

## Compile-Time Graph Library

Generated by Doxygen 1.8.20



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>7</b>
3.1 Class List	7
<b>4 File Index</b>	<b>11</b>
4.1 File List	11
<b>5 Namespace Documentation</b>	<b>13</b>
5.1 GLib Namespace Reference	13
5.2 Objects Namespace Reference	13
5.2.1 Detailed Description	13
5.3 StreamFunctorType Namespace Reference	13
5.3.1 Detailed Description	14
5.3.2 Typedef Documentation	14
5.3.2.1 BinaryOperation	14
5.3.2.2 Consumer	14
5.3.2.3 Function	14
5.3.2.4 Predicate	14
5.4 TL Namespace Reference	15
5.4.1 Detailed Description	15
<b>6 Class Documentation</b>	<b>17</b>
6.1 TL::Add< T, ind, Arg, Args > Struct Template Reference	17
6.1.1 Detailed Description	17
6.2 TL::Add< T, 0, TypeList< Arg, Args... > > Struct Template Reference	17
6.2.1 Detailed Description	17
6.2.2 Member Typedef Documentation	18
6.2.2.1 result	18
6.3 TL::Add< T, 0, TypeList< Args... > > Struct Template Reference	18
6.3.1 Detailed Description	18
6.3.2 Member Typedef Documentation	18
6.3.2.1 result	18
6.4 TL::Add< T, ind, TypeList< Arg, Args... > > Struct Template Reference	19
6.4.1 Detailed Description	19
6.4.2 Member Typedef Documentation	19
6.4.2.1 end	19
6.4.2.2 result	20
6.5 GLib::AddEdge< GraphType, graph, edge > Struct Template Reference	20
6.5.1 Detailed Description	20

6.6 GLib::AddEdge< ADJACENCY_LIST, graph, edge > Struct Template Reference . . . . .	20
6.6.1 Detailed Description . . . . .	21
6.6.2 Member Typedef Documentation . . . . .	21
6.6.2.1 adjacent_vertexes . . . . .	21
6.6.2.2 new_adjacency_list . . . . .	21
6.6.2.3 new_adjacent_vertexes . . . . .	22
6.6.2.4 result . . . . .	22
6.6.3 Member Data Documentation . . . . .	22
6.6.3.1 vertex_num . . . . .	22
6.7 AdjacencyListGraph< nodes, adjacency_list > Struct Template Reference . . . . .	22
6.7.1 Detailed Description . . . . .	23
6.7.2 Member Typedef Documentation . . . . .	24
6.7.2.1 adjacency_list_ . . . . .	24
6.7.2.2 vertexes_ . . . . .	24
6.7.3 Member Function Documentation . . . . .	24
6.7.3.1 GetVertexIndex() . . . . .	24
6.7.3.2 HasEdge() . . . . .	25
6.7.4 Member Data Documentation . . . . .	25
6.7.4.1 TYPE . . . . .	25
6.8 AdjacencyMatrixGraph< nodes, matrix > Struct Template Reference . . . . .	25
6.8.1 Detailed Description . . . . .	26
6.8.2 Member Typedef Documentation . . . . .	26
6.8.2.1 matrix_ . . . . .	26
6.8.2.2 vertexes_ . . . . .	27
6.8.3 Member Data Documentation . . . . .	27
6.8.3.1 TYPE . . . . .	27
6.9 Objects::Boolean< boolean > Struct Template Reference . . . . .	27
6.9.1 Detailed Description . . . . .	27
6.9.2 Member Data Documentation . . . . .	27
6.9.2.1 value . . . . .	28
6.10 CheckContainsConstructibleParent< type_list, T, is_parent > Struct Template Reference . . . . .	28
6.10.1 Detailed Description . . . . .	28
6.11 CheckContainsConstructibleParent< type_list, T, false > Struct Template Reference . . . . .	28
6.11.1 Detailed Description . . . . .	28
6.11.2 Member Data Documentation . . . . .	29
6.11.2.1 result . . . . .	29
6.12 CheckContainsConstructibleParent< type_list, T, true > Struct Template Reference . . . . .	29
6.12.1 Detailed Description . . . . .	29
6.12.2 Member Data Documentation . . . . .	29
6.12.2.1 result . . . . .	30
6.13 CheckContainsParent< type_list, T, is_parent > Struct Template Reference . . . . .	30
6.13.1 Detailed Description . . . . .	30

6.14 CheckContainsParent< type_list, T, false > Struct Template Reference . . . . .	30
6.14.1 Detailed Description . . . . .	30
6.14.2 Member Data Documentation . . . . .	31
6.14.2.1 result . . . . .	31
6.15 CheckContainsParent< type_list, T, true > Struct Template Reference . . . . .	31
6.15.1 Detailed Description . . . . .	31
6.15.2 Member Data Documentation . . . . .	31
6.15.2.1 result . . . . .	32
6.16 CheckFindParentTypeList< contains_class, T, type_list, type_lists > Struct Template Reference . . . . .	32
6.16.1 Detailed Description . . . . .	32
6.16.2 Member Typedef Documentation . . . . .	32
6.16.2.1 result . . . . .	32
6.17 CheckFindParentTypeList< false, T, type_list, type_lists... > Struct Template Reference . . . . .	33
6.17.1 Detailed Description . . . . .	33
6.17.2 Member Typedef Documentation . . . . .	33
6.17.2.1 result . . . . .	33
6.18 CheckFindParentTypeList< true, T, type_list, type_lists... > Struct Template Reference . . . . .	33
6.18.1 Detailed Description . . . . .	34
6.18.2 Member Typedef Documentation . . . . .	34
6.18.2.1 result . . . . .	34
6.19 CheckFindTypeListByClass< contains_class, T, type_list, type_lists > Struct Template Reference . . . . .	34
6.19.1 Detailed Description . . . . .	34
6.19.2 Member Typedef Documentation . . . . .	34
6.19.2.1 result . . . . .	35
6.20 CheckFindTypeListByClass< false, T, type_list, type_lists... > Struct Template Reference . . . . .	35
6.20.1 Detailed Description . . . . .	35
6.20.2 Member Typedef Documentation . . . . .	35
6.20.2.1 result . . . . .	35
6.21 CheckFindTypeListByClass< true, T, type_list, type_lists... > Struct Template Reference . . . . .	36
6.21.1 Detailed Description . . . . .	36
6.21.2 Member Typedef Documentation . . . . .	36
6.21.2.1 result . . . . .	36
6.22 CheckHasDerivedAndConstructible< type_list, T, is_head_parent_of_T > Struct Template Reference . . . . .	36
6.22.1 Detailed Description . . . . .	36
6.23 CheckHasDerivedAndConstructible< type_list, T, false > Struct Template Reference . . . . .	37
6.23.1 Detailed Description . . . . .	37
6.23.2 Member Data Documentation . . . . .	37
6.23.2.1 result . . . . .	37
6.24 CheckHasDerivedAndConstructible< type_list, T, true > Struct Template Reference . . . . .	37
6.24.1 Detailed Description . . . . .	38
6.24.2 Member Data Documentation . . . . .	38
6.24.2.1 result . . . . .	38

6.25 CheckIsBaseOf< has_parent, parent, derived > Struct Template Reference . . . . .	38
6.25.1 Detailed Description . . . . .	38
6.26 CheckIsBaseOf< false, parent, derived > Struct Template Reference . . . . .	38
6.26.1 Detailed Description . . . . .	39
6.26.2 Member Data Documentation . . . . .	39
6.26.2.1 result . . . . .	39
6.27 CheckIsBaseOf< true, parent, derived > Struct Template Reference . . . . .	39
6.27.1 Detailed Description . . . . .	39
6.27.2 Member Data Documentation . . . . .	39
6.27.2.1 result . . . . .	40
6.28 CheckMostDerived< type_list, T, is_head_parent_of_T > Struct Template Reference . . . . .	40
6.28.1 Detailed Description . . . . .	40
6.28.2 Member Typedef Documentation . . . . .	40
6.28.2.1 result . . . . .	40
6.29 CheckMostDerived< type_list, T, false > Struct Template Reference . . . . .	41
6.29.1 Detailed Description . . . . .	41
6.29.2 Member Typedef Documentation . . . . .	41
6.29.2.1 result . . . . .	41
6.30 CheckMostDerived< type_list, T, true > Struct Template Reference . . . . .	41
6.30.1 Detailed Description . . . . .	41
6.30.2 Member Typedef Documentation . . . . .	42
6.30.2.1 result . . . . .	42
6.31 CheckMostDerivedAndConstructible< type_list, T, is_head_parent_of_T > Struct Template Reference . . . . .	42
6.31.1 Detailed Description . . . . .	42
6.32 CheckMostDerivedAndConstructible< type_list, T, false > Struct Template Reference . . . . .	42
6.32.1 Detailed Description . . . . .	43
6.32.2 Member Typedef Documentation . . . . .	43
6.32.2.1 result . . . . .	43
6.33 CheckMostDerivedAndConstructible< type_list, T, true > Struct Template Reference . . . . .	43
6.33.1 Detailed Description . . . . .	43
6.33.2 Member Typedef Documentation . . . . .	43
6.33.2.1 result . . . . .	44
6.34 TL::Concatenate< front, back > Struct Template Reference . . . . .	44
6.34.1 Detailed Description . . . . .	44
6.34.2 Member Typedef Documentation . . . . .	44
6.34.2.1 result . . . . .	45
6.34.2.2 reversed_front . . . . .	45
6.35 TL::Contains< type_list, T > Struct Template Reference . . . . .	45
6.35.1 Detailed Description . . . . .	45
6.35.2 Member Data Documentation . . . . .	46
6.35.2.1 result . . . . .	46
6.36 TL::ContainsConstructibleParent< type_list, T > Struct Template Reference . . . . .	46

6.36.1 Detailed Description . . . . .	46
6.36.2 Member Data Documentation . . . . .	46
6.36.2.1 result . . . . .	47
6.37 TL::ContainsConstructibleParent< EmptyTypeList, T > Struct Template Reference . . . . .	47
6.37.1 Detailed Description . . . . .	47
6.37.2 Member Data Documentation . . . . .	47
6.37.2.1 result . . . . .	47
6.38 TL::ContainsParent< type_list, T > Struct Template Reference . . . . .	48
6.38.1 Detailed Description . . . . .	48
6.38.2 Member Data Documentation . . . . .	48
6.38.2.1 result . . . . .	48
6.39 TL::ContainsParent< EmptyTypeList, T > Struct Template Reference . . . . .	49
6.39.1 Detailed Description . . . . .	49
6.39.2 Member Data Documentation . . . . .	49
6.39.2.1 result . . . . .	49
6.40 ConvertGraph< From, To, graph > Struct Template Reference . . . . .	49
6.40.1 Detailed Description . . . . .	49
6.41 ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph > Struct Template Reference . . . . .	50
6.41.1 Detailed Description . . . . .	50
6.41.2 Member Typedef Documentation . . . . .	50
6.41.2.1 result . . . . .	51
6.42 ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph > Struct Template Reference . . . . .	51
6.42.1 Detailed Description . . . . .	51
6.42.2 Member Typedef Documentation . . . . .	51
6.42.2.1 result [1/2] . . . . .	52
6.42.2.2 result [2/2] . . . . .	52
6.43 ConvertGraph< EDGE_LIST, POINTER_STRUCTURE, graph > Struct Template Reference . . . . .	52
6.43.1 Detailed Description . . . . .	52
6.43.2 Member Typedef Documentation . . . . .	52
6.43.2.1 adjacency_list . . . . .	53
6.43.2.2 result . . . . .	53
6.44 ConvertGraph< POINTER_STRUCTURE, EDGE_LIST, graph > Struct Template Reference . . . . .	53
6.44.1 Detailed Description . . . . .	53
6.45 ConvertGraph< type, type, graph > Struct Template Reference . . . . .	53
6.45.1 Detailed Description . . . . .	54
6.45.2 Member Typedef Documentation . . . . .	54
6.45.2.1 result . . . . .	54
6.46 EdgeListGraph< nodes, edge_list >::ConvertTo< type > Struct Template Reference . . . . .	54
6.46.1 Detailed Description . . . . .	54
6.46.2 Member Typedef Documentation . . . . .	55
6.46.2.1 result . . . . .	55
6.47 PointerStructureGraph< nodes >::ConvertTo< type > Struct Template Reference . . . . .	55

