

Compile-Time Graph Library

Generated by Doxygen 1.8.20

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	7
3.1 Class List	7
4 Namespace Documentation	11
4.1 FunctorType Namespace Reference	11
4.1.1 Detailed Description	11
4.2 Objects Namespace Reference	11
4.2.1 Detailed Description	11
4.3 TL Namespace Reference	12
4.3.1 Detailed Description	12
5 Class Documentation	13
5.1 TL::Add< T, ind, Arg, Args > Struct Template Reference	13
5.1.1 Detailed Description	13
5.2 TL::Add< T, 0, TypeList< Arg, Args... > > Struct Template Reference	13
5.2.1 Detailed Description	13
5.3 TL::Add< T, 0, TypeList< Args... > > Struct Template Reference	14
5.3.1 Detailed Description	14
5.4 TL::Add< T, ind, TypeList< Arg, Args... > > Struct Template Reference	14
5.4.1 Detailed Description	14
5.5 GLib::AddEdge< GraphType, graph, edge > Struct Template Reference	15
5.5.1 Detailed Description	15
5.6 GLib::AddEdge< ADJACENCY_LIST, graph, edge > Struct Template Reference	15
5.6.1 Detailed Description	16
5.7 AdjacencyListGraph< vertexes, adjacency_list > Struct Template Reference	16
5.7.1 Detailed Description	17
5.7.2 Member Function Documentation	17
5.7.2.1 GetVertexIndex()	17
5.7.2.2 HasEdge()	18
5.8 AdjacencyMatrixGraph< vertexes, matrix > Struct Template Reference	18
5.8.1 Detailed Description	18
5.9 Objects::Boolean< boolean > Struct Template Reference	19
5.10 CheckContainsConstructibleParent< type_list, T, is_parent > Struct Template Reference	19
5.11 CheckContainsConstructibleParent< type_list, T, false > Struct Template Reference	19
5.11.1 Member Data Documentation	19
5.11.1.1 result	20
5.12 CheckContainsConstructibleParent< type_list, T, true > Struct Template Reference	20
5.13 CheckContainsParent< type_list, T, is_parent > Struct Template Reference	20

5.14 CheckContainsParent< type_list, T, false > Struct Template Reference	20
5.14.1 Member Data Documentation	20
5.14.1.1 result	21
5.15 CheckContainsParent< type_list, T, true > Struct Template Reference	21
5.16 CheckFindParentTypeList< contains_class, T, type_list, type_lists > Struct Template Reference	21
5.17 CheckFindParentTypeList< false, T, type_list, type_lists... > Struct Template Reference	21
5.18 CheckFindParentTypeList< true, T, type_list, type_lists... > Struct Template Reference	22
5.19 CheckFindTypeListByClass< contains_class, T, type_list, type_lists > Struct Template Reference	22
5.20 CheckFindTypeListByClass< false, T, type_list, type_lists... > Struct Template Reference	22
5.21 CheckFindTypeListByClass< true, T, type_list, type_lists... > Struct Template Reference	22
5.22 CheckHasDerivedAndConstructible< type_list, T, is_head_parent_of_T > Struct Template Reference	23
5.23 CheckHasDerivedAndConstructible< type_list, T, false > Struct Template Reference	23
5.24 CheckHasDerivedAndConstructible< type_list, T, true > Struct Template Reference	23
5.25 CheckIsBaseOf< has_parent, parent, derived > Struct Template Reference	23
5.26 CheckIsBaseOf< false, parent, derived > Struct Template Reference	24
5.27 CheckIsBaseOf< true, parent, derived > Struct Template Reference	24
5.27.1 Member Data Documentation	24
5.27.1.1 result	24
5.28 CheckMostDerived< type_list, T, is_head_parent_of_T > Struct Template Reference	24
5.29 CheckMostDerived< type_list, T, false > Struct Template Reference	25
5.30 CheckMostDerived< type_list, T, true > Struct Template Reference	25
5.31 CheckMostDerivedAndConstructible< type_list, T, is_head_parent_of_T > Struct Template Reference	25
5.32 CheckMostDerivedAndConstructible< type_list, T, false > Struct Template Reference	25
5.33 CheckMostDerivedAndConstructible< type_list, T, true > Struct Template Reference	26
5.34 Class< T > Struct Template Reference	26
5.34.1 Detailed Description	26
5.35 TL::Concatenate< front, back > Struct Template Reference	26
5.35.1 Detailed Description	27
5.36 TL::Concatenate< front, EmptyTypeList > Struct Template Reference	27
5.36.1 Detailed Description	27
5.37 TL::Contains< type_list, T > Struct Template Reference	27
5.37.1 Detailed Description	28
5.38 TL::ContainsConstructibleParent< type_list, T > Struct Template Reference	28
5.38.1 Detailed Description	28
5.38.2 Member Data Documentation	29
5.38.2.1 result	29
5.39 TL::ContainsConstructibleParent< EmptyTypeList, T > Struct Template Reference	29
5.40 TL::ContainsParent< type_list, T > Struct Template Reference	29
5.40.1 Detailed Description	29
5.40.2 Member Data Documentation	30
5.40.2.1 result	30
5.41 TL::ContainsParent< EmptyTypeList, T > Struct Template Reference	30

5.42 ConvertGraph< From, To, graph > Struct Template Reference	30
5.42.1 Detailed Description	30
5.43 ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph > Struct Template Reference	31
5.43.1 Detailed Description	31
5.44 ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph > Struct Template Reference	32
5.44.1 Detailed Description	32
5.45 ConvertGraph< POINTER_STRUCTURE, EDGE_LIST, graph > Struct Template Reference	32
5.45.1 Detailed Description	32
5.46 AdjacencyListGraph< vertexes, adjacency_list >::ConvertTo< GraphType > Struct Template Reference	33
5.46.1 Detailed Description	33
5.47 EdgeListGraph< vertexes, edge_list >::ConvertTo< GraphType > Struct Template Reference	34
5.47.1 Detailed Description	34
5.48 AdjacencyMatrixGraph< vertexes, matrix >::ConvertTo< GraphType > Struct Template Reference	34
5.48.1 Detailed Description	34
5.49 EdgeListGraph< vertexes, edge_list >::ConvertTo< ADJACENCY_LIST > Struct Reference	35
5.49.1 Detailed Description	35
5.50 AdjacencyMatrixGraph< vertexes, matrix >::ConvertTo< EDGE_LIST > Struct Reference	35
5.51 AdjacencyListGraph< vertexes, adjacency_list >::ConvertTo< POINTER_STRUCTURE > Struct Reference	36
5.52 PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes > Struct Template Reference	36
5.52.1 Detailed Description	36
5.53 Edge< from_, to_, weight_ > Struct Template Reference	37
5.53.1 Detailed Description	37
5.54 EdgeListGraph< vertexes, edge_list > Struct Template Reference	37
5.54.1 Detailed Description	38
5.55 TL::FindParentTypeList< T, type_list, type_lists > Struct Template Reference	38
5.55.1 Detailed Description	38
5.56 TL::FindTypeListByClass< T, type_list, type_lists > Struct Template Reference	39
5.56.1 Detailed Description	39
5.57 Functor< ResultType, ArgTypes > Class Template Reference	40
5.58 Functor< ResultType(ArgTypes...) > Class Template Reference	40
5.58.1 Detailed Description	40
5.58.2 Member Function Documentation	40
5.58.2.1 operator()()	40
5.59 TL::GenerateTypeLists< n > Struct Template Reference	41
5.59.1 Detailed Description	41
5.60 TL::GenerateTypeLists< 0 > Struct Reference	42
5.61 Graph Struct Reference	42
5.61.1 Detailed Description	42
5.62 TL::HasDerivedAndConstructible< type_list, T > Struct Template Reference	42
5.62.1 Detailed Description	42

5.62.2 Member Data Documentation	43
5.62.2.1 result	43
5.63 TL::HasDerivedAndConstructible< EmptyTypeList, T > Struct Template Reference	43
5.63.1 Detailed Description	43
5.64 TL::IndexOf< type_list, T > Struct Template Reference	44
5.64.1 Detailed Description	44
5.65 TL::IndexOf< EmptyTypeList, T > Struct Template Reference	44
5.65.1 Detailed Description	44
5.66 TL::IndexOf< type_list, typename type_list::Head > Struct Template Reference	45
5.66.1 Detailed Description	45
5.67 Objects::Integer< integer > Struct Template Reference	45
5.68 TL::IsBaseOf< parent, derived > Struct Template Reference	45
5.68.1 Detailed Description	45
5.68.2 Member Data Documentation	46
5.68.2.1 result	46
5.69 TL::IsBaseOf< EmptyTypeList, derived > Struct Template Reference	46
5.70 TL::IsBaseOf< EmptyTypeList, EmptyTypeList > Struct Reference	46
5.71 TL::IsBaseOf< parent, EmptyTypeList > Struct Template Reference	47
5.72 TL::IsTypeList< T > Struct Template Reference	47
5.72.1 Detailed Description	47
5.73 TL::IsTypeList< TypeList< Args... > > Struct Template Reference	47
5.74 PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >::IterateThroughChildren< current_children > Struct Template Reference	48
5.75 PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >::IterateThroughChildren< EmptyTypeList > Struct Reference	48
5.76 ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< edge_list > Struct Template Reference	48
5.77 ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< EmptyTypeList > Struct Reference	49
5.78 ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< current_vertexes, current_adjacency_list > Struct Template Reference	49
5.79 ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< EmptyTypeList, EmptyTypeList > Struct Reference	49
5.80 TL::MostDerived< type_list, T > Struct Template Reference	49
5.80.1 Detailed Description	49
5.81 TL::MostDerived< EmptyTypeList, T > Struct Template Reference	50
5.82 TL::MostDerivedAndConstructible< type_list, T > Struct Template Reference	50
5.82.1 Detailed Description	50
5.83 TL::MostDerivedAndConstructible< EmptyTypeList, T > Struct Template Reference	51
5.84 PointerStructureGraph< vertexes >::Node< vertex_, children_ > Struct Template Reference	51
5.84.1 Detailed Description	51
5.85 TL::NoDuplicates< type_list > Struct Template Reference	52
5.85.1 Detailed Description	52
5.86 TL::NoDuplicates< EmptyTypeList > Struct Reference	52

