RefType, ContType >
- class Common::Iterators::TSafeReverseBlockIterator< PtrType, RefType, ContType >

Namespaces

- Common
- Common::Iterators

10.3 CommonTypes/Optional.h File Reference

```
#include <new>
#include "Exception.h"
#include "./../CommonUtils/TypeOperations.h"
#include "Private/Optional.tpp"
```

Classes

- class Common::TOptional< T >

  Represents object that may not exist.

Namespaces

- Common

10.4 CommonTypes/Pair.h File Reference

Classes

- class Common::TPair< T1, T2 >

  Container that represents a pair of objects.

Namespaces

- Common

Functions

- template<typename T1 , typename T2 >
  TPair< T1, T2 > Common::MakePair (const T1 &First, const T2 &Second) noexcept

10.5 CommonTypes/Vector.h File Reference

```
#include <initializer_list>
#include <type_traits>
#include "Exception.h"
#include "Iterators/Block.h"
#include "../CommonUtils/Assert.h"
#include "./../CommonUtils/TypeOperations.h"
#include "./../CommonUtils/AdvancedIteration.h"
#include "./../CommonUtils/BlockAllocation.h"
#include "Private/Vector/Vector.tpp"
#include "Private/Vector/Iterator.tpp"
```

Classes

- class Common::TVector< T >

  Container representing array that can change its size.

Namespaces

- Common

10.6 CommonUtils/AdvancedIteration.h File Reference

```
#include "Private/AdvancedIteration.tpp"
```

Namespaces

- Common

Functions

- template<typename IteratorType >
  size_t Common::GetIteratorDistance (IteratorType Begin, IteratorType End)

  Counts elements between two iterators. Range: [Begin: End)

10.7 CommonUtils/Assert.h File Reference

```
#include <iostream>
```

Macros

- #define ASSERT(Condition, Message)

10.7.1 Macro Definition Documentation

10.7.1.1 ASSERT

```
#define ASSERT(
               Condition,
               Message )
```

Value:

```
    do { \
        if (!(Condition)) { \
            std::cerr « "ASSERTION (" #Condition ") FAILED in " « __FILE__ \
                « ", line " « __LINE__ « ": " « (Message) « std::endl; \
            std::abort(); \
        } \
    } while (false)
```

10.8 CommonUtils/BlockAllocation.h File Reference

```
#include <new>
#include "./../CommonTypes/Exception.h"
#include "TypeOperations.h"
#include "Private/BlockAllocation.tpp"
```

Namespaces

- Common

Functions

- template<typename T >
  void Common::Allocate (size_t NewSize, T ∗&OutBuffer)
- template<typename T >
  void Common::Deallocate (T ∗&OutBuffer) noexcept
- template<typename T >
  void Common::Construct (size_t Index, T ∗OutBuffer, const T &Value)
- template<typename T >
  void Common::Destruct (size_t Index, T ∗OutBuffer) noexcept
- template<typename T >
  void Common::DestructRange (size_t From, size_t To, T ∗OutBuffer) noexcept
- template<typename T >
  void Common::DestructAll (size_t Size, T ∗OutBuffer) noexcept
- template<typename T >
  void Common::SafeMoveBlock (size_t Size, T ∗FromBuffer, T ∗ToBuffer)
- template<typename T >
  void Common::SafeMoveBlockReverse (size_t Size, T ∗FromBuffer, T ∗ToBuffer)
- template<typename T >
  void Common::Reconstruct (size_t CopySize, size_t NewCapacity, T ∗&OutBuffer, size_t &OutCapacity, size_t &OutSize)
- template<typename IteratorType , typename T >
  void Common::SafeBulkConstruct (size_t StartPosition, IteratorType From, IteratorType To, T ∗OutBuffer)
- template<typename T >
  void Common::SafeFillConstruct (size_t StartPosition, size_t EndPosition, T ∗OutBuffer, const T &Value)

10.9 CommonUtils/RawString.h File Reference

Namespaces

- Common

Functions

- size_t Common::GetRawStringLength (const char ∗NullTermString)

  Calculates length of the C-string.
- size_t Common::GetRawStringLength (const char ∗NullTermString, size_t MaxLength)

  Calculates length of the C-string. Stops when null character is reached or MaxLength elements have been counted. Useful with malformatted input.
- void Common::CopyRawString (const char ∗NullTermStringFrom, char ∗const RawStringTo)

  Does the copy of C-style string (ended with '\0').
- void Common::CopyRawString (const char ∗NullTermStringFrom, char ∗RawStringTo, size_t MaxLength)

  Does the copy of C-style string. Stops when null character is reached or MaxLength elements have been copied. Useful when string should be cut.
- bool Common::AreRawStringsEqual (const char ∗NullTermString1, const char ∗NullTermString2)

  Checks whether two C-style strings are equal.
- bool Common::AreRawStringsEqual (const char ∗NullTermString1, const char ∗NullTermString2, size_↩
  t MaxCompareLength)

  Checks whether two C-style strings are equal. Use this to compare N first elements, even if strings are not null-terminated.

10.10 CommonUtils/Sort.h File Reference

```
#include "TypeOperations.h"
#include "AdvancedIteration.h"
#include "Private/Sort.tpp"
```

Namespaces

- Common

Functions

- template<typename IteratorType , typename FunctionType >
  void Common::BubbleSort (IteratorType Begin, IteratorType End, FunctionType Comparator)
- template<typename IteratorType , typename FunctionType >
  void Common::SelectionSort (IteratorType Begin, IteratorType End, FunctionType Comparator)
- template<typename IteratorType , typename FunctionType >
  void Common::QuickSort (IteratorType Begin, IteratorType End, FunctionType Comparator)

10.11 CommonUtils/TypeOperations.h File Reference

```
#include "Private/TypeOperations.tpp"
```

Classes

- struct Common::RemoveReference< T >
- struct Common::RemoveReference< T & >
- struct Common::RemoveReference< T && >

Namespaces

- Common

Functions

- template<typename T >
  RemoveReference< T >::Type && Common::Move (T &&Value)
- template<typename T >
  void Common::Swap (T &First, T &Second)

10.12 A:/Yuri - work/Desktop/CommonLibs/Pages.dox File Reference

Index