6.47.1 Detailed Description	55
6.47.2 Member Typedef Documentation	56
6.47.2.1 result	56
6.48 AdjacencyListGraph< nodes, adjacency_list >::ConvertTo< type > Struct Template Reference	56
6.48.1 Detailed Description	56
6.48.2 Member Typedef Documentation	57
6.48.2.1 result	57
6.49 AdjacencyMatrixGraph< nodes, matrix >::ConvertTo< type > Struct Template Reference	57
6.49.1 Detailed Description	57
6.49.2 Member Typedef Documentation	58
6.49.2.1 result	58
6.50 GLib::DFS< cur_node, graph, visited_nodes > Struct Template Reference	58
6.50.1 Detailed Description	58
6.50.2 Member Typedef Documentation	59
6.50.2.1 iterate_through_children	59
6.50.2.2 new_visited	59
6.50.2.3 result	59
6.50.2.4 upd_visited	60
6.51 Edge< from_, to_, weight_ > Struct Template Reference	60
6.51.1 Detailed Description	60
6.51.2 Member Typedef Documentation	60
6.51.2.1 from	61
6.51.2.2 to	61
6.51.2.3 weight	61
6.52 EdgeListGraph< nodes, edge_list > Struct Template Reference	61
6.52.1 Detailed Description	62
6.52.2 Member Typedef Documentation	62
6.52.2.1 edge_list_	62
6.52.2.2 vertexes_	62
6.52.3 Member Data Documentation	63
6.52.3.1 TYPE	63
6.53 GLib::FindNodeByVertex< vertex, graph > Struct Template Reference	63
6.53.1 Detailed Description	63
6.53.2 Member Typedef Documentation	64
6.53.2.1 result	64
6.54 GLib::FindNodeByVertex< vertex, EmptyTypeList > Struct Template Reference	64
6.54.1 Detailed Description	64
6.54.2 Member Typedef Documentation	64
6.54.2.1 result	65
6.55 TL::FindParentTypeList< T, type_list, type_lists > Struct Template Reference	65
6.55.1 Detailed Description	65
6.55.2 Member Typedef Documentation	65



6.55.2.1 result . . . . .	66
6.56 GLib::FindPath< graph_raw, start, finish > Struct Template Reference . . . . .	66
6.56.1 Detailed Description . . . . .	66
6.56.2 Member Typedef Documentation . . . . .	67
6.56.2.1 dfs_search . . . . .	67
6.56.2.2 finish_node . . . . .	67
6.56.2.3 graph . . . . .	67
6.56.2.4 iterate_through_edges . . . . .	68
6.56.2.5 path . . . . .	68
6.56.2.6 reversed . . . . .	68
6.56.2.7 reversed_path . . . . .	68
6.56.2.8 reversed_weights . . . . .	68
6.56.2.9 start_node . . . . .	69
6.56.2.10 weights . . . . .	69
6.57 TL::FindTypeListByClass< T, type_list, type_lists > Struct Template Reference . . . . .	69
6.57.1 Detailed Description . . . . .	69
6.57.2 Member Typedef Documentation . . . . .	70
6.57.2.1 result . . . . .	70
6.58 Functor< ResultType, ArgTypes > Class Template Reference . . . . .	70
6.58.1 Detailed Description . . . . .	70
6.59 Functor< ResultType(ArgTypes...) > Class Template Reference . . . . .	70
6.59.1 Detailed Description . . . . .	70
6.59.2 Constructor & Destructor Documentation . . . . .	71
6.59.2.1 Functor() [1/4] . . . . .	71
6.59.2.2 Functor() [2/4] . . . . .	71
6.59.2.3 Functor() [3/4] . . . . .	71
6.59.2.4 Functor() [4/4] . . . . .	71
6.59.3 Member Function Documentation . . . . .	72
6.59.3.1 operator>() . . . . .	72
6.59.3.2 operator=() . . . . .	72
6.60 TL::GenerateTypeLists< n > Struct Template Reference . . . . .	72
6.60.1 Detailed Description . . . . .	72
6.60.2 Member Typedef Documentation . . . . .	73
6.60.2.1 result . . . . .	73
6.61 TL::GenerateTypeLists< 0 > Struct Reference . . . . .	73
6.61.1 Detailed Description . . . . .	73
6.61.2 Member Typedef Documentation . . . . .	73
6.61.2.1 result . . . . .	74
6.62 GLib::GetReachedVertexes< graph, start > Struct Template Reference . . . . .	74
6.62.1 Detailed Description . . . . .	74
6.62.2 Member Typedef Documentation . . . . .	75
6.62.2.1 dfs_search . . . . .	75

6.62.2.2 result . . . . .	75
6.62.2.3 start_node . . . . .	75
6.63 Graph Struct Reference . . . . .	75
6.63.1 Detailed Description . . . . .	76
6.64 TL::HasDerivedAndConstructible< type_list, T > Struct Template Reference . . . . .	76
6.64.1 Detailed Description . . . . .	76
6.64.2 Member Data Documentation . . . . .	76
6.64.2.1 result . . . . .	77
6.65 TL::HasDerivedAndConstructible< EmptyTypeList, T > Struct Template Reference . . . . .	77
6.65.1 Detailed Description . . . . .	77
6.65.2 Member Data Documentation . . . . .	77
6.65.2.1 result . . . . .	77
6.66 TL::IndexOf< type_list, T > Struct Template Reference . . . . .	78
6.66.1 Detailed Description . . . . .	78
6.66.2 Member Data Documentation . . . . .	78
6.66.2.1 value . . . . .	78
6.67 TL::IndexOf< EmptyTypeList, T > Struct Template Reference . . . . .	78
6.67.1 Detailed Description . . . . .	79
6.67.2 Member Data Documentation . . . . .	79
6.67.2.1 value . . . . .	79
6.68 TL::IndexOf< type_list, typename type_list::Head > Struct Template Reference . . . . .	79
6.68.1 Detailed Description . . . . .	79
6.68.2 Member Data Documentation . . . . .	80
6.68.2.1 value . . . . .	80
6.69 Objects::Integer< integer > Struct Template Reference . . . . .	80
6.69.1 Detailed Description . . . . .	80
6.69.2 Member Data Documentation . . . . .	80
6.69.2.1 value . . . . .	80
6.70 TL::IsBaseOf< parent, derived > Struct Template Reference . . . . .	81
6.70.1 Detailed Description . . . . .	81
6.70.2 Member Data Documentation . . . . .	81
6.70.2.1 result . . . . .	81
6.71 TL::IsBaseOf< EmptyTypeList, derived > Struct Template Reference . . . . .	82
6.71.1 Detailed Description . . . . .	82
6.71.2 Member Data Documentation . . . . .	82
6.71.2.1 result . . . . .	82
6.72 TL::IsBaseOf< EmptyTypeList, EmptyTypeList > Struct Reference . . . . .	82
6.72.1 Detailed Description . . . . .	82
6.72.2 Member Data Documentation . . . . .	83
6.72.2.1 result . . . . .	83
6.73 TL::IsBaseOf< parent, EmptyTypeList > Struct Template Reference . . . . .	83
6.73.1 Detailed Description . . . . .	83

6.73.2 Member Data Documentation	83
6.73.2.1 result	83
6.74 TL::IsTypeList< T > Struct Template Reference	84
6.74.1 Detailed Description	84
6.75 TL::IsTypeList< TypeList< Args... > > Struct Template Reference	84
6.75.1 Detailed Description	85
6.76 GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< cur_children, cur_visited > Struct Template Reference	85
6.76.1 Detailed Description	85
6.76.2 Member Typedef Documentation	85
6.76.2.1 cur_child	85
6.76.2.2 cur_edge	86
6.76.2.3 new_visited	86
6.76.2.4 result	86
6.77 GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< EmptyTypeList, cur_↔ unvisited > Struct Template Reference	86
6.77.1 Detailed Description	87
6.77.2 Member Typedef Documentation	87
6.77.2.1 new_visited	87
6.77.2.2 result	87
6.78 GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< cur_edges, wanted_node > Struct Template Reference	87
6.78.1 Detailed Description	88
6.78.2 Member Typedef Documentation	88
6.78.2.1 cur_edge	88
6.78.2.2 path	88
6.78.2.3 weights	89
6.78.3 Member Data Documentation	89
6.78.3.1 found	89
6.79 GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< cur_edges > Struct Template Reference	89
6.79.1 Detailed Description	90
6.79.2 Member Typedef Documentation	90
6.79.2.1 cur_edge	90
6.79.2.2 result	90
6.80 ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< edge_list > Struct Template Reference	90
6.80.1 Detailed Description	91
6.80.2 Member Typedef Documentation	91
6.80.2.1 result [1/2]	91
6.80.2.2 result [2/2]	91
6.81 GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< EmptyTypeList > Struct Reference	91
6.81.1 Detailed Description	92

6.81.2 Member Typedef Documentation	92
6.81.2.1 result	92
6.82 ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< EmptyTypeList > Struct Reference	92
6.82.1 Detailed Description	92
6.82.2 Member Typedef Documentation	92
6.82.2.1 result [1/2]	93
6.82.2.2 result [2/2]	93
6.83 GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< EmptyTypeList, wanted_node > Struct Template Reference	93
6.83.1 Detailed Description	93
6.83.2 Member Typedef Documentation	93
6.83.2.1 path	94
6.83.2.2 weights	94
6.84 TL::Reverse< type_list >::IterateThroughElements< cur_type_list, cur_result > Struct Template Reference	94
6.84.1 Detailed Description	94
6.84.2 Member Typedef Documentation	94
6.84.2.1 result	95
6.85 TL::Reverse< type_list >::IterateThroughElements< EmptyTypeList, cur_result > Struct Template Reference	95
6.85.1 Detailed Description	95
6.85.2 Member Typedef Documentation	95
6.85.2.1 result	95
6.86 GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< cur_nodes > Struct Template Reference	96
6.86.1 Detailed Description	96
6.86.2 Member Typedef Documentation	96
6.86.2.1 cur_node	96
6.86.2.2 result	96
6.87 GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< EmptyTypeList > Struct Reference	97
6.87.1 Detailed Description	97
6.87.2 Member Typedef Documentation	97
6.87.2.1 result	97
6.88 TL::Concatenate< front, back >::IterateThroughReversedFront< elements, current > Struct Template Reference	97
6.88.1 Detailed Description	98
6.88.2 Member Typedef Documentation	98
6.88.2.1 added	98
6.88.2.2 result	98
6.89 TL::Concatenate< front, back >::IterateThroughReversedFront< EmptyTypeList, current > Struct Template Reference	98
6.89.1 Detailed Description	99
6.89.2 Member Typedef Documentation	99