5.86.1 Detailed Description	53
5.87 NullType Struct Reference	53
5.87.1 Detailed Description	53
5.88 PointerStructureGraph< vertexes > Struct Template Reference	53
5.88.1 Detailed Description	53
5.89 TL::Remove< type_list, T > Struct Template Reference	54
5.89.1 Detailed Description	54
5.90 TL::Remove< EmptyTypeList, T > Struct Template Reference	54
5.90.1 Detailed Description	55
5.91 TL::Remove< type_list, typename type_list::Head > Struct Template Reference	55
5.91.1 Detailed Description	55
5.92 TL::RemoveAll< type_list, T > Struct Template Reference	55
5.92.1 Detailed Description	55
5.93 TL::RemoveAll< type_list, typename type_list::Head > Struct Template Reference	56
5.93.1 Detailed Description	56
5.94 TL::Replace< T, ind, Arg, Args > Struct Template Reference	56
5.94.1 Detailed Description	56
5.95 TL::Replace< T, 0, TypeList< Arg, Args... > > Struct Template Reference	57
5.95.1 Detailed Description	57
5.96 TL::Replace< T, ind, TypeList< Arg, Args... > > Struct Template Reference	57
5.96.1 Detailed Description	58
5.97 TL::Size< type_list > Struct Template Reference	58
5.97.1 Detailed Description	58
5.98 TL::Size< EmptyTypeList > Struct Reference	59
5.98.1 Detailed Description	59
5.99 TL::TypeAt< type_list, ind > Struct Template Reference	59
5.99.1 Detailed Description	59
5.100 TL::TypeAt< type_list, 0 > Struct Template Reference	60
5.100.1 Detailed Description	60
5.101 TypeList< Args > Struct Template Reference	60
5.101.1 Detailed Description	60
5.102 TypeList< H, T... > Struct Template Reference	61
5.102.1 Detailed Description	61
5.103 TypeList< T > Struct Template Reference	61
5.103.1 Detailed Description	61
5.104 VertexStream< stream, graph > Struct Template Reference	62
5.104.1 Detailed Description	62
5.104.2 Member Function Documentation	62
5.104.2.1 ForEach()	62
5.105 VertexStream< EmptyTypeList, graph > Struct Template Reference	63
5.105.1 Detailed Description	63

Chapter 1

Namespace Index

1.1 Namespace List

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

FunctorType	11
Objects	11
TL	12

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

TL::Add< T, ind, Arg, Args >	13
TL::Add< T, 0, TypeList< Arg, Args... > >	13
TL::Add< T, 0, TypeList< Args... > >	14
TL::Add< T, ind, TypeList< Arg, Args... > >	14
GLib::AddEdge< GraphType, graph, edge >	15
GLib::AddEdge< ADJACENCY_LIST, graph, edge >	15
AdjacencyMatrixGraph< vertexes, matrix >	18
Objects::Boolean< boolean >	19
CheckContainsConstructibleParent< type_list, T, is_parent >	19
CheckContainsConstructibleParent< type_list, T, false >	19
CheckContainsConstructibleParent< type_list, T, true >	20
CheckContainsParent< type_list, T, is_parent >	20
CheckContainsParent< type_list, T, false >	20
CheckContainsParent< type_list, T, true >	21
CheckFindParentTypeList< contains_class, T, type_list, type_lists >	21
CheckFindParentTypeList< false, T, type_list, type_lists... >	21
CheckFindParentTypeList< true, T, type_list, type_lists... >	22
CheckFindTypeListByClass< contains_class, T, type_list, type_lists >	22
CheckFindTypeListByClass< false, T, type_list, type_lists... >	22
CheckFindTypeListByClass< true, T, type_list, type_lists... >	22
CheckHasDerivedAndConstructible< type_list, T, is_head_parent_of_T >	23
CheckHasDerivedAndConstructible< type_list, T, false >	23
CheckHasDerivedAndConstructible< type_list, T, true >	23
CheckIsBaseOf< has_parent, parent, derived >	23
CheckIsBaseOf< false, parent, derived >	24
CheckIsBaseOf< true, parent, derived >	24
CheckMostDerived< type_list, T, is_head_parent_of_T >	24
CheckMostDerived< type_list, T, false >	25
CheckMostDerived< type_list, T, true >	25
CheckMostDerivedAndConstructible< type_list, T, is_head_parent_of_T >	25
CheckMostDerivedAndConstructible< type_list, T, false >	25
CheckMostDerivedAndConstructible< type_list, T, true >	26
Class< T >	26
TL::Concatenate< front, back >	26
TL::Concatenate< front, EmptyTypeList >	27

TL::Contains< type_list, T >	27
TL::ContainsConstructibleParent< type_list, T >	28
TL::ContainsConstructibleParent< EmptyTypeList, T >	29
TL::ContainsParent< type_list, T >	29
TL::ContainsParent< EmptyTypeList, T >	30
ConvertGraph< From, To, graph >	30
ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >	31
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >	32
ConvertGraph< POINTER_STRUCTURE, EDGE_LIST, graph >	32
AdjacencyListGraph< vertexes, adjacency_list >::ConvertTo< GraphType >	33
EdgeListGraph< vertexes, edge_list >::ConvertTo< GraphType >	34
AdjacencyMatrixGraph< vertexes, matrix >::ConvertTo< GraphType >	34
EdgeListGraph< vertexes, edge_list >::ConvertTo< ADJACENCY_LIST >	35
AdjacencyMatrixGraph< vertexes, matrix >::ConvertTo< EDGE_LIST >	35
AdjacencyListGraph< vertexes, adjacency_list >::ConvertTo< POINTER_STRUCTURE >	36
PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >	36
Edge< from_, to_, weight_ >	37
EdgeListGraph< vertexes, edge_list >	37
false_type	
TL::IsTypeList< T >	47
TL::FindParentTypeList< T, type_list, type_lists >	38
TL::FindTypeListByClass< T, type_list, type_lists >	39
Functor< ResultType, ArgTypes >	40
Functor< ResultType(ArgTypes...) >	40
TL::GenerateTypeLists< n >	41
TL::GenerateTypeLists< 0 >	42
Graph	42
AdjacencyListGraph< vertexes, adjacency_list >	16
PointerStructureGraph< vertexes >	53
TL::HasDerivedAndConstructible< type_list, T >	42
TL::HasDerivedAndConstructible< EmptyTypeList, T >	43
TL::IndexOf< type_list, T >	44
TL::IndexOf< EmptyTypeList, T >	44
TL::IndexOf< type_list, typename type_list::Head >	45
Objects::Integer< integer >	45
TL::IsBaseOf< parent, derived >	45
TL::IsBaseOf< EmptyTypeList, derived >	46
TL::IsBaseOf< EmptyTypeList, EmptyTypeList >	46
TL::IsBaseOf< parent, EmptyTypeList >	47
PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >::IterateThrough↔ Children< current_children >	48
PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >::IterateThrough↔ Children< EmptyTypeList >	48
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< edge_list >	48
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< EmptyTypeList >	49
ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< current_vertexes, current_adjacency_list >	49
ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< EmptyTypeList, EmptyTypeList >	49
TL::MostDerived< type_list, T >	49
TL::MostDerived< EmptyTypeList, T >	50
TL::MostDerivedAndConstructible< type_list, T >	50
TL::MostDerivedAndConstructible< EmptyTypeList, T >	51
PointerStructureGraph< vertexes >::Node< vertex_, children_ >	51
TL::NoDuplicates< type_list >	52
TL::NoDuplicates< EmptyTypeList >	52
NullType	53
TL::Remove< type_list, T >	54

TL::Remove< EmptyTypeList, T >	54
TL::Remove< type_list, typename type_list::Head >	55
TL::RemoveAll< type_list, T >	55
TL::RemoveAll< type_list, typename type_list::Head >	56
TL::Replace< T, ind, Arg, Args >	56
TL::Replace< T, 0, TypeList< Arg, Args... > >	57
TL::Replace< T, ind, TypeList< Arg, Args... > >	57
TL::Size< type_list >	58
TL::Size< EmptyTypeList >	59
true_type	
TL::IsTypeList< TypeList< Args... > >	47
TL::TypeAt< type_list, ind >	59
TL::TypeAt< type_list, 0 >	60
TypeList< Args >	60
TypeList< H, T... >	61
TypeList< T >	61
VertexStream< stream, graph >	62
VertexStream< EmptyTypeList, graph >	63

Chapter 3

Class Index

3.1 Class List

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

TL::Add< T, ind, Arg, Args >	13
TL::Add< T, 0, TypeList< Arg, Args... > >	13
TL::Add< T, 0, TypeList< Args... > >	14
TL::Add< T, ind, TypeList< Arg, Args... > >	14
GLib::AddEdge< GraphType, graph, edge >	15
GLib::AddEdge< ADJACENCY_LIST, graph, edge >	15
AdjacencyListGraph< vertexes, adjacency_list >	16
AdjacencyMatrixGraph< vertexes, matrix >	18
Objects::Boolean< boolean >	19
CheckContainsConstructibleParent< type_list, T, is_parent >	19
CheckContainsConstructibleParent< type_list, T, false >	19
CheckContainsConstructibleParent< type_list, T, true >	20
CheckContainsParent< type_list, T, is_parent >	20
CheckContainsParent< type_list, T, false >	20
CheckContainsParent< type_list, T, true >	21
CheckFindParentTypeList< contains_class, T, type_list, type_lists >	21
CheckFindParentTypeList< false, T, type_list, type_lists... >	21
CheckFindParentTypeList< true, T, type_list, type_lists... >	22
CheckFindTypeListByClass< contains_class, T, type_list, type_lists >	22
CheckFindTypeListByClass< false, T, type_list, type_lists... >	22
CheckFindTypeListByClass< true, T, type_list, type_lists... >	22
CheckHasDerivedAndConstructible< type_list, T, is_head_parent_of_T >	23
CheckHasDerivedAndConstructible< type_list, T, false >	23
CheckHasDerivedAndConstructible< type_list, T, true >	23
CheckIsBaseOf< has_parent, parent, derived >	23
CheckIsBaseOf< false, parent, derived >	24
CheckIsBaseOf< true, parent, derived >	24
CheckMostDerived< type_list, T, is_head_parent_of_T >	24
CheckMostDerived< type_list, T, false >	25
CheckMostDerived< type_list, T, true >	25
CheckMostDerivedAndConstructible< type_list, T, is_head_parent_of_T >	25
CheckMostDerivedAndConstructible< type_list, T, false >	25
CheckMostDerivedAndConstructible< type_list, T, true >	26
Class< T >	26
TL::Concatenate< front, back >	26