6.89.2.1 result . . . . .	99
6.90 ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructure↔ Graph< current_vertexes, current_adjacency_list > Struct Template Reference . . . . .	99
6.90.1 Detailed Description . . . . .	99
6.90.2 Member Typedef Documentation . . . . .	100
6.90.2.1 result . . . . .	100
6.90.2.2 type_list_without_first . . . . .	100
6.91 ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructure↔ Graph< EmptyTypeList, EmptyTypeList > Struct Reference . . . . .	100
6.91.1 Detailed Description . . . . .	100
6.91.2 Member Typedef Documentation . . . . .	101
6.91.2.1 result . . . . .	101
6.92 TL::MostDerived< type_list, T > Struct Template Reference . . . . .	101
6.92.1 Detailed Description . . . . .	101
6.92.2 Member Typedef Documentation . . . . .	101
6.92.2.1 result . . . . .	102
6.93 TL::MostDerived< EmptyTypeList, T > Struct Template Reference . . . . .	102
6.93.1 Detailed Description . . . . .	102
6.93.2 Member Typedef Documentation . . . . .	102
6.93.2.1 result . . . . .	102
6.94 TL::MostDerivedAndConstructible< type_list, T > Struct Template Reference . . . . .	103
6.94.1 Detailed Description . . . . .	103
6.94.2 Member Typedef Documentation . . . . .	103
6.94.2.1 result . . . . .	103
6.95 TL::MostDerivedAndConstructible< EmptyTypeList, T > Struct Template Reference . . . . .	104
6.95.1 Detailed Description . . . . .	104
6.95.2 Member Typedef Documentation . . . . .	104
6.95.2.1 result . . . . .	104
6.96 TL::NoDuplicates< type_list > Struct Template Reference . . . . .	104
6.96.1 Detailed Description . . . . .	104
6.96.2 Member Typedef Documentation . . . . .	105
6.96.2.1 result . . . . .	105
6.97 TL::NoDuplicates< EmptyTypeList > Struct Reference . . . . .	105
6.97.1 Detailed Description . . . . .	105
6.97.2 Member Typedef Documentation . . . . .	106
6.97.2.1 result . . . . .	106
6.98 NullType Struct Reference . . . . .	106
6.98.1 Detailed Description . . . . .	106
6.99 PointerStructureGraph< nodes > Struct Template Reference . . . . .	106
6.99.1 Detailed Description . . . . .	107
6.99.2 Member Typedef Documentation . . . . .	107
6.99.2.1 nodes_ . . . . .	107
6.99.3 Member Data Documentation . . . . .	107

6.99.3.1 TYPE	108
6.100 PointerStructureNode< vertex_, children_ > Struct Template Reference	108
6.100.1 Detailed Description	108
6.100.2 Member Typedef Documentation	108
6.100.2.1 children	109
6.100.2.2 vertex	109
6.101 TL::Remove< type_list, T > Struct Template Reference	109
6.101.1 Detailed Description	109
6.101.2 Member Typedef Documentation	110
6.101.2.1 result	110
6.102 TL::Remove< EmptyTypeList, T > Struct Template Reference	110
6.102.1 Detailed Description	110
6.102.2 Member Typedef Documentation	110
6.102.2.1 result	110
6.103 TL::Remove< type_list, typename type_list::Head > Struct Template Reference	111
6.103.1 Detailed Description	111
6.103.2 Member Typedef Documentation	111
6.103.2.1 result	111
6.104 TL::RemoveAll< type_list, T > Struct Template Reference	111
6.104.1 Detailed Description	111
6.104.2 Member Typedef Documentation	112
6.104.2.1 result	112
6.105 TL::RemoveAll< type_list, typename type_list::Head > Struct Template Reference	112
6.105.1 Detailed Description	112
6.105.2 Member Typedef Documentation	113
6.105.2.1 result	113
6.106 TL::Replace< T, ind, Arg, Args > Struct Template Reference	113
6.106.1 Detailed Description	113
6.107 TL::Replace< T, 0, TypeList< Arg, Args... > > Struct Template Reference	113
6.107.1 Detailed Description	114
6.107.2 Member Typedef Documentation	114
6.107.2.1 result	114
6.108 TL::Replace< T, ind, TypeList< Arg, Args... > > Struct Template Reference	114
6.108.1 Detailed Description	115
6.108.2 Member Typedef Documentation	115
6.108.2.1 end	115
6.108.2.2 result	115
6.109 TL::Reverse< type_list > Struct Template Reference	115
6.109.1 Detailed Description	115
6.109.2 Member Typedef Documentation	116
6.109.2.1 result	116
6.110 TL::Size< type_list > Struct Template Reference	116

6.110.1 Detailed Description	116
6.110.2 Member Data Documentation	117
6.110.2.1 size	117
6.111 TL::Size< EmptyTypeList > Struct Reference	117
6.111.1 Detailed Description	117
6.111.2 Member Data Documentation	117
6.111.2.1 size	118
6.112 Stream< T > Class Template Reference	118
6.112.1 Detailed Description	118
6.112.2 Constructor & Destructor Documentation	118
6.112.2.1 Stream()	119
6.112.3 Member Function Documentation	119
6.112.3.1 Collect()	119
6.112.3.2 Filter()	119
6.112.3.3 ForEach()	120
6.112.3.4 Map()	120
6.112.3.5 Reduce()	120
6.113 TL::TypeAt< type_list, ind > Struct Template Reference	121
6.113.1 Detailed Description	121
6.113.2 Member Typedef Documentation	121
6.113.2.1 value	122
6.114 TL::TypeAt< type_list, 0 > Struct Template Reference	122
6.114.1 Detailed Description	122
6.114.2 Member Typedef Documentation	122
6.114.2.1 value	122
6.115 TypeList< Args > Struct Template Reference	123
6.115.1 Detailed Description	123
6.115.2 Member Typedef Documentation	123
6.115.2.1 Head	123
6.115.2.2 Tail	123
6.116 TypeList< H, T... > Struct Template Reference	124
6.116.1 Detailed Description	124
6.116.2 Member Typedef Documentation	124
6.116.2.1 Head	124
6.116.2.2 Tail	124
6.117 TypeList< T > Struct Template Reference	125
6.117.1 Detailed Description	125
6.117.2 Member Typedef Documentation	125
6.117.2.1 Head	125
6.117.2.2 Tail	125
6.118 VertexStream< stream, graph > Class Template Reference	126
6.118.1 Detailed Description	126

6.118.2 Member Function Documentation	126
6.118.2.1 MapVertexesToIndexes()	126
6.118.2.2 MapVertexesToReversedIndexes()	127
6.119 VertexStream< EmptyTypeList, graph > Struct Template Reference	127
6.119.1 Detailed Description	127
6.119.2 Member Function Documentation	127
6.119.2.1 MapVertexesToReversedIndexes()	127
<b>7 File Documentation</b>	<b>129</b>
7.1 Debug/CodeAnalysisResultManifest.txt File Reference	129
7.2 Debug/library.vcxproj.FileListAbsolute.txt File Reference	129
7.3 functor.h File Reference	129
7.4 graph/edge.h File Reference	129
7.5 graph/examples/graph_examples.cpp File Reference	129
7.5.1 Function Documentation	130
7.5.1.1 main()	130
7.6 graph/examples/vertex_stream_example.cpp File Reference	130
7.6.1 Function Documentation	130
7.6.1.1 Add1()	130
7.6.1.2 IsAmongFirst3()	131
7.6.1.3 main()	131
7.7 graph/GLib/add_edge.h File Reference	131
7.8 graph/GLib/dfs.h File Reference	131
7.9 graph/GLib/find_node_by_vertex.h File Reference	132
7.10 graph/GLib/find_path.h File Reference	132
7.11 graph/GLib/get_reached_vertexes.h File Reference	133
7.12 graph/GLib/map_indexes_to_vertexes.h File Reference	133
7.13 graph/graphs/adjacency_list_graph.h File Reference	133
7.14 graph/graphs/adjacency_matrix_graph.h File Reference	133
7.15 graph/graphs/convert_from_edge_list.h File Reference	134
7.16 graph/graphs/convert_from_pointer_structure.h File Reference	134
7.17 graph/graphs/convert_graph.h File Reference	134
7.18 graph/graphs/convert_to_adjacency_list.h File Reference	135
7.19 graph/graphs/convert_to_pointer_structure.h File Reference	135
7.20 graph/graphs/edge_list_graph.h File Reference	136
7.21 graph/graphs/graph.h File Reference	136
7.22 graph/graphs/graph_type.h File Reference	136
7.22.1 Enumeration Type Documentation	136
7.22.1.1 GraphType	136
7.23 graph/graphs/pointer_structure_graph.h File Reference	137
7.24 graph/graphs/pointer_structure_node.h File Reference	137
7.25 graph/objects.h File Reference	137



7.26 graph/stream.h File Reference . . . . .	138
7.27 graph/stream_functor_type.h File Reference . . . . .	138
7.28 graph/vertex_stream.h File Reference . . . . .	138
7.29 TL/add.h File Reference . . . . .	139
7.30 TL/concatenate.h File Reference . . . . .	139
7.31 TL/contains.h File Reference . . . . .	139
7.32 TL/contains_constructible_parent.h File Reference . . . . .	140
7.33 TL/contains_parent.h File Reference . . . . .	140
7.34 TL/find_parent_type_list.h File Reference . . . . .	140
7.35 TL/find_type_list_by_class.h File Reference . . . . .	141
7.36 TL/generate_type_lists.h File Reference . . . . .	141
7.37 TL/has_derived_and_constructible.h File Reference . . . . .	142
7.38 TL/index_of.h File Reference . . . . .	142
7.39 TL/is_base_of.h File Reference . . . . .	142
7.40 TL/is_type_list.h File Reference . . . . .	143
7.41 TL/most_derived.h File Reference . . . . .	143
7.42 TL/most_derived_and_constructible.h File Reference . . . . .	144
7.43 TL/no_duplicates.h File Reference . . . . .	144
7.44 TL/null_type.h File Reference . . . . .	144
7.45 TL/remove.h File Reference . . . . .	144
7.46 TL/replace.h File Reference . . . . .	145
7.47 TL/reverse.h File Reference . . . . .	145
7.48 TL/size.h File Reference . . . . .	146
7.49 TL/type_at.h File Reference . . . . .	146
7.50 TL/type_list.h File Reference . . . . .	146
7.50.1 Typedef Documentation . . . . .	147
7.50.1.1 EmptyTypeList . . . . .	147
7.50.1.2 Typelist . . . . .	147
<b>8 Example Documentation</b> . . . . .	<b>149</b>
8.1 get_reached_vertexes.h . . . . .	149
8.2 graph_examples.cpp . . . . .	149
8.3 vertex_stream_example.cpp . . . . .	149
<b>Index</b> . . . . .	<b>151</b>



# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">GLib</a>	13
<a href="#">Objects</a>	13
<a href="#">StreamFuncorType</a>	13
<a href="#">TL</a>	15



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

TL::Add< T, ind, Arg, Args > . . . . .	17
TL::Add< T, 0, TypeList< Arg, Args... > > . . . . .	17
TL::Add< T, 0, TypeList< Args... > > . . . . .	18
TL::Add< T, ind, TypeList< Arg, Args... > > . . . . .	19
GLib::AddEdge< GraphType, graph, edge > . . . . .	20
GLib::AddEdge< ADJACENCY_LIST, graph, edge > . . . . .	20
AdjacencyMatrixGraph< nodes, matrix > . . . . .	25
Objects::Boolean< boolean > . . . . .	27
CheckContainsConstructibleParent< type_list, T, is_parent > . . . . .	28
CheckContainsConstructibleParent< type_list, T, false > . . . . .	28
CheckContainsConstructibleParent< type_list, T, true > . . . . .	29
CheckContainsParent< type_list, T, is_parent > . . . . .	30
CheckContainsParent< type_list, T, false > . . . . .	30
CheckContainsParent< type_list, T, true > . . . . .	31
CheckFindParentTypeList< contains_class, T, type_list, type_lists > . . . . .	32
CheckFindParentTypeList< false, T, type_list, type_lists... > . . . . .	33
CheckFindParentTypeList< true, T, type_list, type_lists... > . . . . .	33
CheckFindTypeListByClass< contains_class, T, type_list, type_lists > . . . . .	34
CheckFindTypeListByClass< false, T, type_list, type_lists... > . . . . .	35
CheckFindTypeListByClass< true, T, type_list, type_lists... > . . . . .	36
CheckHasDerivedAndConstructible< type_list, T, is_head_parent_of_T > . . . . .	36
CheckHasDerivedAndConstructible< type_list, T, false > . . . . .	37
CheckHasDerivedAndConstructible< type_list, T, true > . . . . .	37
CheckIsBaseOf< has_parent, parent, derived > . . . . .	38
CheckIsBaseOf< false, parent, derived > . . . . .	38
CheckIsBaseOf< true, parent, derived > . . . . .	39
CheckMostDerived< type_list, T, is_head_parent_of_T > . . . . .	40
CheckMostDerived< type_list, T, false > . . . . .	41
CheckMostDerived< type_list, T, true > . . . . .	41
CheckMostDerivedAndConstructible< type_list, T, is_head_parent_of_T > . . . . .	42
CheckMostDerivedAndConstructible< type_list, T, false > . . . . .	42
CheckMostDerivedAndConstructible< type_list, T, true > . . . . .	43
TL::Concatenate< front, back > . . . . .	44
TL::Contains< type_list, T > . . . . .	45
TL::ContainsConstructibleParent< type_list, T > . . . . .	46

TL::ContainsConstructibleParent< EmptyTypeList, T > . . . . .	47
TL::ContainsParent< type_list, T > . . . . .	48
TL::ContainsParent< EmptyTypeList, T > . . . . .	49
ConvertGraph< From, To, graph > . . . . .	49
ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph > . . . . .	50
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph > . . . . .	51
ConvertGraph< EDGE_LIST, POINTER_STRUCTURE, graph > . . . . .	52
ConvertGraph< POINTER_STRUCTURE, EDGE_LIST, graph > . . . . .	53
ConvertGraph< type, type, graph > . . . . .	53
EdgeListGraph< nodes, edge_list >::ConvertTo< type > . . . . .	54
PointerStructureGraph< nodes >::ConvertTo< type > . . . . .	55
AdjacencyListGraph< nodes, adjacency_list >::ConvertTo< type > . . . . .	56
AdjacencyMatrixGraph< nodes, matrix >::ConvertTo< type > . . . . .	57
GLib::DFS< cur_node, graph, visited_nodes > . . . . .	58
Edge< from_, to_, weight_ > . . . . .	60
EdgeListGraph< nodes, edge_list > . . . . .	61
false_type	
TL::IsTypeList< T > . . . . .	84
GLib::FindNodeByVertex< vertex, graph > . . . . .	63
GLib::FindNodeByVertex< vertex, EmptyTypeList > . . . . .	64
TL::FindParentTypeList< T, type_list, type_lists > . . . . .	65
GLib::FindPath< graph_raw, start, finish > . . . . .	66
TL::FindTypeListByClass< T, type_list, type_lists > . . . . .	69
Functor< ResultType, ArgTypes > . . . . .	70
Functor< ResultType(ArgTypes...) > . . . . .	70
TL::GenerateTypeLists< n > . . . . .	72
TL::GenerateTypeLists< 0 > . . . . .	73
GLib::GetReachedVertexes< graph, start > . . . . .	74
Graph . . . . .	75
AdjacencyListGraph< nodes, adjacency_list > . . . . .	22
PointerStructureGraph< nodes > . . . . .	106
TL::HasDerivedAndConstructible< type_list, T > . . . . .	76
TL::HasDerivedAndConstructible< EmptyTypeList, T > . . . . .	77
TL::IndexOf< type_list, T > . . . . .	78
TL::IndexOf< EmptyTypeList, T > . . . . .	78
TL::IndexOf< type_list, typename type_list::Head > . . . . .	79
Objects::Integer< integer > . . . . .	80
TL::IsBaseOf< parent, derived > . . . . .	81
TL::IsBaseOf< EmptyTypeList, derived > . . . . .	82
TL::IsBaseOf< EmptyTypeList, EmptyTypeList > . . . . .	82
TL::IsBaseOf< parent, EmptyTypeList > . . . . .	83
GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< cur_children, cur_visited > . . . . .	85
GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< EmptyTypeList, cur_unvisited > . . . . .	86
GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< cur_edges, wanted_node > . . . . .	87
GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< cur_edges > . . . . .	89
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< edge_list > . . . . .	90
GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< EmptyTypeList > . . . . .	91
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< EmptyTypeList > . . . . .	92
GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< EmptyTypeList, wanted_node > . . . . .	93
TL::Reverse< type_list >::IterateThroughElements< cur_type_list, cur_result > . . . . .	94
TL::Reverse< type_list >::IterateThroughElements< EmptyTypeList, cur_result > . . . . .	95
GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< cur_nodes > . . . . .	96
GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< EmptyTypeList > . . . . .	97
TL::Concatenate< front, back >::IterateThroughReversedFront< elements, current > . . . . .	97
TL::Concatenate< front, back >::IterateThroughReversedFront< EmptyTypeList, current > . . . . .	98
ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< current_vertexes, current_adjacency_list > . . . . .	99

ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< EmptyTypeList, EmptyTypeList > . . . . .	100
TL::MostDerived< type_list, T > . . . . .	101
TL::MostDerived< EmptyTypeList, T > . . . . .	102
TL::MostDerivedAndConstructible< type_list, T > . . . . .	103
TL::MostDerivedAndConstructible< EmptyTypeList, T > . . . . .	104
TL::NoDuplicates< type_list > . . . . .	104
TL::NoDuplicates< EmptyTypeList > . . . . .	105
NullType . . . . .	106
PointerStructureNode< vertex_, children_ > . . . . .	108
TL::Remove< type_list, T > . . . . .	109
TL::Remove< EmptyTypeList, T > . . . . .	110
TL::Remove< type_list, typename type_list::Head > . . . . .	111
TL::RemoveAll< type_list, T > . . . . .	111
TL::RemoveAll< type_list, typename type_list::Head > . . . . .	112
TL::Replace< T, ind, Arg, Args > . . . . .	113
TL::Replace< T, 0, TypeList< Arg, Args... > > . . . . .	113
TL::Replace< T, ind, TypeList< Arg, Args... > > . . . . .	114
TL::Reverse< type_list > . . . . .	115
TL::Size< type_list > . . . . .	116
TL::Size< EmptyTypeList > . . . . .	117
Stream< T > . . . . .	118
true_type	
TL::IsTypeList< TypeList< Args... > > . . . . .	84
TL::TypeAt< type_list, ind > . . . . .	121
TL::TypeAt< type_list, 0 > . . . . .	122
TypeList< Args > . . . . .	123
TypeList< H, T... > . . . . .	124
TypeList< T > . . . . .	125
VertexStream< stream, graph > . . . . .	126
VertexStream< EmptyTypeList, graph > . . . . .	127





## 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 > . . . . .	17
TL::Add< T, 0, TypeList< Arg, Args... > > . . . . .	17
TL::Add< T, 0, TypeList< Args... > > . . . . .	18
TL::Add< T, ind, TypeList< Arg, Args... > > . . . . .	19
GLib::AddEdge< GraphType, graph, edge > . . . . .	20
GLib::AddEdge< ADJACENCY_LIST, graph, edge > . . . . .	20
AdjacencyListGraph< nodes, adjacency_list > . . . . .	22
AdjacencyMatrixGraph< nodes, matrix > . . . . .	25
Objects::Boolean< boolean > . . . . .	27
CheckContainsConstructibleParent< type_list, T, is_parent > . . . . .	28
CheckContainsConstructibleParent< type_list, T, false > . . . . .	28
CheckContainsConstructibleParent< type_list, T, true > . . . . .	29
CheckContainsParent< type_list, T, is_parent > . . . . .	30
CheckContainsParent< type_list, T, false > . . . . .	30
CheckContainsParent< type_list, T, true > . . . . .	31
CheckFindParentTypeList< contains_class, T, type_list, type_lists > . . . . .	32
CheckFindParentTypeList< false, T, type_list, type_lists... > . . . . .	33
CheckFindParentTypeList< true, T, type_list, type_lists... > . . . . .	33
CheckFindTypeListByClass< contains_class, T, type_list, type_lists > . . . . .	34
CheckFindTypeListByClass< false, T, type_list, type_lists... > . . . . .	35
CheckFindTypeListByClass< true, T, type_list, type_lists... > . . . . .	36
CheckHasDerivedAndConstructible< type_list, T, is_head_parent_of_T > . . . . .	36
CheckHasDerivedAndConstructible< type_list, T, false > . . . . .	37
CheckHasDerivedAndConstructible< type_list, T, true > . . . . .	37
CheckIsBaseOf< has_parent, parent, derived > . . . . .	38
CheckIsBaseOf< false, parent, derived > . . . . .	38
CheckIsBaseOf< true, parent, derived > . . . . .	39
CheckMostDerived< type_list, T, is_head_parent_of_T > . . . . .	40
CheckMostDerived< type_list, T, false > . . . . .	41
CheckMostDerived< type_list, T, true > . . . . .	41
CheckMostDerivedAndConstructible< type_list, T, is_head_parent_of_T > . . . . .	42
CheckMostDerivedAndConstructible< type_list, T, false > . . . . .	42
CheckMostDerivedAndConstructible< type_list, T, true > . . . . .	43
TL::Concatenate< front, back > . . . . .	44
TL::Contains< type_list, T > . . . . .	45