TL::Concatenate< front, EmptyTypeList >	27
TL::Contains< type_list, T >	27
TL::ContainsConstructibleParent< type_list, T >	28
TL::ContainsConstructibleParent< EmptyTypeList, T >	29
TL::ContainsParent< type_list, T >	29
TL::ContainsParent< EmptyTypeList, T >	30
ConvertGraph< From, To, graph >	30
ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >	31
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >	32
ConvertGraph< POINTER_STRUCTURE, EDGE_LIST, graph >	32
AdjacencyListGraph< vertexes, adjacency_list >::ConvertTo< GraphType >	33
EdgeListGraph< vertexes, edge_list >::ConvertTo< GraphType >	34
AdjacencyMatrixGraph< vertexes, matrix >::ConvertTo< GraphType >	34
EdgeListGraph< vertexes, edge_list >::ConvertTo< ADJACENCY_LIST >	35
AdjacencyMatrixGraph< vertexes, matrix >::ConvertTo< EDGE_LIST >	35
AdjacencyListGraph< vertexes, adjacency_list >::ConvertTo< POINTER_STRUCTURE >	36
PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >	36
Edge< from_, to_, weight_ >	37
EdgeListGraph< vertexes, edge_list >	37
TL::FindParentTypeList< T, type_list, type_lists >	38
TL::FindTypeListByClass< T, type_list, type_lists >	39
Functor< ResultType, ArgTypes >	40
Functor< ResultType(ArgTypes...) >	40
TL::GenerateTypeLists< n >	41
TL::GenerateTypeLists< 0 >	42
Graph	42
TL::HasDerivedAndConstructible< type_list, T >	42
TL::HasDerivedAndConstructible< EmptyTypeList, T >	43
TL::IndexOf< type_list, T >	44
TL::IndexOf< EmptyTypeList, T >	44
TL::IndexOf< type_list, typename type_list::Head >	45
Objects::Integer< integer >	45
TL::IsBaseOf< parent, derived >	45
TL::IsBaseOf< EmptyTypeList, derived >	46
TL::IsBaseOf< EmptyTypeList, EmptyTypeList >	46
TL::IsBaseOf< parent, EmptyTypeList >	47
TL::IsTypeList< T >	47
TL::IsTypeList< TypeList< Args... > >	47
PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >::IterateThroughChildren< current_children >	48
PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >::IterateThroughChildren< EmptyTypeList >	48
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< edge_list >	48
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< EmptyTypeList >	49
ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< current_vertexes, current_children >	49
ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< EmptyTypeList, EmptyTypeList >	49
TL::MostDerived< type_list, T >	49
TL::MostDerived< EmptyTypeList, T >	50
TL::MostDerivedAndConstructible< type_list, T >	50
TL::MostDerivedAndConstructible< EmptyTypeList, T >	51
PointerStructureGraph< vertexes >::Node< vertex_, children_ >	51
TL::NoDuplicates< type_list >	52
TL::NoDuplicates< EmptyTypeList >	52
NullType	53
PointerStructureGraph< vertexes >	53
TL::Remove< type_list, T >	54

TL::Remove< EmptyTypeList, T >	54
TL::Remove< type_list, typename type_list::Head >	55
TL::RemoveAll< type_list, T >	55
TL::RemoveAll< type_list, typename type_list::Head >	56
TL::Replace< T, ind, Arg, Args >	56
TL::Replace< T, 0, TypeList< Arg, Args... > >	57
TL::Replace< T, ind, TypeList< Arg, Args... > >	57
TL::Size< type_list >	58
TL::Size< EmptyTypeList >	59
TL::TypeAt< type_list, ind >	59
TL::TypeAt< type_list, 0 >	60
TypeList< Args >	60
TypeList< H, T... >	61
TypeList< T >	61
VertexStream< stream, graph >	62
VertexStream< EmptyTypeList, graph >	63

Chapter 4

Namespace Documentation

4.1 FunctorType Namespace Reference

Typedefs

- `template<typename ... InputArgs>`
using **Consumer** = [Functor](#)< void(InputArgs...)>

4.1.1 Detailed Description

Represents different types of functors.

See also

Package `java.util.function` in Java

4.2 Objects Namespace Reference

Classes

- struct [Boolean](#)
- struct [Integer](#)

4.2.1 Detailed Description

Represents class holders of different objects

4.3 TL Namespace Reference

Classes

- struct [Add](#)
- struct [Add< T, 0, TypeList< Arg, Args... > >](#)
- struct [Add< T, 0, TypeList< Args... > >](#)
- struct [Add< T, ind, TypeList< Arg, Args... > >](#)
- struct [Concatenate](#)
- struct [Concatenate< front, EmptyTypeList >](#)
- struct [Contains](#)
- struct [ContainsConstructibleParent](#)
- struct [ContainsConstructibleParent< EmptyTypeList, T >](#)
- struct [ContainsParent](#)
- struct [ContainsParent< EmptyTypeList, T >](#)
- struct [FindParentTypeList](#)
- struct [FindTypeListByClass](#)
- struct [GenerateTypeLists](#)
- struct [GenerateTypeLists< 0 >](#)
- struct [HasDerivedAndConstructible](#)
- struct [HasDerivedAndConstructible< EmptyTypeList, T >](#)
- struct [IndexOf](#)
- struct [IndexOf< EmptyTypeList, T >](#)
- struct [IndexOf< type_list, typename type_list::Head >](#)
- struct [IsBaseOf](#)
- struct [IsBaseOf< EmptyTypeList, derived >](#)
- struct [IsBaseOf< EmptyTypeList, EmptyTypeList >](#)
- struct [IsBaseOf< parent, EmptyTypeList >](#)
- struct [IsTypeList](#)
- struct [IsTypeList< TypeList< Args... > >](#)
- struct [MostDerived](#)
- struct [MostDerived< EmptyTypeList, T >](#)
- struct [MostDerivedAndConstructible](#)
- struct [MostDerivedAndConstructible< EmptyTypeList, T >](#)
- struct [NoDuplicates](#)
- struct [NoDuplicates< EmptyTypeList >](#)
- struct [Remove](#)
- struct [Remove< EmptyTypeList, T >](#)
- struct [Remove< type_list, typename type_list::Head >](#)
- struct [RemoveAll](#)
- struct [RemoveAll< type_list, typename type_list::Head >](#)
- struct [Replace](#)
- struct [Replace< T, 0, TypeList< Arg, Args... > >](#)
- struct [Replace< T, ind, TypeList< Arg, Args... > >](#)
- struct [Size](#)
- struct [Size< EmptyTypeList >](#)
- struct [TypeAt](#)
- struct [TypeAt< type_list, 0 >](#)

4.3.1 Detailed Description

Represents functions (as structs) for working with [TypeList](#)

Chapter 5

Class Documentation

5.1 TL::Add< T, ind, Arg, Args > Struct Template Reference

5.1.1 Detailed Description

```
template<typename T, size_t ind, class Arg, class ... Args>
struct TL::Add< T, ind, Arg, Args >
```

See also

[Add](#)<T, ind, TypeList<Arg, Args...>>

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

- TL/add.h

5.2 TL::Add< T, 0, TypeList< Arg, Args... > > Struct Template Reference

```
#include <add.h>
```

Public Types

- using **result** = [TypeList](#)< T, Arg, Args... >

5.2.1 Detailed Description

```
template<typename T, class Arg, class ... Args>
struct TL::Add< T, 0, TypeList< Arg, Args... > >
```

See also

[Add](#)<T, ind, TypeList<Arg, Args...>>

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

- TL/add.h

5.3 TL::Add< T, 0, TypeList< Args... > > Struct Template Reference

```
#include <add.h>
```

Public Types

- using **result** = [TypeList](#)< T, Args... >

5.3.1 Detailed Description

```
template<typename T, class ... Args>
struct TL::Add< T, 0, TypeList< Args... > >
```

See also

[Add](#)<T, ind, TypeList<Arg, Args...>>

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

- TL/add.h

5.4 TL::Add< T, ind, TypeList< Arg, Args... > > Struct Template Reference

```
#include <add.h>
```

Public Types

- using **end** = typename [Add](#)< T, ind - 1, [TypeList](#)< Args... > >::result
- using **result** = typename [Add](#)< Arg, 0, end >::result

5.4.1 Detailed Description

```
template<typename T, size_t ind, class Arg, class ... Args>
struct TL::Add< T, ind, TypeList< Arg, Args... > >
```

Adds typename to a specific position in [TypeList](#)

Parameters

<i>T</i>	Typename to add to a specific position in TypeList
<i>ind</i>	Number of this position
<i>TypeList<Arg,Args...></i>	This TypeList

Returns

Parameter result, new type list with typename added to position ind

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

- TL/add.h

5.5 GLib::AddEdge< GraphType, graph, edge > Struct Template Reference

5.5.1 Detailed Description

```
template<GraphType, class graph, class edge>
struct GLib::AddEdge< GraphType, graph, edge >
```

Returns new graph with added edge

Parameters

<i>GraphType</i>	Template parameter, type of a graph
<i>graph</i>	Template parameter, initial graph
<i>edge</i>	Template parameter, edge to add

See also

[Edge](#)

GraphType

Returns

Parameter result, new graph with added edge

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

- graph/add_edge.h

5.6 GLib::AddEdge< ADJACENCY_LIST, graph, edge > Struct Template Reference

```
#include <add_edge.h>
```

Public Types

- using **adjacent_vertexes** = typename [TL::TypeAt](#)< typename graph::adjacency_list_, vertex_num >::value
- using **new_adjacent_vertexes** = typename [TL::Add](#)< edge, 0, adjacent_vertexes >::result
- using **new_adjacency_list** = typename [TL::Replace](#)< new_adjacent_vertexes, vertex_num, typename graph::adjacency_list_ >::result
- using **result** = [AdjacencyListGraph](#)< typename graph::vertexes_, new_adjacency_list >

Static Public Attributes

- constexpr static size_t **vertex_num** = graph::template GetVertexIndex<typename edge::from>()

5.6.1 Detailed Description

```
template<class graph, class edge>
struct GLib::AddEdge< ADJACENCY_LIST, graph, edge >
```

See also

[AddEdge](#)

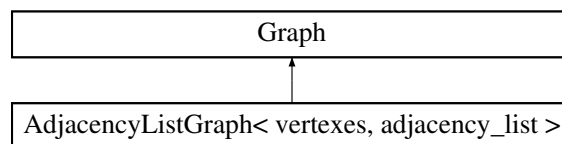
The documentation for this struct was generated from the following file:

- graph/add_edge.h

5.7 AdjacencyListGraph< vertexes, adjacency_list > Struct Template Reference

```
#include <adjacency_list_graph.h>
```

Inheritance diagram for AdjacencyListGraph< vertexes, adjacency_list >:



Classes

- struct [ConvertTo](#)
- struct [ConvertTo](#)< POINTER_STRUCTURE >

Public Types

- using **vertexes_** = vertexes
TypeList of vertexes in graph.
- using **adjacency_list_** = adjacency_list
TypeList of TypeLists of edges, which are grouped by starting vertex.