TL::ContainsConstructibleParent< type_list, T > . . . . .	46
TL::ContainsConstructibleParent< EmptyTypeList, T > . . . . .	47
TL::ContainsParent< type_list, T > . . . . .	48
TL::ContainsParent< EmptyTypeList, T > . . . . .	49
ConvertGraph< From, To, graph > . . . . .	49
ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph > . . . . .	50
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph > . . . . .	51
ConvertGraph< EDGE_LIST, POINTER_STRUCTURE, graph > . . . . .	52
ConvertGraph< POINTER_STRUCTURE, EDGE_LIST, graph > . . . . .	53
ConvertGraph< type, type, graph > . . . . .	53
EdgeListGraph< nodes, edge_list >::ConvertTo< type > . . . . .	54
PointerStructureGraph< nodes >::ConvertTo< type > . . . . .	55
AdjacencyListGraph< nodes, adjacency_list >::ConvertTo< type > . . . . .	56
AdjacencyMatrixGraph< nodes, matrix >::ConvertTo< type > . . . . .	57
GLib::DFS< cur_node, graph, visited_nodes > . . . . .	58
Edge< from_, to_, weight_ > . . . . .	60
EdgeListGraph< nodes, edge_list > . . . . .	61
GLib::FindNodeByVertex< vertex, graph > . . . . .	63
GLib::FindNodeByVertex< vertex, EmptyTypeList > . . . . .	64
TL::FindParentTypeList< T, type_list, type_lists > . . . . .	65
GLib::FindPath< graph_raw, start, finish > . . . . .	66
TL::FindTypeListByClass< T, type_list, type_lists > . . . . .	69
Functor< ResultType, ArgTypes > . . . . .	70
Functor< ResultType(ArgTypes...) > . . . . .	70
TL::GenerateTypeLists< n > . . . . .	72
TL::GenerateTypeLists< 0 > . . . . .	73
GLib::GetReachedVertexes< graph, start > . . . . .	74
Graph . . . . .	75
TL::HasDerivedAndConstructible< type_list, T > . . . . .	76
TL::HasDerivedAndConstructible< EmptyTypeList, T > . . . . .	77
TL::IndexOf< type_list, T > . . . . .	78
TL::IndexOf< EmptyTypeList, T > . . . . .	78
TL::IndexOf< type_list, typename type_list::Head > . . . . .	79
Objects::Integer< integer > . . . . .	80
TL::IsBaseOf< parent, derived > . . . . .	81
TL::IsBaseOf< EmptyTypeList, derived > . . . . .	82
TL::IsBaseOf< EmptyTypeList, EmptyTypeList > . . . . .	82
TL::IsBaseOf< parent, EmptyTypeList > . . . . .	83
TL::IsTypeList< T > . . . . .	84
TL::IsTypeList< TypeList< Args... > > . . . . .	84
GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< cur_children, cur_visited > . . . . .	85
GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< EmptyTypeList, cur_unvisited > . . . . .	86
GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< cur_edges, wanted_node > . . . . .	87
GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< cur_edges > . . . . .	89
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< edge_list > . . . . .	90
GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< EmptyTypeList > . . . . .	91
ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< EmptyTypeList > . . . . .	92
GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< EmptyTypeList, wanted_node > . . . . .	93
TL::Reverse< type_list >::IterateThroughElements< cur_type_list, cur_result > . . . . .	94
TL::Reverse< type_list >::IterateThroughElements< EmptyTypeList, cur_result > . . . . .	95
GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< cur_nodes > . . . . .	96
GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< EmptyTypeList > . . . . .	97
TL::Concatenate< front, back >::IterateThroughReversedFront< elements, current > . . . . .	97
TL::Concatenate< front, back >::IterateThroughReversedFront< EmptyTypeList, current > . . . . .	98
ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< current_vertexes, current > . . . . .	99

ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< EmptyTypeList, EmptyTypeList >	100
TL::MostDerived< type_list, T >	101
TL::MostDerived< EmptyTypeList, T >	102
TL::MostDerivedAndConstructible< type_list, T >	103
TL::MostDerivedAndConstructible< EmptyTypeList, T >	104
TL::NoDuplicates< type_list >	104
TL::NoDuplicates< EmptyTypeList >	105
NullType	106
PointerStructureGraph< nodes >	106
PointerStructureNode< vertex_, children_ >	108
TL::Remove< type_list, T >	109
TL::Remove< EmptyTypeList, T >	110
TL::Remove< type_list, typename type_list::Head >	111
TL::RemoveAll< type_list, T >	111
TL::RemoveAll< type_list, typename type_list::Head >	112
TL::Replace< T, ind, Arg, Args >	113
TL::Replace< T, 0, TypeList< Arg, Args... > >	113
TL::Replace< T, ind, TypeList< Arg, Args... > >	114
TL::Reverse< type_list >	115
TL::Size< type_list >	116
TL::Size< EmptyTypeList >	117
Stream< T >	118
TL::TypeAt< type_list, ind >	121
TL::TypeAt< type_list, 0 >	122
TypeList< Args >	123
TypeList< H, T... >	124
TypeList< T >	125
VertexStream< stream, graph >	126
VertexStream< EmptyTypeList, graph >	127



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

<a href="#">functor.h</a>	129
<a href="#">graph/edge.h</a>	129
<a href="#">graph/objects.h</a>	137
<a href="#">graph/stream.h</a>	138
<a href="#">graph/stream_functor_type.h</a>	138
<a href="#">graph/vertex_stream.h</a>	138
<a href="#">graph/examples/graph_examples.cpp</a>	129
<a href="#">graph/examples/vertex_stream_example.cpp</a>	130
<a href="#">graph/GLib/add_edge.h</a>	131
<a href="#">graph/GLib/dfs.h</a>	131
<a href="#">graph/GLib/find_node_by_vertex.h</a>	132
<a href="#">graph/GLib/find_path.h</a>	132
<a href="#">graph/GLib/get_reached_vertexes.h</a>	133
<a href="#">graph/GLib/map_indexes_to_vertexes.h</a>	133
<a href="#">graph/graphs/adjacency_list_graph.h</a>	133
<a href="#">graph/graphs/adjacency_matrix_graph.h</a>	133
<a href="#">graph/graphs/convert_from_edge_list.h</a>	134
<a href="#">graph/graphs/convert_from_pointer_structure.h</a>	134
<a href="#">graph/graphs/convert_graph.h</a>	134
<a href="#">graph/graphs/convert_to_adjacency_list.h</a>	135
<a href="#">graph/graphs/convert_to_pointer_structure.h</a>	135
<a href="#">graph/graphs/edge_list_graph.h</a>	136
<a href="#">graph/graphs/graph.h</a>	136
<a href="#">graph/graphs/graph_type.h</a>	136
<a href="#">graph/graphs/pointer_structure_graph.h</a>	137
<a href="#">graph/graphs/pointer_structure_node.h</a>	137
<a href="#">TL/add.h</a>	139
<a href="#">TL/concatenate.h</a>	139
<a href="#">TL/contains.h</a>	139
<a href="#">TL/contains_constructible_parent.h</a>	140
<a href="#">TL/contains_parent.h</a>	140
<a href="#">TL/find_parent_type_list.h</a>	140
<a href="#">TL/find_type_list_by_class.h</a>	141
<a href="#">TL/generate_type_lists.h</a>	141
<a href="#">TL/has_derived_and_constructible.h</a>	142

<a href="#">TL/index_of.h</a>	142
<a href="#">TL/is_base_of.h</a>	142
<a href="#">TL/is_type_list.h</a>	143
<a href="#">TL/most_derived.h</a>	143
<a href="#">TL/most_derived_and_constructible.h</a>	144
<a href="#">TL/no_duplicates.h</a>	144
<a href="#">TL/null_type.h</a>	144
<a href="#">TL/remove.h</a>	144
<a href="#">TL/replace.h</a>	145
<a href="#">TL/reverse.h</a>	145
<a href="#">TL/size.h</a>	146
<a href="#">TL/type_at.h</a>	146
<a href="#">TL/type_list.h</a>	146

## Chapter 5

# Namespace Documentation

### 5.1 GLib Namespace Reference

#### Classes

- struct [AddEdge](#)
- struct [AddEdge< ADJACENCY\\_LIST, graph, edge >](#)
- struct [DFS](#)
- struct [FindNodeByVertex](#)
- struct [FindNodeByVertex< vertex, EmptyTypeList >](#)
- struct [FindPath](#)
- struct [GetReachedVertexes](#)

### 5.2 Objects Namespace Reference

#### Classes

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

#### 5.2.1 Detailed Description

Represents class holders of different objects

### 5.3 StreamFunctorType Namespace Reference

#### Typedefs

- template<class graph >  
using [Consumer](#) = [Functor](#)< void(size\_t)>
- template<class graph >  
using [Predicate](#) = [Functor](#)< bool(size\_t)>
- template<class graph , class ResultType >  
using [Function](#) = [Functor](#)< ResultType(size\_t)>
- template<class graph , class ResultType >  
using [BinaryOperation](#) = [Functor](#)< ResultType(ResultType, size\_t)>

### 5.3.1 Detailed Description

Represents different types of functors for use in [VertexStream](#). To use it, you need to pass graph as a template parameter and an index of a vertex in this graph as an argument.

See also

[Class](#)

[Functor](#)

Package java.util.function in Java

### 5.3.2 Typedef Documentation

#### 5.3.2.1 BinaryOperation

```
template<class graph , class ResultType >  
using StreamFunctorType::BinaryOperation = typedef Functor<ResultType(ResultType, size_t)>
```

Definition at line 23 of file stream\_functor\_type.h.

#### 5.3.2.2 Consumer

```
template<class graph >  
using StreamFunctorType::Consumer = typedef Functor<void(size_t)>
```

Definition at line 14 of file stream\_functor\_type.h.

#### 5.3.2.3 Function

```
template<class graph , class ResultType >  
using StreamFunctorType::Function = typedef Functor<ResultType(size_t)>
```

Definition at line 20 of file stream\_functor\_type.h.

#### 5.3.2.4 Predicate

```
template<class graph >  
using StreamFunctorType::Predicate = typedef Functor<bool(size_t)>
```

Definition at line 17 of file stream\_functor\_type.h.



## 5.4 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 [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 [Reverse](#)
- struct [Size](#)
- struct [Size< EmptyTypeList >](#)
- struct [TypeAt](#)
- struct [TypeAt< type\\_list, 0 >](#)

### 5.4.1 Detailed Description

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



## Chapter 6

# Class Documentation

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

#### 6.1.1 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...>>

Definition at line 10 of file add.h.

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

- [TL/add.h](#)

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

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

#### Public Types

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

#### 6.2.1 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...>>

Definition at line 33 of file add.h.

## 6.2.2 Member Typedef Documentation

### 6.2.2.1 result

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

Definition at line 34 of file add.h.

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

- [TL/add.h](#)

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

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

### Public Types

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

### 6.3.1 Detailed Description

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

See also

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

Definition at line 41 of file add.h.

## 6.3.2 Member Typedef Documentation

### 6.3.2.1 result

```
template<typename T , class ... Args>
using TL::Add< T, 0, TypeList< Args... > >::result = TypeList<T, Args...>
```

Definition at line 42 of file add.h.

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

- [TL/add.h](#)

## 6.4 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`

### 6.4.1 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 <a href="#">TypeList</a>
<i>ind</i>	Number of this position
<i>TypeList&lt;Arg,Args...&gt;</i>	This <a href="#">TypeList</a>

#### Returns

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

Definition at line 19 of file add.h.

### 6.4.2 Member Typedef Documentation

#### 6.4.2.1 end

```
template<typename T , size_t ind, class Arg , class ... Args>
using TL::Add< T, ind, TypeList< Arg, Args... > >::end = typename Add< T, ind - 1, TypeList<Args...>
>::result
```

Definition at line 20 of file add.h.

### 6.4.2.2 result

```
template<typename T , size_t ind, class Arg , class ... Args>
using TL::Add< T, ind, TypeList< Arg, Args... > >::result = typename Add<Arg, 0, end>↵
::result
```

Definition at line 26 of file add.h.

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

- [TL/add.h](#)

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

### 6.5.1 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

Definition at line 20 of file add\_edge.h.

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

- [graph/GLib/add\\_edge.h](#)

## 6.6 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>()

### 6.6.1 Detailed Description

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

See also

[AddEdge](#)

Definition at line 26 of file add\_edge.h.

### 6.6.2 Member Typedef Documentation

#### 6.6.2.1 adjacent\_vertexes

```
template<class graph , class edge >
using GLib::AddEdge< ADJACENCY_LIST, graph, edge >::adjacent_vertexes = typename TL::TypeAt<typename
graph::adjacency_list_, vertex\_num>::value
```

Definition at line 28 of file add\_edge.h.

#### 6.6.2.2 new\_adjacency\_list

```
template<class graph , class edge >
using GLib::AddEdge< ADJACENCY_LIST, graph, edge >::new_adjacency_list = typename TL::Replace<new\_adjacent\_v
vertex\_num, typename graph::adjacency_list_>::result
```

Definition at line 31 of file add\_edge.h.

### 6.6.2.3 new\_adjacent\_vertexes

```
template<class graph , class edge >
using GLib::AddEdge< ADJACENCY_LIST, graph, edge >::new_adjacent_vertexes = typename TL::Add<edge,
0, adjacent_vertexes>::result
```

Definition at line 30 of file add\_edge.h.

### 6.6.2.4 result

```
template<class graph , class edge >
using GLib::AddEdge< ADJACENCY_LIST, graph, edge >::result = AdjacencyListGraph<typename
graph::vertexes_, new_adjacency_list>
```

Definition at line 32 of file add\_edge.h.

## 6.6.3 Member Data Documentation

### 6.6.3.1 vertex\_num

```
template<class graph , class edge >
constexpr static size_t GLib::AddEdge< ADJACENCY_LIST, graph, edge >::vertex_num = graph↔
::template GetVertexIndex<typename edge::from>() [static], [constexpr]
```

Definition at line 27 of file add\_edge.h.

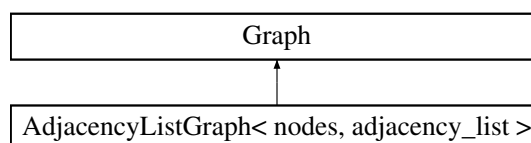
The documentation for this struct was generated from the following file:

- [graph/GLib/add\\_edge.h](#)

## 6.7 AdjacencyListGraph< nodes, adjacency\_list > Struct Template Reference

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

Inheritance diagram for AdjacencyListGraph< nodes, adjacency\_list >:





## Classes

- struct [ConvertTo](#)

## Public Types

- using [vertexes\\_](#) = nodes  
*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](#) ()

## Static Public Attributes

- constexpr static [GraphType](#) TYPE = ADJACENCY\_LIST

### 6.7.1 Detailed Description

```
template<class nodes, class adjacency_list>
struct AdjacencyListGraph< nodes, 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

See also

[Graph](#)

#### Parameters

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

Definition at line 24 of file `adjacency_list_graph.h`.

## 6.7.2 Member Typedef Documentation

### 6.7.2.1 adjacency\_list\_

```
template<class nodes , class adjacency_list >
using AdjacencyListGraph< nodes, adjacency_list >::adjacency_list_ = adjacency_list
```

[TypeList](#) of TypeLists of edges, which are grouped by starting vertex.

Definition at line 28 of file adjacency\_list\_graph.h.

### 6.7.2.2 vertexes\_

```
template<class nodes , class adjacency_list >
using AdjacencyListGraph< nodes, adjacency_list >::vertexes_ = nodes
```

[TypeList](#) of vertexes in graph.

Definition at line 27 of file adjacency\_list\_graph.h.

## 6.7.3 Member Function Documentation

### 6.7.3.1 GetVertexIndex()

```
template<class nodes , class adjacency_list >
template<typename vertex >
constexpr static size_t AdjacencyListGraph< nodes, 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](#)

Definition at line 51 of file adjacency\_list\_graph.h.

### 6.7.3.2 HasEdge()

```
template<class nodes , class adjacency_list >
template<class edge >
constexpr bool AdjacencyListGraph< nodes, 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

Definition at line 39 of file adjacency\_list\_graph.h.

## 6.7.4 Member Data Documentation

### 6.7.4.1 TYPE

```
template<class nodes , class adjacency_list >
constexpr static GraphType AdjacencyListGraph< nodes, adjacency_list >::TYPE = ADJACENCY_LIST
[static], [constexpr]
```

Definition at line 25 of file adjacency\_list\_graph.h.

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

- graph/graphs/[adjacency\\_list\\_graph.h](#)

## 6.8 AdjacencyMatrixGraph< nodes, matrix > Struct Template Reference

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

### Classes

- struct [ConvertTo](#)

### Public Types

- using [vertexes\\_](#) = nodes  
*TypeList of vertexes in graph.*
- using [matrix\\_](#) = matrix  
*TypeList of TypeLists of Edges.*

## Static Public Attributes

- constexpr static [GraphType](#) TYPE = ADJACENCY\_MATRIX

### 6.8.1 Detailed Description

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

Represents graph as an adjacency matrix. Element in row i, column j must be [Objects::Boolean](#). If it's `Boolean<false>`, then there's no edge between them. Otherwise there is.

See also

[Objects::Boolean](#)

[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

Definition at line 23 of file `adjacency_matrix_graph.h`.

### 6.8.2 Member Typedef Documentation

#### 6.8.2.1 `matrix_`

```
template<class nodes , class matrix >
using AdjacencyMatrixGraph< nodes, matrix >::matrix_ = matrix
```

[TypeList](#) of TypeLists of Edges.

Definition at line 28 of file `adjacency_matrix_graph.h`.

### 6.8.2.2 vertexes\_

```
template<class nodes , class matrix >
using AdjacencyMatrixGraph< nodes, matrix >::vertexes_ = nodes
```

[TypeList](#) of vertexes in graph.

Definition at line 27 of file adjacency\_matrix\_graph.h.

## 6.8.3 Member Data Documentation

### 6.8.3.1 TYPE

```
template<class nodes , class matrix >
constexpr static GraphType AdjacencyMatrixGraph< nodes, matrix >::TYPE = ADJACENCY_MATRIX
[static], [constexpr]
```

Definition at line 24 of file adjacency\_matrix\_graph.h.

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

- graph/graphs/[adjacency\\_matrix\\_graph.h](#)

## 6.9 Objects::Boolean< boolean > Struct Template Reference

```
#include <objects.h>
```

### Static Public Attributes

- constexpr static bool [value](#) = boolean

### 6.9.1 Detailed Description

```
template<bool boolean>
struct Objects::Boolean< boolean >
```

Definition at line 13 of file objects.h.

### 6.9.2 Member Data Documentation

### 6.9.2.1 value

```
template<bool boolean>
constexpr static bool Objects::Boolean< boolean >::value = boolean [static], [constexpr]
```

Definition at line 14 of file [objects.h](#).

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

- [graph/objects.h](#)

## 6.10 CheckContainsConstructibleParent< type\_list, T, is\_parent > Struct Template Reference

```
#include <contains_constructible_parent.h>
```

### 6.10.1 Detailed Description

```
template<class type_list, typename T, bool is_parent>
struct CheckContainsConstructibleParent< type_list, T, is_parent >
```

Definition at line 18 of file [contains\\_constructible\\_parent.h](#).

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

- [TL/contains\\_constructible\\_parent.h](#)

## 6.11 CheckContainsConstructibleParent< type\_list, T, false > Struct Template Reference

```
#include <contains_constructible_parent.h>
```

### Static Public Attributes

- constexpr static bool [result](#)

### 6.11.1 Detailed Description

```
template<class type_list, typename T>
struct CheckContainsConstructibleParent< type_list, T, false >
```

Definition at line 21 of file [contains\\_constructible\\_parent.h](#).

## 6.11.2 Member Data Documentation

### 6.11.2.1 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
```

Definition at line 22 of file contains\_constructible\_parent.h.

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

- [TL/contains\\_constructible\\_parent.h](#)

## 6.12 CheckContainsConstructibleParent< type\_list, T, true > Struct Template Reference

```
#include <contains_constructible_parent.h>
```

### Static Public Attributes

- constexpr static bool [result](#) = true

### 6.12.1 Detailed Description

```
template<class type_list, typename T>
struct CheckContainsConstructibleParent< type_list, T, true >
```

Definition at line 29 of file contains\_constructible\_parent.h.

### 6.12.2 Member Data Documentation

### 6.12.2.1 result

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

Definition at line 30 of file contains\_constructible\_parent.h.

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

- [TL/contains\\_constructible\\_parent.h](#)

## 6.13 CheckContainsParent< type\_list, T, is\_parent > Struct Template Reference

```
#include <contains_parent.h>
```

### 6.13.1 Detailed Description

```
template<class type_list, typename T, bool is_parent>
struct CheckContainsParent< type_list, T, is_parent >
```

Definition at line 19 of file contains\_parent.h.

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

- [TL/contains\\_parent.h](#)

## 6.14 CheckContainsParent< type\_list, T, false > Struct Template Reference

```
#include <contains_parent.h>
```

### Static Public Attributes

- constexpr static bool [result](#)

### 6.14.1 Detailed Description

```
template<class type_list, typename T>
struct CheckContainsParent< type_list, T, false >
```

Definition at line 22 of file contains\_parent.h.



## 6.14.2 Member Data Documentation

### 6.14.2.1 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
```

Definition at line 23 of file contains\_parent.h.

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

- [TL/contains\\_parent.h](#)

## 6.15 CheckContainsParent< type\_list, T, true > Struct Template Reference

```
#include <contains_parent.h>
```

### Static Public Attributes

- constexpr static bool [result](#) = true

### 6.15.1 Detailed Description

```
template<class type_list, typename T>
struct CheckContainsParent< type_list, T, true >
```

Definition at line 30 of file contains\_parent.h.

### 6.15.2 Member Data Documentation

### 6.15.2.1 result

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

Definition at line 31 of file contains\_parent.h.

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

- [TL/contains\\_parent.h](#)

## 6.16 CheckFindParentTypeList< contains\_class, T, type\_list, type\_lists > Struct Template Reference

```
#include <find_parent_type_list.h>
```

### Public Types

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

### 6.16.1 Detailed Description

```
template<bool contains_class, typename T, class type_list, class ... type_lists>
struct CheckFindParentTypeList< contains_class, T, type_list, type_lists >
```

Definition at line 19 of file find\_parent\_type\_list.h.

### 6.16.2 Member Typedef Documentation

#### 6.16.2.1 result

```
template<bool contains_class, typename T , class type_list , class ... type_lists>
using CheckFindParentTypeList< contains_class, T, type_list, type_lists >::result = NullType
```

Definition at line 20 of file find\_parent\_type\_list.h.

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

- [TL/find\\_parent\\_type\\_list.h](#)

## 6.17 CheckFindParentTypeList< false, T, type\_list, type\_lists... > Struct Template Reference

```
#include <find_parent_type_list.h>
```

### Public Types

- using [result](#) = typename [TL::FindParentTypeList](#)< T, type\_lists... >::[result](#)

#### 6.17.1 Detailed Description

```
template<typename T, class type_list, class ... type_lists>
struct CheckFindParentTypeList< false, T, type_list, type_lists... >
```

Definition at line 29 of file `find_parent_type_list.h`.

#### 6.17.2 Member Typedef Documentation

##### 6.17.2.1 result

```
template<typename T , class type_list , class ... type_lists>
using CheckFindParentTypeList< false, T, type_list, type_lists... >::result = typename TL::FindParentTypeList
type_lists...>::result
```

Definition at line 30 of file `find_parent_type_list.h`.

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

- [TL/find\\_parent\\_type\\_list.h](#)

## 6.18 CheckFindParentTypeList< true, T, type\_list, type\_lists... > Struct Template Reference