Public Member Functions

- template<class edge >
constexpr bool [HasEdge](#) ()

Static Public Member Functions

- template<typename vertex >
constexpr static size_t [GetVertexIndex](#) ()

5.7.1 Detailed Description

template<class vertexes, class adjacency_list>
struct AdjacencyListGraph< vertexes, adjacency_list >

Represents graph vertexes defined in vertexes_, and edges, which are derived from adjacency_list_ Size of an adjacency list must be equal to amount of vertexes

Parameters

<i>vertexes_</i>	TypeList of vertexes in graph.
<i>adjacency_↔ list_</i>	- TypeList of TypeLists of edges, which are grouped by starting vertex i.e. edge (from, to, weight) goes to adjacency_list_[from]

5.7.2 Member Function Documentation

5.7.2.1 GetVertexIndex()

```
template<class vertexes , class adjacency_list >
template<typename vertex >
constexpr static size_t AdjacencyListGraph< vertexes, adjacency_list >::GetVertexIndex ( )
[inline], [static], [constexpr]
```

Gets index of a passed vertex, throws assert if there is no such vertex

Parameters

<i>vertex</i>	Template parameter
---------------	--------------------

Returns

Position of this vertex in vertexes_ [TypeList](#)

5.7.2.2 HasEdge()

```
template<class vertexes , class adjacency_list >
template<class edge >
constexpr bool AdjacencyListGraph< vertexes, adjacency_list >::HasEdge ( ) [inline], [constexpr]
```

Checks if edge, passed as a template, is located in this graph

Parameters

<i>edge</i>	Template parameter, represents an edge to check
-------------	---

Returns

true if this edge in the graph, false otherwise

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

- graph/graphs/adjacency_list_graph.h

5.8 AdjacencyMatrixGraph< vertexes, matrix > Struct Template Reference

```
#include <adjacency_matrix_graph.h>
```

Classes

- struct [ConvertTo](#)
- struct [ConvertTo< EDGE_LIST >](#)

Public Types

- using [vertexes_](#) = vertexes
TypeList of vertexes in graph.
- using [matrix_](#) = matrix
TypeList of TypeLists of Edges.

5.8.1 Detailed Description

```
template<class vertexes, class matrix>
struct AdjacencyMatrixGraph< vertexes, matrix >
```

Represents graph as an adjacency matrix. If element in row i, column j is [NullType](#), then there's no edge between them. Otherwise it should be a corresponding [Edge](#).

See also

[Graph](#)
[Edge](#)

Parameters

<i>vertexes</i>	Template parameter, vertexes of a graph
<i>matrix</i>	Template parameter, an adjacency matrix

Returns

Parameter result, resulting graph

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

- graph/graphs/adjacency_matrix_graph.h

5.9 Objects::Boolean< boolean > Struct Template Reference

Static Public Attributes

- constexpr static bool **value** = boolean

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

- graph/objects.h

5.10 CheckContainsConstructibleParent< type_list, T, is_parent > Struct Template Reference

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

- TL/contains_constructible_parent.h

5.11 CheckContainsConstructibleParent< type_list, T, false > Struct Template Reference

Static Public Attributes

- constexpr static bool **result**

5.11.1 Member Data Documentation

5.11.1.1 result

```
template<class type_list , typename T >
constexpr static bool CheckContainsConstructibleParent< type_list, T, false >::result [static],
[constexpr]
```

Initial value:

```
= TL::ContainsConstructibleParent<
    typename type_list::Tail,
    T
>::result
```

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

- TL/contains_constructible_parent.h

5.12 CheckContainsConstructibleParent< type_list, T, true > Struct Template Reference

Static Public Attributes

- constexpr static bool **result** = true

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

- TL/contains_constructible_parent.h

5.13 CheckContainsParent< type_list, T, is_parent > Struct Template Reference

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

- TL/contains_parent.h

5.14 CheckContainsParent< type_list, T, false > Struct Template Reference

Static Public Attributes

- constexpr static bool **result**

5.14.1 Member Data Documentation

5.14.1.1 result

```
template<class type_list , typename T >
constexpr static bool CheckContainsParent< type_list, T, false >::result [static], [constexpr]
```

Initial value:

```
= TL::ContainsParent<
    typename type_list::Tail,
    T
>::result
```

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

- TL/contains_parent.h

5.15 CheckContainsParent< type_list, T, true > Struct Template Reference

Static Public Attributes

- constexpr static bool **result** = true

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

- TL/contains_parent.h

5.16 CheckFindParentTypeList< contains_class, T, type_list, type_lists > Struct Template Reference

Public Types

- using **result** = [NullType](#)

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

- TL/find_parent_type_list.h

5.17 CheckFindParentTypeList< false, T, type_list, type_lists... > Struct Template Reference

Public Types

- using **result** = typename [TL::FindParentTypeList](#)< T, type_lists... >::result

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

- TL/find_parent_type_list.h

5.18 CheckFindParentTypeList< true, T, type_list, type_lists... > Struct Template Reference

Public Types

- using **result** = type_list

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

- TL/find_parent_type_list.h

5.19 CheckFindTypeListByClass< contains_class, T, type_list, type_lists > Struct Template Reference

Public Types

- using **result** = [NullType](#)

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

- TL/find_type_list_by_class.h

5.20 CheckFindTypeListByClass< false, T, type_list, type_lists... > Struct Template Reference

Public Types

- using **result** = typename [TL::FindTypeListByClass](#)< T, type_lists... >::result

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

- TL/find_type_list_by_class.h

5.21 CheckFindTypeListByClass< true, T, type_list, type_lists... > Struct Template Reference

Public Types

- using **result** = type_list

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

- TL/find_type_list_by_class.h

5.22 CheckHasDerivedAndConstructible< type_list, T, is_head_parent_of_T > Struct Template Reference

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

- TL/has_derived_and_constructible.h

5.23 CheckHasDerivedAndConstructible< type_list, T, false > Struct Template Reference

Static Public Attributes

- constexpr static bool **result** = [TL::HasDerivedAndConstructible](#)<typename type_list::Tail, T>::result

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

- TL/has_derived_and_constructible.h

5.24 CheckHasDerivedAndConstructible< type_list, T, true > Struct Template Reference

Static Public Attributes

- constexpr static bool **result** = true

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

- TL/has_derived_and_constructible.h

5.25 CheckIsBaseOf< has_parent, parent, derived > Struct Template Reference

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

- TL/is_base_of.h

5.26 CheckIsBaseOf< false, parent, derived > Struct Template Reference

Static Public Attributes

- constexpr static bool **result** = false

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

- TL/is_base_of.h

5.27 CheckIsBaseOf< true, parent, derived > Struct Template Reference

Static Public Attributes

- constexpr static bool **result**

5.27.1 Member Data Documentation

5.27.1.1 result

```
template<class parent , class derived >
constexpr static bool CheckIsBaseOf< true, parent, derived >::result [static], [constexpr]
```

Initial value:

```
= TL::IsBaseOf<
    parent,
    typename derived::Tail
>::result
```

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

- TL/is_base_of.h

5.28 CheckMostDerived< type_list, T, is_head_parent_of_T > Struct Template Reference

Public Types

- using **result** = [NullType](#)

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

- TL/most_derived.h

5.29 CheckMostDerived< type_list, T, false > Struct Template Reference

Public Types

- using **result** = typename [TL::MostDerived](#)< typename type_list::Tail, T >::result

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

- TL/most_derived.h

5.30 CheckMostDerived< type_list, T, true > Struct Template Reference

Public Types

- using **result** = typename [TL::MostDerived](#)< typename type_list::Tail, typename type_list::Head >::result

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

- TL/most_derived.h

5.31 CheckMostDerivedAndConstructible< type_list, T, is_head_parent_of_T > Struct Template Reference

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

- TL/most_derived_and_constructible.h

5.32 CheckMostDerivedAndConstructible< type_list, T, false > Struct Template Reference

Public Types

- using **result** = typename [TL::MostDerivedAndConstructible](#)< typename type_list::Tail, T >::result

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

- TL/most_derived_and_constructible.h

5.33 CheckMostDerivedAndConstructible< type_list, T, true > Struct Template Reference

Public Types

- using **result** = typename [TL::MostDerivedAndConstructible](#)< typename type_list::Tail, typename type_list::Head >::result

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

- TL/most_derived_and_constructible.h

5.34 Class< T > Struct Template Reference

```
#include <class.h>
```

Public Types

- using [value](#) = T
Holder of a typename.

5.34.1 Detailed Description

```
template<typename T>
struct Class< T >
```

Represents holder for a typename

Parameters

<i>T</i>	Template parameter, typename that should be contained
----------	---

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

- class.h

5.35 TL::Concatenate< front, back > Struct Template Reference

```
#include <concatenate.h>
```

Public Types

- using **result** = [Concatenate](#)< typename [Add](#)< typename back::Head, [Size](#)< front >::size, front >::result, typename back::Tail >

5.35.1 Detailed Description

```
template<class front, class back>
struct TL::Concatenate< front, back >
```

Concatenates two TypeLists

Parameters

<i>front</i>	Template parameter
<i>back</i>	Template parameter

Returns

Parameter result, Concatenated [TypeList](#)

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

- TL/concatenate.h

5.36 TL::Concatenate< front, EmptyTypeList > Struct Template Reference

```
#include <concatenate.h>
```

Public Types

- using **result** = front

5.36.1 Detailed Description

```
template<class front>
struct TL::Concatenate< front, EmptyTypeList >
```

See also

[Concatenate](#)

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

- TL/concatenate.h

5.37 TL::Contains< type_list, T > Struct Template Reference

```
#include <contains.h>
```

Static Public Attributes

- constexpr static bool **result** = [IndexOf](#)<type_list, T>::value >= 0

5.37.1 Detailed Description

```
template<class type_list, typename T>
struct TL::Contains< type_list, T >
```

Checks if type_list contains typename T

Parameters

<i>type_list</i>	Template parameter
<i>T</i>	Template parameter

Returns

Parameter result, true if type_list contains typename T, false otherwise

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

- TL/contains.h

5.38 TL::ContainsConstructibleParent< type_list, T > Struct Template Reference

Static Public Attributes

- constexpr static bool **result**

5.38.1 Detailed Description

```
template<class type_list, typename T>
struct TL::ContainsConstructibleParent< type_list, T >
```