```
#include <find_parent_type_list.h>
```

### Public Types

- using [result](#) = type\_list

### 6.18.1 Detailed Description

```
template<typename T, class type_list, class ... type_lists>
struct CheckFindParentTypeList< true, T, type_list, type_lists... >
```

Definition at line 24 of file find\_parent\_type\_list.h.

### 6.18.2 Member Typedef Documentation

#### 6.18.2.1 result

```
template<typename T , class type_list , class ... type_lists>
using CheckFindParentTypeList< true, T, type_list, type_lists... >::result = type_list
```

Definition at line 25 of file find\_parent\_type\_list.h.

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

- [TL/find\\_parent\\_type\\_list.h](#)

## 6.19 CheckFindTypeListByClass< contains\_class, T, type\_list, type\_lists > Struct Template Reference

```
#include <find_type_list_by_class.h>
```

### Public Types

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

#### 6.19.1 Detailed Description

```
template<bool contains_class, typename T, class type_list, class ... type_lists>
struct CheckFindTypeListByClass< contains_class, T, type_list, type_lists >
```

Definition at line 19 of file find\_type\_list\_by\_class.h.

#### 6.19.2 Member Typedef Documentation

### 6.19.2.1 result

```
template<bool contains_class, typename T , class type_list , class ... type_lists>
using CheckFindTypeListByClass< contains_class, T, type_list, type_lists >::result = NullType
```

Definition at line 20 of file find\_type\_list\_by\_class.h.

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

- [TL/find\\_type\\_list\\_by\\_class.h](#)

## 6.20 CheckFindTypeListByClass< false, T, type\_list, type\_lists... > Struct Template Reference

```
#include <find_type_list_by_class.h>
```

### Public Types

- using [result](#) = typename [TL::FindTypeListByClass](#)< T, type\_lists... >::result

### 6.20.1 Detailed Description

```
template<typename T, class type_list, class ... type_lists>
struct CheckFindTypeListByClass< false, T, type_list, type_lists... >
```

Definition at line 29 of file find\_type\_list\_by\_class.h.

### 6.20.2 Member Typedef Documentation

#### 6.20.2.1 result

```
template<typename T , class type_list , class ... type_lists>
using CheckFindTypeListByClass< false, T, type_list, type_lists... >::result = typename
TL::FindTypeListByClass<T, type_lists...>::result
```

Definition at line 30 of file find\_type\_list\_by\_class.h.

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

- [TL/find\\_type\\_list\\_by\\_class.h](#)

## 6.21 CheckFindTypeListByClass< true, T, type\_list, type\_lists... > Struct Template Reference

```
#include <find_type_list_by_class.h>
```

### Public Types

- using [result](#) = type\_list

#### 6.21.1 Detailed Description

```
template<typename T, class type_list, class ... type_lists>  
struct CheckFindTypeListByClass< true, T, type_list, type_lists... >
```

Definition at line 24 of file `find_type_list_by_class.h`.

#### 6.21.2 Member Typedef Documentation

##### 6.21.2.1 result

```
template<typename T , class type_list , class ... type_lists>  
using CheckFindTypeListByClass< true, T, type_list, type_lists... >::result = type_list
```

Definition at line 25 of file `find_type_list_by_class.h`.

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

- [TL/find\\_type\\_list\\_by\\_class.h](#)

## 6.22 CheckHasDerivedAndConstructible< type\_list, T, is\_head\_parent\_of\_T > Struct Template Reference

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

#### 6.22.1 Detailed Description

```
template<class type_list, typename T, bool is_head_parent_of_T>  
struct CheckHasDerivedAndConstructible< type_list, T, is_head_parent_of_T >
```

Definition at line 19 of file `has_derived_and_constructible.h`.

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

- [TL/has\\_derived\\_and\\_constructible.h](#)

## 6.23 CheckHasDerivedAndConstructible< type\_list, T, false > Struct Template Reference

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

### Static Public Attributes

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

### 6.23.1 Detailed Description

```
template<class type_list, typename T>
struct CheckHasDerivedAndConstructible< type_list, T, false >
```

Definition at line 27 of file [has\\_derived\\_and\\_constructible.h](#).

### 6.23.2 Member Data Documentation

#### 6.23.2.1 result

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

Definition at line 28 of file [has\\_derived\\_and\\_constructible.h](#).

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

- [TL/has\\_derived\\_and\\_constructible.h](#)

## 6.24 CheckHasDerivedAndConstructible< type\_list, T, true > Struct Template Reference

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

### Static Public Attributes

- constexpr static bool [result](#) = true

### 6.24.1 Detailed Description

```
template<class type_list, typename T>
struct CheckHasDerivedAndConstructible< type_list, T, true >
```

Definition at line 22 of file `has_derived_and_constructible.h`.

### 6.24.2 Member Data Documentation

#### 6.24.2.1 result

```
template<class type_list , typename T >
constexpr static bool CheckHasDerivedAndConstructible< type_list, T, true >::result = true
[static], [constexpr]
```

Definition at line 23 of file `has_derived_and_constructible.h`.

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

- [TL/has\\_derived\\_and\\_constructible.h](#)

## 6.25 CheckIsBaseOf< has\_parent, parent, derived > Struct Template Reference

```
#include <is_base_of.h>
```

### 6.25.1 Detailed Description

```
template<bool has_parent, class parent, class derived>
struct CheckIsBaseOf< has_parent, parent, derived >
```

Definition at line 21 of file `is_base_of.h`.

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

- [TL/is\\_base\\_of.h](#)

## 6.26 CheckIsBaseOf< false, parent, derived > Struct Template Reference

```
#include <is_base_of.h>
```



## Static Public Attributes

- constexpr static bool [result](#) = false

### 6.26.1 Detailed Description

```
template<class parent, class derived>
struct CheckIsBaseOf< false, parent, derived >
```

Definition at line 24 of file `is_base_of.h`.

### 6.26.2 Member Data Documentation

#### 6.26.2.1 result

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

Definition at line 25 of file `is_base_of.h`.

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

- [TL/is\\_base\\_of.h](#)

## 6.27 CheckIsBaseOf< true, parent, derived > Struct Template Reference

```
#include <is_base_of.h>
```

## Static Public Attributes

- constexpr static bool [result](#)

### 6.27.1 Detailed Description

```
template<class parent, class derived>
struct CheckIsBaseOf< true, parent, derived >
```

Definition at line 29 of file `is_base_of.h`.

### 6.27.2 Member Data Documentation

### 6.27.2.1 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
```

Definition at line 30 of file `is_base_of.h`.

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

- [TL/is\\_base\\_of.h](#)

## 6.28 CheckMostDerived< type\_list, T, is\_head\_parent\_of\_T > Struct Template Reference

```
#include <most_derived.h>
```

### Public Types

- using `result` = `NullType`

### 6.28.1 Detailed Description

```
template<class type_list, typename T, bool is_head_parent_of_T>
struct CheckMostDerived< type_list, T, is_head_parent_of_T >
```

Definition at line 19 of file `most_derived.h`.

### 6.28.2 Member Typedef Documentation

#### 6.28.2.1 result

```
template<class type_list , typename T , bool is_head_parent_of_T>
using CheckMostDerived< type_list, T, is_head_parent_of_T >::result = NullType
```

Definition at line 20 of file `most_derived.h`.

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

- [TL/most\\_derived.h](#)

## 6.29 CheckMostDerived< type\_list, T, false > Struct Template Reference

```
#include <most_derived.h>
```

### Public Types

- using [result](#) = typename [TL::MostDerived](#)< typename type\_list::Tail, T >::[result](#)

#### 6.29.1 Detailed Description

```
template<class type_list, typename T>
struct CheckMostDerived< type_list, T, false >
```

Definition at line 29 of file `most_derived.h`.

#### 6.29.2 Member Typedef Documentation

##### 6.29.2.1 result

```
template<class type_list , typename T >
using CheckMostDerived< type_list, T, false >::result = typename TL::MostDerived<typename
type_list::Tail, T>::result
```

Definition at line 30 of file `most_derived.h`.

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

- [TL/most\\_derived.h](#)

## 6.30 CheckMostDerived< type\_list, T, true > Struct Template Reference

```
#include <most_derived.h>
```

### Public Types

- using [result](#) = typename [TL::MostDerived](#)< typename type\_list::Tail, typename type\_list::Head >::[result](#)

#### 6.30.1 Detailed Description

```
template<class type_list, typename T>
struct CheckMostDerived< type_list, T, true >
```

Definition at line 24 of file `most_derived.h`.

## 6.30.2 Member Typedef Documentation

### 6.30.2.1 result

```
template<class type_list , typename T >
using CheckMostDerived< type_list, T, true >::result = typename TL::MostDerived<typename
type_list::Tail, typename type_list::Head>::result
```

Definition at line 25 of file most\_derived.h.

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

- [TL/most\\_derived.h](#)

## 6.31 CheckMostDerivedAndConstructible< type\_list, T, is\_head\_parent\_of\_T > Struct Template Reference

```
#include <most_derived_and_constructible.h>
```

### 6.31.1 Detailed Description

```
template<class type_list, typename T, bool is_head_parent_of_T>
struct CheckMostDerivedAndConstructible< type_list, T, is_head_parent_of_T >
```

Definition at line 19 of file most\_derived\_and\_constructible.h.

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

- [TL/most\\_derived\\_and\\_constructible.h](#)

## 6.32 CheckMostDerivedAndConstructible< type\_list, T, false > Struct Template Reference

```
#include <most_derived_and_constructible.h>
```

### Public Types

- using [result](#) = typename [TL::MostDerivedAndConstructible](#)< typename type\_list::Tail, T >::result

### 6.32.1 Detailed Description

```
template<class type_list, typename T>
struct CheckMostDerivedAndConstructible< type_list, T, false >
```

Definition at line 27 of file `most_derived_and_constructible.h`.

### 6.32.2 Member Typedef Documentation

#### 6.32.2.1 result

```
template<class type_list , typename T >
using CheckMostDerivedAndConstructible< type_list, T, false >::result = typename TL::MostDerivedAndConstructible<
type_list::Tail, T>::result
```

Definition at line 28 of file `most_derived_and_constructible.h`.

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

- [TL/most\\_derived\\_and\\_constructible.h](#)

## 6.33 CheckMostDerivedAndConstructible< type\_list, T, true > Struct Template Reference

```
#include <most_derived_and_constructible.h>
```

### Public Types

- using `result` = `typename TL::MostDerivedAndConstructible< typename type_list::Tail, typename type_list::Head >::result`

### 6.33.1 Detailed Description

```
template<class type_list, typename T>
struct CheckMostDerivedAndConstructible< type_list, T, true >
```

Definition at line 22 of file `most_derived_and_constructible.h`.

### 6.33.2 Member Typedef Documentation

### 6.33.2.1 result

```
template<class type_list , typename T >
using CheckMostDerivedAndConstructible< type_list, T, true >::result = typename TL::MostDerivedAndConstructible<
type_list::Tail, typename type_list::Head>::result
```

Definition at line 23 of file `most_derived_and_constructible.h`.