Checks if type_list contains constructible parent of T

Parameters

<i>type_list</i>	Template parameter
<i>T</i>	Template parameter

Returns

Parameter result, true if type_list contains constructible parent of T, false otherwise

5.38.2 Member Data Documentation**5.38.2.1 result**

```
template<class type_list , typename T >
constexpr static bool TL::ContainsConstructibleParent< type_list, T >::result [static], [constexpr]
```

Initial value:

```
= CheckContainsConstructibleParent<
    type_list,
    T,
    std::is_base_of<typename type_list::Head, T>::value &&
    std::is_constructible<typename type_list::Head>::value
>::result
```

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

- TL/contains_constructible_parent.h

5.39 TL::ContainsConstructibleParent< EmptyTypeList, T > Struct Template Reference**Static Public Attributes**

- constexpr static bool **result** = false

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

- TL/contains_constructible_parent.h

5.40 TL::ContainsParent< type_list, T > Struct Template Reference**Static Public Attributes**

- constexpr static bool **result**

5.40.1 Detailed Description

```
template<class type_list, typename T>
struct TL::ContainsParent< type_list, T >
```

Checks if type_list contains parent of T

Parameters

<i>type_list</i>	Template parameter
<i>T</i>	Template parameter

Returns

Parameter result, true if *type_list* contains parent of *T*, false otherwise

5.40.2 Member Data Documentation

5.40.2.1 result

```
template<class type_list , typename T >
constexpr static bool TL::ContainsParent< type_list, T >::result [static], [constexpr]
```

Initial value:

```
= CheckContainsParent<
    type_list,
    T,
    std::is_base_of<typename type_list::Head, T>::value
>::result
```

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

- TL/contains_parent.h

5.41 TL::ContainsParent< EmptyTypeList, T > Struct Template Reference

Static Public Attributes

- constexpr static bool **result** = false

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

- TL/contains_parent.h

5.42 ConvertGraph< From, To, graph > Struct Template Reference

```
#include <convert_graph.h>
```

5.42.1 Detailed Description

```
template<GraphType From, GraphType To, class graph>
struct ConvertGraph< From, To, graph >
```

An adapter to convert graph of type *From* to type *To*. Defined separate from its realizations in order to avoid cycle dependency. Is used as a visitor in Visitor pattern.

See also

GraphType

Parameters

<i>From</i>	Template parameter
<i>To</i>	Template parameter
<i>graph</i>	Template parameter

Returns

Parameter result, resulting graph.

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

- graph/graphs/convert_graph.h

5.43 ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph > Struct Template Reference

```
#include <convert_from_adjacency_list.h>
```

Classes

- struct [MakePointerStructureGraph](#)
- struct [MakePointerStructureGraph< EmptyTypeList, EmptyTypeList >](#)

Public Types

- using **result** = [PointerStructureGraph](#)< typename [MakePointerStructureGraph](#)< typename graph↔::vertexes_, typename graph::adjacency_list_ >::**result** >

5.43.1 Detailed Description

```
template<class graph>
struct ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >
```

See also

[ConvertGraph](#)

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

- graph/graphs/convert_from_adjacency_list.h

5.44 ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph > Struct Template Reference

```
#include <convert_from_edge_list.h>
```

Classes

- struct [IterateThroughEdges](#)
- struct [IterateThroughEdges< EmptyTypeList >](#)

Public Types

- using **result** = typename IterateThroughEdges< typename graph::edge_list_ >::result

5.44.1 Detailed Description

```
template<class graph>  
struct ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >
```

See also

[ConvertGraph](#)

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

- graph/graphs/convert_from_edge_list.h

5.45 ConvertGraph< POINTER_STRUCTURE, EDGE_LIST, graph > Struct Template Reference

```
#include <convert_from_pointer_structure.h>
```

5.45.1 Detailed Description

```
template<class graph>  
struct ConvertGraph< POINTER_STRUCTURE, EDGE_LIST, graph >
```

See also

[ConvertGraph](#)

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

- graph/graphs/convert_from_pointer_structure.h

5.46 AdjacencyListGraph< vertexes, adjacency_list >::ConvertTo< GraphType > Struct Template Reference

5.46.1 Detailed Description

```
template<class vertexes, class adjacency_list>
template<GraphType>
struct AdjacencyListGraph< vertexes, adjacency_list >::ConvertTo< GraphType >
```

Represents an adapter, which converts one type of a graph into another. Is used as an element in Visitor pattern.

Parameters

<i>GraphType</i>	Template parameter, type of a resulting graph
------------------	---

Returns

Parameter result, resulting graph

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

- graph/graphs/adjacency_list_graph.h

5.47 **EdgeListGraph< vertexes, edge_list >::ConvertTo< GraphType > Struct Template Reference**

5.47.1 Detailed Description

```
template<class vertexes, class edge_list>
template<GraphType>
struct EdgeListGraph< vertexes, edge_list >::ConvertTo< GraphType >
```

Represents an adapter, which converts one type of a graph into another. Is used as an element in Visitor pattern.

Parameters

<i>GraphType</i>	Template parameter, type of a resulting graph
------------------	---

Returns

Parameter result, resulting graph

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

- graph/graphs/edge_list_graph.h

5.48 **AdjacencyMatrixGraph< vertexes, matrix >::ConvertTo< GraphType > Struct Template Reference**

5.48.1 Detailed Description

```
template<class vertexes, class matrix>
template<GraphType>
struct AdjacencyMatrixGraph< vertexes, matrix >::ConvertTo< GraphType >
```

Represents an adapter, which converts one type of a graph into another. Is used as an element in Visitor pattern.

Parameters

<i>GraphType</i>	Template parameter, type of a resulting graph
------------------	---

Returns

Parameter result, resulting graph

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

- graph/graphs/adjacency_matrix_graph.h

5.49 EdgeListGraph< vertexes, edge_list >::ConvertTo< ADJACENCY_LIST > Struct Reference

```
#include <edge_list_graph.h>
```

Public Types

- using **result** = typename [ConvertGraph](#)< EDGE_LIST, ADJACENCY_LIST, [EdgeListGraph](#)< vertexes_, edge_list_ > >::result

5.49.1 Detailed Description

```
template<class vertexes, class edge_list>
struct EdgeListGraph< vertexes, edge_list >::ConvertTo< ADJACENCY_LIST >
```

See also

[ConvertTo](#)

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

- graph/graphs/edge_list_graph.h

5.50 AdjacencyMatrixGraph< vertexes, matrix >::ConvertTo< EDGE_LIST > Struct Reference

Public Types

- using **result** = typename [ConvertGraph](#)< ADJACENCY_MATRIX, EDGE_LIST, [AdjacencyMatrixGraph](#)< vertexes_, matrix_ > >::result

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

- graph/graphs/adjacency_matrix_graph.h

5.51 AdjacencyListGraph< vertexes, adjacency_list >::ConvertTo< POINTER_STRUCTURE > Struct Reference

Public Types

- using **result** = typename [ConvertGraph](#)< ADJACENCY_LIST, POINTER_STRUCTURE, [AdjacencyListGraph](#)< vertexes, adjacency_list > >::result

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

- graph/graphs/adjacency_list_graph.h

5.52 PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes > Struct Template Reference

```
#include <pointer_structure_graph.h>
```

Classes

- struct [IterateThroughChildren](#)
- struct [IterateThroughChildren](#)< [EmptyTypeList](#) >

Public Types

- using **vertexes_to_visit** = typename [TL::Remove](#)< unvisited_vertexes, current_vertex >::result
- using **result** = typename [IterateThroughChildren](#)< typename current_vertex::children >::result

5.52.1 Detailed Description

```
template<class vertexes>
template<class current_vertex, class unvisited_vertexes = vertexes_>
struct PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >
```

Performs Depth-First Search, starting from passed vertex. It doesn't visit vertexes that have been visited already. It returns visited edges in chronological order, from which it's easy to deduce [DFS](#). It's more versatile than one may think) Also a variation of Composite pattern.

Parameters

<i>starting_vertex</i>	Template parameter, starting vertex of DFS .
------------------------	--

Returns

Parameter result, [TypeList](#) of visited edges in chronological order.

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

- graph/graphs/pointer_structure_graph.h

5.53 Edge< from_, to_, weight_ > Struct Template Reference

```
#include <edge.h>
```

Public Types

- using [from](#) = from_
Starting vertex of an edge.
- using [to](#) = to_
Ending vertex of an edge.
- using [weight](#) = weight_
Additional property of an edge.

5.53.1 Detailed Description

```
template<typename from_, typename to_, typename weight_ = NullType>
struct Edge< from_, to_, weight_ >
```

Represents an edge in the graph.

Parameters

<i>from</i> _↔ —	Template parameter, starting vertex of an edge
<i>to</i> _—	Template parameter, ending vertex of an edge
<i>weight</i> _↔ —	Template parameter, additional property of an edge

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

- graph/edge.h

5.54 EdgeListGraph< vertexes, edge_list > Struct Template Reference

```
#include <edge_list_graph.h>
```

Classes

- struct [ConvertTo](#)
- struct [ConvertTo](#)< [ADJACENCY_LIST](#) >

Public Types

- using [vertexes_](#) = vertexes
TypeList of vertexes in graph.
- using [edge_list_](#) = edge_list
TypeList of edges.

5.54.1 Detailed Description

```
template<class vertexes, class edge_list>
struct EdgeListGraph< vertexes, edge_list >
```

Represents graph as a list of edges.

See also

[Graph](#)
[Edge](#)

Parameters

<i>vertexes</i>	Template parameter, vertexes of a graph
<i>edge_list</i>	Template parameter, TypeList of Edge

Returns

Parameter result, resulting graph

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

- graph/graphs/edge_list_graph.h

5.55 TL::FindParentTypeList< T, type_list, type_lists > Struct Template Reference

Public Types

- using **result** = typename [CheckFindParentTypeList](#)< [TL::IsBaseOf](#)< type_list, T >::result, T, type_list, type_lists... >::result

5.55.1 Detailed Description

```
template<typename T, class type_list, class ... type_lists>
struct TL::FindParentTypeList< T, type_list, type_lists >
```

Finds and returns [TypeList](#) that has the parent of T

Parameters

<i>T</i>	
<i>type_list</i>	First TypeList among other TypeLists
<i>...type_lists</i>	Other TypeLists to check

Returns

Parameter result, first [TypeList](#) that contains the parent of T, compilation error otherwise

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

- TL/find_parent_type_list.h

5.56 TL::FindTypeListByClass< T, type_list, type_lists > Struct Template Reference

Public Types

- using **result** = typename [CheckFindTypeListByClass](#)< [TL::Contains](#)< type_list, T >::result, T, type_list, type_lists... >::result

5.56.1 Detailed Description

```
template<typename T, class type_list, class ... type_lists>
struct TL::FindTypeListByClass< T, type_list, type_lists >
```

Finds and returns [TypeList](#) that has T

Parameters

<i>T</i>	Template parameter
<i>type_list</i>	Template parameter, first TypeList among other TypeLists
<i>...type_lists</i>	Template parameter, other TypeLists to check

Returns

Parameter result, first [TypeList](#) that contains T, compilation error otherwise

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

- TL/find_type_list_by_class.h

5.57 Functor< ResultType, ArgTypes > Class Template Reference

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

- functor.h

5.58 Functor< ResultType(ArgTypes...)> Class Template Reference

```
#include <functor.h>
```

Public Member Functions

- `template<typename Function >`
Functor (Function function)
- `template<typename Function , class Class >`
Functor (Function Class::*function)
- **Functor** (const [Functor](#) &other)
- [Functor](#) & **operator=** (const [Functor](#) &other)
- `ResultType operator\(\) (ArgTypes... args)`

5.58.1 Detailed Description

```
template<typename ResultType, typename ... ArgTypes>
class Functor< ResultType(ArgTypes...)>
```

Provides an object that contains a function

Parameters

<i>ResultType</i>	Template parameter, type of an object function returns
<i>ArgTypes</i>	Template parameters, types of an object function accepts

5.58.2 Member Function Documentation

5.58.2.1 operator>()

```
template<typename ResultType , typename ... ArgTypes>
ResultType Functor< ResultType(ArgTypes...)>::operator() (
    ArgTypes... args ) [inline]
```

Invokes function

Parameters

<i>args</i>	Arguments for a function
-------------	--------------------------

Returns

Result of a function with passed args as arguments

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

- functor.h

5.59 TL::GenerateTypeLists< n > Struct Template Reference

```
#include <generate_type_lists.h>
```

Public Types

- using **result** = typename [Add](#)< [EmptyTypeList](#), 0, typename [GenerateTypeLists](#)< n - 1 >::result >::result

5.59.1 Detailed Description

```
template<int n>
struct TL::GenerateTypeLists< n >
```

Generates [TypeList](#) of n [EmptyTypeLists](#)

See also

[EmptyTypeList](#)

Parameters

<i>n</i>	Template parameter, a number of EmptyTypeLists to generate
----------	--

Returns

Parameter result, [TypeList](#) of n [EmptyTypeList](#)

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

- TL/generate_type_lists.h

5.60 TL::GenerateTypeLists< 0 > Struct Reference

Public Types

- using **result** = [EmptyTypeList](#)

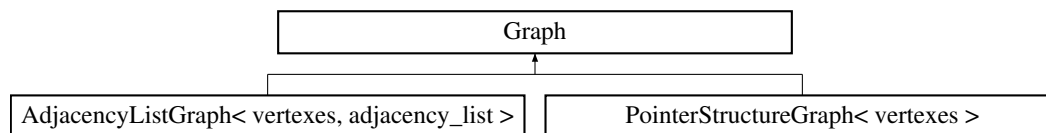
The documentation for this struct was generated from the following file:

- TL/generate_type_lists.h

5.61 Graph Struct Reference

```
#include <graph.h>
```

Inheritance diagram for Graph:



5.61.1 Detailed Description

Represents placeholder for a graph. [Graph](#) is a pair of vertexes (collection of some elements), and edges (collection of pairs of vertexes). [Graph](#) can be represented in multiple ways in code. This library provides several preexisting implementations. Also it should be noted that preexisting implementations are built in compile-time, and it's recommended to follow this rule.

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

- graph/graphs/graph.h

5.62 TL::HasDerivedAndConstructible< type_list, T > Struct Template Reference

Static Public Attributes

- constexpr static bool **result**

5.62.1 Detailed Description

```
template<class type_list, typename T>
struct TL::HasDerivedAndConstructible< type_list, T >
```

Checks if type_list contains derived and constructible child of T

Parameters

<i>type_list</i>	Template parameter
<i>T</i>	Template parameter

Returns

Parameter result, true if *type_list* contains derived and constructible child of *T*, false otherwise

5.62.2 Member Data Documentation

5.62.2.1 result

```
template<class type_list , typename T >
constexpr static bool TL::HasDerivedAndConstructible< type_list, T >::result [static], [constexpr]
```

Initial value:

```
= CheckHasDerivedAndConstructible<
    type_list,
    T,
    std::is_base_of<T, typename type_list::Head>::value&&
    std::is_constructible<typename type_list::Head>::value
>::result
```

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

- TL/has_derived_and_constructible.h

5.63 TL::HasDerivedAndConstructible< EmptyTypeList, T > Struct Template Reference

```
#include <has_derived_and_constructible.h>
```

Static Public Attributes

- constexpr static bool **result** = false

5.63.1 Detailed Description

```
template<typename T>
struct TL::HasDerivedAndConstructible< EmptyTypeList, T >
```

See also

[HasDerivedAndConstructible](#)

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

- TL/has_derived_and_constructible.h

5.64 TL::IndexOf< type_list, T > Struct Template Reference

```
#include <index_of.h>
```

Static Public Attributes

- constexpr static int **value** = 1 + [IndexOf](#)<typename type_list::Tail, T>::value

5.64.1 Detailed Description

```
template<class type_list, typename T>
struct TL::IndexOf< type_list, T >
```

Gets index of a first occurrence of typename T in type_list

Parameters

<i>type_list</i>	Template parameter
<i>T</i>	Template parameter

Returns

Parameter value, index of a first occurrence of typename T in type_list, INT32_MIN otherwise

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

- TL/index_of.h

5.65 TL::IndexOf< EmptyTypeList, T > Struct Template Reference

```
#include <index_of.h>
```

Static Public Attributes

- constexpr static int **value** = INT32_MIN

5.65.1 Detailed Description

```
template<typename T>
struct TL::IndexOf< EmptyTypeList, T >
```

See also

[IndexOf](#)

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

- TL/index_of.h

5.66 TL::IndexOf< type_list, typename type_list::Head > Struct Template Reference

```
#include <index_of.h>
```

Static Public Attributes

- constexpr static int **value** = 0

5.66.1 Detailed Description

```
template<class type_list>
struct TL::IndexOf< type_list, typename type_list::Head >
```

See also

[IndexOf](#)

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

- TL/index_of.h

5.67 Objects::Integer< integer > Struct Template Reference

Static Public Attributes

- constexpr static int **value** = integer

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

- graph/objects.h

5.68 TL::IsBaseOf< parent, derived > Struct Template Reference

Static Public Attributes

- constexpr static bool **result**

5.68.1 Detailed Description

```
template<class parent, class derived>
struct TL::IsBaseOf< parent, derived >
```

Checks if [TypeList](#) "parent" is in fact parent of another [TypeList](#) "derived" "parent" is parent of "derived" if and only if for every class C in "derived", "parent" has parent of C

Parameters

<i>parent</i>	Template parameter
<i>derived</i>	Template parameter

Returns

true if [TypeList](#) "parent" is in fact parent of another [TypeList](#) "derived", false otherwise

5.68.2 Member Data Documentation

5.68.2.1 result

```
template<class parent , class derived >
constexpr static bool TL::IsBaseOf< parent, derived >::result [static], [constexpr]
```

Initial value:

```
= CheckIsBaseOf<
    ContainsParent<parent, typename derived::Head::result,
    parent,
    derived
>::result
```

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

- [TL/is_base_of.h](#)

5.69 [TL::IsBaseOf](#)< [EmptyTypeList](#), derived > Struct Template Reference

Static Public Attributes

- constexpr static bool **result** = false

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

- [TL/is_base_of.h](#)

5.70 [TL::IsBaseOf](#)< [EmptyTypeList](#), [EmptyTypeList](#) > Struct Reference

Static Public Attributes

- constexpr static bool **result** = true

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

- [TL/is_base_of.h](#)

5.71 TL::IsBaseOf< parent, EmptyTypeList > Struct Template Reference

Static Public Attributes

- constexpr static bool **result** = true

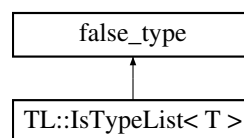
The documentation for this struct was generated from the following file:

- TL/is_base_of.h

5.72 TL::IsTypeList< T > Struct Template Reference

```
#include <is_type_list.h>
```

Inheritance diagram for TL::IsTypeList< T >:



5.72.1 Detailed Description

```
template<class T>
struct TL::IsTypeList< T >
```

Checks if passed class T is a [TypeList](#)

Parameters

<i>T</i>	Template argument
----------	-------------------

Returns

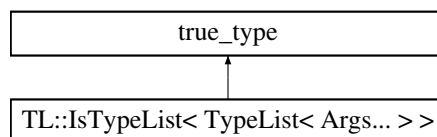
Parameter value, true if T is a [TypeList](#), false otherwise

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

- TL/is_type_list.h

5.73 TL::IsTypeList< TypeList< Args... > > Struct Template Reference

Inheritance diagram for TL::IsTypeList< TypeList< Args... > >:



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

- TL/is_type_list.h

5.74 PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >::IterateThroughChildren< current_children > Struct Template Reference

Public Types

- using **result** = std::conditional_t< [TL::Contains](#)< vertexes_to_visit, typename current_children::Head >::value, [TL::Concatenate](#)< typename [DFS](#)< typename current_children::Head, vertexes_to_visit >::result, typename [IterateThroughChildren](#)< typename current_children::Tail >::result >, typename [IterateThroughChildren](#)< typename current_children::Tail >::result >

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

- graph/graphs/pointer_structure_graph.h

5.75 PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >::IterateThroughChildren< EmptyTypeList > Struct Reference

Public Types

- using **result** = [EmptyTypeList](#)

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

- graph/graphs/pointer_structure_graph.h

5.76 ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< edge_list > Struct Template Reference

Public Types

- using **result** = typename [GLib::AddEdge](#)< ADJACENCY_LIST, typename [IterateThroughEdges](#)< typename edge_list::Tail >::result, typename edge_list::Head >::result

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

- graph/graphs/convert_from_edge_list.h

5.77 ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< EmptyTypeList > Struct Reference

Public Types

- using **result** = AdjacencyListGraph< typename graph::vertexes_, typename TL::GenerateTypeLists< TL::Size< typename graph::vertexes_ >::size >::result >

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

- graph/graphs/convert_from_edge_list.h

5.78 ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< current_vertexes, current_adjacency_list > Struct Template Reference

Public Types

- using **type_list_without_first** = typename MakePointerStructureGraph< typename current_vertexes::Tail, typename current_adjacency_list::Tail >::result
- using **result** = typename TL::Add< PointerStructureGraph::Node< typename current_vertexes::Head, typename current_adjacency_list::Head >, 0, type_list_without_first >::result

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

- graph/graphs/convert_from_adjacency_list.h

5.79 ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< EmptyTypeList, EmptyTypeList > Struct Reference

Public Types

- using **result** = EmptyTypeList

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

- graph/graphs/convert_from_adjacency_list.h

5.80 TL::MostDerived< type_list, T > Struct Template Reference

Public Types

- using **result** = typename CheckMostDerived< type_list, T, std::is_base_of< T, typename type_list::Head >::value >::result

5.80.1 Detailed Description

```
template<class type_list, typename T>
struct TL::MostDerived< type_list, T >
```

Finds the most derived child of T in type_list

Parameters

<i>type_list</i>	Template parameter
<i>T</i>	Template parameter

Returns

Parameter result, the most derived child of T in type_list

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

- TL/most_derived.h

5.81 TL::MostDerived< EmptyTypeList, T > Struct Template Reference

Public Types

- using **result** = T

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

- TL/most_derived.h

5.82 TL::MostDerivedAndConstructible< type_list, T > Struct Template Reference

Public Types

- using **result** = typename [CheckMostDerivedAndConstructible](#)< type_list, T, std::is_base_of< T, typename type_list::Head >::value &&std::is_constructible< typename type_list::Head >::value >::result

5.82.1 Detailed Description

```
template<class type_list, typename T>
struct TL::MostDerivedAndConstructible< type_list, T >
```

Finds the most derived and constructible child of T in type_list

Parameters

<i>type_list</i>	Template parameter
<i>T</i>	Template parameter

Returns

Parameter result, the most derived and constructible child of T in type_list

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

- TL/most_derived_and_constructible.h

5.83 TL::MostDerivedAndConstructible< EmptyTypeList, T > Struct Template Reference

Public Types

- using **result** = T

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

- TL/most_derived_and_constructible.h

5.84 PointerStructureGraph< vertexes >::Node< vertex_, children_ > Struct Template Reference

```
#include <pointer_structure_graph.h>
```

Public Types

- using **vertex** = vertex_
- using **children** = children_

5.84.1 Detailed Description

```
template<class vertexes>
template<class vertex_, class children_>
struct PointerStructureGraph< vertexes >::Node< vertex_, children_ >
```

Default version of a suitable class. It's not necessary to use this one. In fact, it's encouraged to make your objects suitable to [PointerStructureGraph](#).

Parameters

<i>vertex</i> ↔ —	Template parameter, vertex that this node represents
<i>children</i> ↔ —	Template parameter, TypeList of Edges, showing who can be reached from this vertex.

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

- graph/graphs/pointer_structure_graph.h

5.85 TL::NoDuplicates< type_list > Struct Template Reference

```
#include <no_duplicates.h>
```

Public Types

- using **result** = [TypeList](#)< typename type_list::Head, typename [NoDuplicates](#)< typename [RemoveAll](#)< type-name type_list::Tail, typename type_list::Head >::result >::result >

5.85.1 Detailed Description

```
template<class type_list>
struct TL::NoDuplicates< type_list >
```

Removes duplicated from [TypeList](#) type_list

Parameters

<i>type_list</i>	Template parameter
------------------	--------------------

Returns

Parameter result, new [TypeList](#) without any duplicates

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

- TL/no_duplicates.h

5.86 TL::NoDuplicates< EmptyTypeList > Struct Reference

```
#include <no_duplicates.h>
```

Public Types

- using **result** = [EmptyTypeList](#)

5.86.1 Detailed Description

See also

[NoDuplicates](#)

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

- `TL/no_duplicates.h`

5.87 NullType Struct Reference

```
#include <null_type.h>
```

5.87.1 Detailed Description

Represents nothing. If there is an absence of some template, it should be represented by [NullType](#).

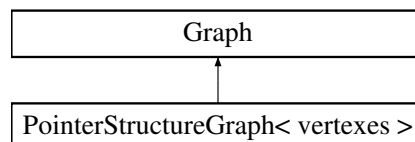
The documentation for this struct was generated from the following file:

- `TL/null_type.h`

5.88 PointerStructureGraph< vertexes > Struct Template Reference

```
#include <pointer_structure_graph.h>
```

Inheritance diagram for PointerStructureGraph< vertexes >:



Classes

- struct [DFS](#)
- struct [Node](#)

Public Types

- using [vertexes_](#) = vertexes
All accounted vertexes in this graph.

5.88.1 Detailed Description

```
template<class vertexes>
struct PointerStructureGraph< vertexes >
```

Represents graph as a structure with pointers Every vertex must contain [TypeList](#) "children", which is a [TypeList](#) of Edges, showing who can be reached from this vertex.

Parameters

<i>vertexes</i>	Template parameter, vertexes in this graph
-----------------	--

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

- graph/graphs/pointer_structure_graph.h

5.89 TL::Remove< type_list, T > Struct Template Reference

```
#include <remove.h>
```

Public Types

- using **result** = [TypeList](#)< typename type_list::Head, typename [Remove](#)< typename type_list::Tail, T >↔
::result >

5.89.1 Detailed Description

```
template<class type_list, typename T>
struct TL::Remove< type_list, T >
```

Removes first occurrence of T in type_list

Parameters

<i>type_list</i>	Template parameter
<i>T</i>	Template parameter

Returns

Parameter result, new [TypeList](#) without first occurrence of T

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

- TL/remove.h

5.90 TL::Remove< EmptyTypeList, T > Struct Template Reference

```
#include <remove.h>
```

Public Types

- using **result** = [EmptyTypeList](#)

5.90.1 Detailed Description

```
template<typename T>
struct TL::Remove< EmptyTypeList, T >
```

See also

[Remove](#)

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

- TL/remove.h

5.91 TL::Remove< type_list, typename type_list::Head > Struct Template Reference

```
#include <remove.h>
```

Public Types

- using **result** = typename type_list::Tail

5.91.1 Detailed Description

```
template<class type_list>
struct TL::Remove< type_list, typename type_list::Head >
```

See also

[Remove](#)

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

- TL/remove.h

5.92 TL::RemoveAll< type_list, T > Struct Template Reference

```
#include <remove.h>
```

Public Types

- using **result** = [TypeList](#)< typename type_list::Head, typename [RemoveAll](#)< typename type_list::Tail, T >↔
::result >

5.92.1 Detailed Description

```
template<class type_list, class T>
struct TL::RemoveAll< type_list, T >
```

Removes all occurrences of T in type_list

Parameters

<i>type_list</i>	Template parameter
<i>T</i>	Template parameter

Returns

Parameter result, new [TypeList](#) without occurrences of *T*

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

- [TL/remove.h](#)

5.93 TL::RemoveAll< type_list, typename type_list::Head > Struct Template Reference

```
#include <remove.h>
```

Public Types

- using **result** = typename [RemoveAll](#)< typename type_list::Tail, typename type_list::Head >::result

5.93.1 Detailed Description

```
template<class type_list>
struct TL::RemoveAll< type_list, typename type_list::Head >
```

See also

[RemoveAll](#)

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

- [TL/remove.h](#)

5.94 TL::Replace< T, ind, Arg, Args > Struct Template Reference

```
#include <replace.h>
```

5.94.1 Detailed Description

```
template<typename T, size_t ind, class Arg, class ... Args>
struct TL::Replace< T, ind, Arg, Args >
```

Replaces typename on a specific position in [TypeList](#)

Parameters

<i>T</i>	Typename that will be on a specific position in TypeList
<i>ind</i>	Number of this position
<i>TypeList<Arg,Args...></i>	This TypeList

Returns

Parameter result, new type list with typename added to position ind

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

- TL/replace.h

5.95 TL::Replace< T, 0, TypeList< Arg, Args... > > Struct Template Reference

```
#include <replace.h>
```

Public Types

- using **result** = [TypeList](#)< T, Args... >

5.95.1 Detailed Description

```
template<typename T, class Arg, class ... Args>
struct TL::Replace< T, 0, TypeList< Arg, Args... > >
```

See also

[Replace](#)

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

- TL/replace.h

5.96 TL::Replace< T, ind, TypeList< Arg, Args... > > Struct Template Reference

```
#include <replace.h>
```

Public Types

- using **end** = typename [Replace](#)< T, ind - 1, [TypeList](#)< Args... > >::result
- using **result** = typename [Add](#)< Arg, 0, end >::result

5.96.1 Detailed Description

```
template<typename T, size_t ind, class Arg, class ... Args>
struct TL::Replace< T, ind, TypeList< Arg, Args... > >
```

See also

[Replace](#)

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

- TL/replace.h

5.97 TL::Size< type_list > Struct Template Reference

```
#include <size.h>
```

Static Public Attributes

- constexpr static size_t **size** = 1 + [Size](#)<typename type_list::Tail>::size

5.97.1 Detailed Description

```
template<class type_list>
struct TL::Size< type_list >
```

Gets length of a [TypeList](#)

Parameters

TypeList	Template parameter
--------------------------	--------------------

Returns

Parameter size, amount of elements in [TypeList](#)

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

- TL/size.h

5.98 TL::Size< EmptyTypeList > Struct Reference

```
#include <size.h>
```

Static Public Attributes

- constexpr static size_t **size** = 0

5.98.1 Detailed Description

See also

[Size](#)

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

- TL/size.h

5.99 TL::TypeAt< type_list, ind > Struct Template Reference

```
#include <type_at.h>
```

Public Types

- using **value** = typename [TypeAt](#)< typename type_list::Tail, ind - 1 >::value

5.99.1 Detailed Description

```
template<class type_list, size_t ind>
struct TL::TypeAt< type_list, ind >
```

Get class at specific index of [TypeList](#)

Parameters

<i>type_list</i>	Template parameter, where required class is located
<i>ind</i>	Template parameter, shows position where required class is located

Returns

Parameter value, class at a specific index of [TypeList](#)

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

- TL/type_at.h

5.100 TL::TypeAt< type_list, 0 > Struct Template Reference

```
#include <type_at.h>
```

Public Types

- using **value** = typename type_list::Head

5.100.1 Detailed Description

```
template<class type_list>  
struct TL::TypeAt< type_list, 0 >
```

See also

[TypeAt](#)

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

- TL/type_at.h

5.101 TypeList< Args > Struct Template Reference

```
#include <type_list.h>
```

Public Types

- using **Head** = [NullType](#)
- using **Tail** = [TypeList<>](#)

5.101.1 Detailed Description

```
template<typename ... Args>  
struct TypeList< Args >
```

See also

[TypeList<H, T...>](#)

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

- TL/type_list.h

5.102 `TypeList< H, T... >` Struct Template Reference

```
#include <type_list.h>
```

Public Types

- using `Head` = `H`
First type in a type list.
- using `Tail` = `TypeList< T... >`
`TypeList` of other types.

5.102.1 Detailed Description

```
template<typename H, typename ... T>
struct TypeList< H, T... >
```

Represents a list of various types

Parameters

<i>H</i>	Template parameter, first object in a type list
<i>T</i>	Template parameter, other objects in a type list

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

- `TL/type_list.h`

5.103 `TypeList< T >` Struct Template Reference

```
#include <type_list.h>
```

Public Types

- using `Head` = `T`
- using `Tail` = `EmptyTypeList`

5.103.1 Detailed Description

```
template<typename T>
struct TypeList< T >
```

See also

[TypeList<H, T...>](#)

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

- `TL/type_list.h`

5.104 VertexStream< stream, graph > Struct Template Reference

```
#include <vertex_stream.h>
```

Public Types

- using [stream_](#) = stream
TypeList of vertexes of a graph.

Public Member Functions

- template<class Consumer >
void [ForEach](#) (Consumer consumer)

5.104.1 Detailed Description

```
template<class stream, class graph>
struct VertexStream< stream, graph >
```

Represents a stream of vertexes of a graph

Parameters

<i>stream</i>	Template parameter, TypeList of vertexes of a graph
<i>graph</i>	Template parameter

5.104.2 Member Function Documentation

5.104.2.1 ForEach()

```
template<class stream , class graph >
template<class Consumer >
void VertexStream< stream, graph >::ForEach (
    Consumer consumer ) [inline]
```

Calls consumer on every vertex in a stream It's recommended that this object must be of type FunctorTypes::↔ Consumer Also it's based on a variation of "Chain of a responsibility" pattern

Parameters

<i>consumer</i>	Consumer, that accepts Class object of a graph and index of a current vertex
-----------------	--

See also

FunctorType::Consumer

[Class](#)

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

- graph/vertex_stream.h

5.105 VertexStream< EmptyTypeList, graph > Struct Template Reference

```
#include <vertex_stream.h>
```

Public Types

- using **stream** = [EmptyTypeList](#)

Public Member Functions

- template<class Consumer >
void **ForEach** (Consumer consumer)

5.105.1 Detailed Description

```
template<class graph>  
struct VertexStream< EmptyTypeList, graph >
```

See also

[VertexStream](#)

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

- graph/vertex_stream.h

Index

AdjacencyListGraph< vertexes, adjacency_list >, 16
 GetVertexIndex, 17
 HasEdge, 17
AdjacencyListGraph< vertexes, adjacency_list >::ConvertTo< GraphType >, 33
AdjacencyListGraph< vertexes, adjacency_list >::ConvertTo< POINTER_STRUCTURE >, 36
AdjacencyMatrixGraph< vertexes, matrix >, 18
AdjacencyMatrixGraph< vertexes, matrix >::ConvertTo< EDGE_LIST >, 35
AdjacencyMatrixGraph< vertexes, matrix >::ConvertTo< GraphType >, 34

CheckContainsConstructibleParent< type_list, T, false >, 19
 result, 19
CheckContainsConstructibleParent< type_list, T, is_parent >, 19
CheckContainsConstructibleParent< type_list, T, true >, 20
CheckContainsParent< type_list, T, false >, 20
 result, 20
CheckContainsParent< type_list, T, is_parent >, 20
CheckContainsParent< type_list, T, true >, 21
CheckFindParentTypeList< contains_class, T, type_list, type_lists >, 21
CheckFindParentTypeList< false, T, type_list, type_lists... >, 21
CheckFindParentTypeList< true, T, type_list, type_lists... >, 22
CheckFindTypeListByClass< contains_class, T, type_list, type_lists >, 22
CheckFindTypeListByClass< false, T, type_list, type_lists... >, 22
CheckFindTypeListByClass< true, T, type_list, type_lists... >, 22
CheckHasDerivedAndConstructible< type_list, T, false >, 23
CheckHasDerivedAndConstructible< type_list, T, is_head_parent_of_T >, 23
CheckHasDerivedAndConstructible< type_list, T, true >, 23
CheckIsBaseOf< false, parent, derived >, 24
CheckIsBaseOf< has_parent, parent, derived >, 23
CheckIsBaseOf< true, parent, derived >, 24
 result, 24
CheckMostDerived< type_list, T, false >, 25
CheckMostDerived< type_list, T, is_head_parent_of_T >, 24
CheckMostDerived< type_list, T, true >, 25
CheckMostDerivedAndConstructible< type_list, T, false >, 25
CheckMostDerivedAndConstructible< type_list, T, is_head_parent_of_T >, 25
CheckMostDerivedAndConstructible< type_list, T, true >, 25
Class< T >, 26
ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >, 31
ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< current_vertexes, current_adjacency_list >, 49
ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< EmptyTypeList, EmptyTypeList >, 49
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >, 32
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< edge_list >, 48
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< EmptyTypeList >, 49
ConvertGraph< From, To, graph >, 30
ConvertGraph< POINTER_STRUCTURE, EDGE_LIST, graph >, 32

Edge< from_, to_, weight_ >, 37
EdgeListGraph< vertexes, edge_list >, 37
EdgeListGraph< vertexes, edge_list >::ConvertTo< ADJACENCY_LIST >, 35
EdgeListGraph< vertexes, edge_list >::ConvertTo< GraphType >, 34

ForEach
 VertexStream< stream, graph >, 62
Functor< ResultType(ArgTypes...)>, 40
 operator(), 40
Functor< ResultType, ArgTypes >, 40
FunctorType, 11

GetVertexIndex
 AdjacencyListGraph< vertexes, adjacency_list >, 17
GLib::AddEdge< ADJACENCY_LIST, graph, edge >, 15
GLib::AddEdge< GraphType, graph, edge >, 15
Graph, 42
HasEdge

- AdjacencyListGraph< vertexes, adjacency_list >, 17
- NullType, 53
- Objects, 11
- Objects::Boolean< boolean >, 19
- Objects::Integer< integer >, 45
- operator()
 - Functor< ResultType(ArgTypes...)>, 40
- PointerStructureGraph< vertexes >, 53
- PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >, 36
- PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >::IterateThroughChildren< current_children >, 48
- PointerStructureGraph< vertexes >::DFS< current_vertex, unvisited_vertexes >::IterateThroughChildren< EmptyTypeList >, 48
- PointerStructureGraph< vertexes >::Node< vertex_, children_ >, 51
- result
 - CheckContainsConstructibleParent< type_list, T, false >, 19
 - CheckContainsParent< type_list, T, false >, 20
 - CheckIsBaseOf< true, parent, derived >, 24
 - TL::ContainsConstructibleParent< type_list, T >, 29
 - TL::ContainsParent< type_list, T >, 30
 - TL::HasDerivedAndConstructible< type_list, T >, 43
 - TL::IsBaseOf< parent, derived >, 46
- TL, 12
- TL::Add< T, 0, TypeList< Arg, Args... > >, 13
- TL::Add< T, 0, TypeList< Args... > >, 14
- TL::Add< T, ind, Arg, Args >, 13
- TL::Add< T, ind, TypeList< Arg, Args... > >, 14
- TL::Concatenate< front, back >, 26
- TL::Concatenate< front, EmptyTypeList >, 27
- TL::Contains< type_list, T >, 27
- TL::ContainsConstructibleParent< EmptyTypeList, T >, 29
- TL::ContainsConstructibleParent< type_list, T >, 28
 - result, 29
- TL::ContainsParent< EmptyTypeList, T >, 30
- TL::ContainsParent< type_list, T >, 29
 - result, 30
- TL::FindParentTypeList< T, type_list, type_lists >, 38
- TL::FindTypeListByClass< T, type_list, type_lists >, 39
- TL::GenerateTypeLists< 0 >, 42
- TL::GenerateTypeLists< n >, 41
- TL::HasDerivedAndConstructible< EmptyTypeList, T >, 43
- TL::HasDerivedAndConstructible< type_list, T >, 42
 - result, 43
- TL::IndexOf< EmptyTypeList, T >, 44
- TL::IndexOf< type_list, T >, 44
- TL::IndexOf< type_list, typename type_list::Head >, 45
- TL::IsBaseOf< EmptyTypeList, derived >, 46
- TL::IsBaseOf< EmptyTypeList, EmptyTypeList >, 46
- TL::IsBaseOf< parent, derived >, 45
 - result, 46
- TL::IsBaseOf< parent, EmptyTypeList >, 47
- TL::IsTypeList< T >, 47
- TL::IsTypeList< TypeList< Args... > >, 47
- TL::MostDerived< EmptyTypeList, T >, 50
- TL::MostDerived< type_list, T >, 49
- TL::MostDerivedAndConstructible< EmptyTypeList, T >, 51
- TL::MostDerivedAndConstructible< type_list, T >, 50
- TL::NoDuplicates< EmptyTypeList >, 52
- TL::NoDuplicates< type_list >, 52
- TL::Remove< EmptyTypeList, T >, 54
- TL::Remove< type_list, T >, 54
- TL::Remove< type_list, typename type_list::Head >, 55
- TL::RemoveAll< type_list, T >, 55
- TL::RemoveAll< type_list, typename type_list::Head >, 56
- TL::Replace< T, 0, TypeList< Arg, Args... > >, 57
- TL::Replace< T, ind, Arg, Args >, 56
- TL::Replace< T, ind, TypeList< Arg, Args... > >, 57
- TL::Size< EmptyTypeList >, 59
- TL::Size< type_list >, 58
- TL::TypeAt< type_list, 0 >, 60
- TL::TypeAt< type_list, ind >, 59
- TypeList< Args >, 60
- TypeList< H, T... >, 61
- TypeList< T >, 61
- VertexStream< EmptyTypeList, graph >, 63
- VertexStream< stream, graph >, 62
 - ForEach, 62