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

- [TL/most\\_derived\\_and\\_constructible.h](#)

## 6.34 TL::Concatenate< front, back > Struct Template Reference

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

### Classes

- struct [IterateThroughReversedFront](#)
- struct [IterateThroughReversedFront< EmptyTypeList, current >](#)

### Public Types

- using [reversed\\_front](#) = typename [Reverse](#)< front >::result
- using [result](#) = typename [IterateThroughReversedFront](#)< [reversed\\_front](#), back >::result

### 6.34.1 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](#)

Definition at line 17 of file `concatenate.h`.

### 6.34.2 Member Typedef Documentation

### 6.34.2.1 result

```
template<class front , class back >
using TL::Concatenate< front, back >::result = typename IterateThroughReversedFront<reversed_front,
back>::result
```

Definition at line 42 of file concatenate.h.

### 6.34.2.2 reversed\_front

```
template<class front , class back >
using TL::Concatenate< front, back >::reversed_front = typename Reverse<front>::result
```

Definition at line 21 of file concatenate.h.

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

- TL/[concatenate.h](#)

## 6.35 TL::Contains< type\_list, T > Struct Template Reference

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

### Static Public Attributes

- constexpr static bool [result](#) = [IndexOf](#)<type\_list, T>::value >= 0

### 6.35.1 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

Definition at line 14 of file contains.h.

## 6.35.2 Member Data Documentation

### 6.35.2.1 result

```
template<class type_list , typename T >
constexpr static bool TL::Contains< type_list, T >::result = IndexOf<type_list, T>::value >=
0 [static], [constexpr]
```

Definition at line 15 of file contains.h.

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

- [TL/contains.h](#)

## 6.36 TL::ContainsConstructibleParent< type\_list, T > Struct Template Reference

```
#include <contains_constructible_parent.h>
```

### Static Public Attributes

- constexpr static bool [result](#)

### 6.36.1 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

Definition at line 35 of file contains\_constructible\_parent.h.

## 6.36.2 Member Data Documentation



### 6.36.2.1 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
```

Definition at line 36 of file contains\_constructible\_parent.h.

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

- [TL/contains\\_constructible\\_parent.h](#)

## 6.37 TL::ContainsConstructibleParent< EmptyTypeList, T > Struct Template Reference

```
#include <contains_constructible_parent.h>
```

### Static Public Attributes

- constexpr static bool [result](#) = false

### 6.37.1 Detailed Description

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

Definition at line 45 of file contains\_constructible\_parent.h.

### 6.37.2 Member Data Documentation

#### 6.37.2.1 result

```
template<typename T >
constexpr static bool TL::ContainsConstructibleParent< EmptyTypeList, T >::result = false
[static], [constexpr]
```

Definition at line 46 of file contains\_constructible\_parent.h.

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

- [TL/contains\\_constructible\\_parent.h](#)

## 6.38 TL::ContainsParent< type\_list, T > Struct Template Reference

```
#include <contains_parent.h>
```

### Static Public Attributes

- constexpr static bool [result](#)

### 6.38.1 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

Definition at line 36 of file contains\_parent.h.

### 6.38.2 Member Data Documentation

#### 6.38.2.1 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
```

Definition at line 37 of file contains\_parent.h.

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

- TL/[contains\\_parent.h](#)

## 6.39 TL::ContainsParent< EmptyTypeList, T > Struct Template Reference

```
#include <contains_parent.h>
```

### Static Public Attributes

- constexpr static bool [result](#) = false

### 6.39.1 Detailed Description

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

Definition at line 45 of file contains\_parent.h.

### 6.39.2 Member Data Documentation

#### 6.39.2.1 result

```
template<typename T >  
constexpr static bool TL::ContainsParent< EmptyTypeList, T >::result = false [static], [constexpr]
```

Definition at line 46 of file contains\_parent.h.

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

- [TL/contains\\_parent.h](#)

## 6.40 ConvertGraph< From, To, graph > Struct Template Reference

### 6.40.1 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. Before conversion, don't forget to include file of a required implementation. 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.

Definition at line 17 of file `convert_graph.h`.

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

- `graph/graphs/convert_graph.h`

## 6.41 **ConvertGraph< ADJACENCY\_LIST, POINTER\_STRUCTURE, graph > Struct Template Reference**

```
#include <convert_to_pointer_structure.h>
```

**Classes**

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

**Public Types**

- using [result](#) = [PointerStructureGraph](#)< typename [MakePointerStructureGraph](#)< typename [graph::vertexes](#)↵  
\_, typename [graph::adjacency\\_list\\_](#) >::[result](#) >

### 6.41.1 Detailed Description

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

See also

[ConvertGraph](#)

Definition at line 24 of file `convert_to_pointer_structure.h`.

### 6.41.2 Member Typedef Documentation

### 6.41.2.1 result

```
template<class graph >
using ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::result = PointerStructureGraph<
typename MakePointerStructureGraph< typename graph::vertexes_, typename graph::adjacency_<
list_ >::result >
```

Definition at line 49 of file convert\_to\_pointer\_structure.h.

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

- graph/graphs/convert\_to\_pointer\_structure.h

## 6.42 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
- using [result](#) = typename IterateThroughEdges< typename graph::edge\_list\_ >::result

### 6.42.1 Detailed Description

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

See also

[ConvertGraph](#)

Definition at line 21 of file convert\_from\_edge\_list.h.

### 6.42.2 Member Typedef Documentation

**6.42.2.1 result [1/2]**

```
template<class graph >
using ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::result = typename IterateThrough↔
Edges<typename graph::edge_list_>::result
```

Definition at line 41 of file convert\_from\_edge\_list.h.

**6.42.2.2 result [2/2]**

```
template<class graph >
using ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::result = typename IterateThrough↔
Edges<typename graph::edge_list_>::result
```

Definition at line 41 of file convert\_to\_adjacency\_list.h.

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

- graph/graphs/convert\_from\_edge\_list.h
- graph/graphs/convert\_to\_adjacency\_list.h

## 6.43 ConvertGraph< EDGE\_LIST, POINTER\_STRUCTURE, graph > Struct Template Reference

```
#include <convert_to_pointer_structure.h>
```

**Public Types**

- using adjacency\_list = typename ConvertGraph< EDGE\_LIST, ADJACENCY\_LIST, graph >::result
- using result = typename ConvertGraph< ADJACENCY\_LIST, POINTER\_STRUCTURE, adjacency\_list >↔::result

**6.43.1 Detailed Description**

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

See also

[ConvertGraph](#)

Definition at line 61 of file convert\_to\_pointer\_structure.h.

**6.43.2 Member Typedef Documentation**

### 6.43.2.1 adjacency\_list

```
template<class graph >
using ConvertGraph< EDGE_LIST, POINTER_STRUCTURE, graph >::adjacency_list = typename ConvertGraph<EDGE_LIST,
ADJACENCY_LIST, graph>::result
```

Definition at line 63 of file convert\_to\_pointer\_structure.h.

### 6.43.2.2 result

```
template<class graph >
using ConvertGraph< EDGE_LIST, POINTER_STRUCTURE, graph >::result = typename ConvertGraph<ADJACENCY_LIST,
POINTER_STRUCTURE, adjacency_list>::result
```

Definition at line 64 of file convert\_to\_pointer\_structure.h.

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

- graph/graphs/[convert\\_to\\_pointer\\_structure.h](#)

## 6.44 ConvertGraph< POINTER\_STRUCTURE, EDGE\_LIST, graph > Struct Template Reference

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

### 6.44.1 Detailed Description

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

See also

[ConvertGraph](#)

Definition at line 20 of file convert\_from\_pointer\_structure.h.

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

- graph/graphs/[convert\\_from\\_pointer\\_structure.h](#)

## 6.45 ConvertGraph< type, type, graph > Struct Template Reference

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

## Public Types

- using [result](#) = graph

### 6.45.1 Detailed Description

```
template<GraphType type, class graph>
struct ConvertGraph< type, type, graph >
```

Definition at line 20 of file `convert_graph.h`.

### 6.45.2 Member Typedef Documentation

#### 6.45.2.1 result

```
template<GraphType type, class graph >
using ConvertGraph< type, type, graph >::result = graph
```

Definition at line 22 of file `convert_graph.h`.

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

- `graph/graphs/convert_graph.h`

## 6.46 `EdgeListGraph< nodes, edge_list >::ConvertTo< type > Struct Template Reference`

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

## Public Types

- using [result](#) = typename [ConvertGraph](#)< [TYPE](#), type, [EdgeListGraph](#)< [vertexes\\_](#), [edge\\_list\\_](#) > >::[result](#)

### 6.46.1 Detailed Description

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

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

Definition at line 35 of file edge\_list\_graph.h.

## 6.46.2 Member Typedef Documentation

### 6.46.2.1 result

```
template<class nodes , class edge_list >
template<GraphType type>
using EdgeListGraph< nodes, edge_list >::ConvertTo< type >::result = typename ConvertGraph<
TYPE, type, EdgeListGraph<vertexes_, edge_list_> >::result
```

Definition at line 36 of file edge\_list\_graph.h.

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

- graph/graphs/[edge\\_list\\_graph.h](#)

## 6.47 PointerStructureGraph< nodes >::ConvertTo< type > Struct Template Reference

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

### Public Types

- using [result](#) = typename [ConvertGraph](#)< [TYPE](#), type, [PointerStructureGraph](#)< nodes > >::result

### 6.47.1 Detailed Description

```
template<class nodes>
template<GraphType type>
struct PointerStructureGraph< nodes >::ConvertTo< type >
```

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

Definition at line 34 of file `pointer_structure_graph.h`.

## 6.47.2 Member Typedef Documentation

### 6.47.2.1 result

```
template<class nodes >
template<GraphType type>
using PointerStructureGraph< nodes >::ConvertTo< type >::result = typename ConvertGraph<
TYPE, type, PointerStructureGraph<nodes> >::result
```

Definition at line 35 of file `pointer_structure_graph.h`.

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

- [graph/graphs/pointer\\_structure\\_graph.h](#)

## 6.48 AdjacencyListGraph< nodes, adjacency\_list >::ConvertTo< type > Struct Template Reference

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

### Public Types

- using `result` = `typename ConvertGraph< TYPE, type, AdjacencyListGraph< vertexes_, adjacency_list_ >::result`

### 6.48.1 Detailed Description

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

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

Definition at line 63 of file adjacency\_list\_graph.h.

## 6.48.2 Member Typedef Documentation

### 6.48.2.1 result

```
template<class nodes , class adjacency_list >
template<GraphType type>
using AdjacencyListGraph< nodes, adjacency_list >::ConvertTo< type >::result = typename ConvertGraph<
TYPE, type, AdjacencyListGraph<vertexes_, adjacency_list_> >::result
```

Definition at line 64 of file adjacency\_list\_graph.h.

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

- graph/graphs/[adjacency\\_list\\_graph.h](#)

## 6.49 AdjacencyMatrixGraph< nodes, matrix >::ConvertTo< type > Struct Template Reference

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

### Public Types

- using [result](#) = typename [ConvertGraph](#)< [TYPE](#), type, [AdjacencyMatrixGraph](#)< nodes, matrix > >::result

### 6.49.1 Detailed Description

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

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

Definition at line 37 of file adjacency\_matrix\_graph.h.

## 6.49.2 Member Typedef Documentation

### 6.49.2.1 result

```
template<class nodes , class matrix >
template<GraphType type>
using AdjacencyMatrixGraph< nodes, matrix >::ConvertTo< type >::result = typename ConvertGraph<
TYPE, type, AdjacencyMatrixGraph<nodes, matrix> >::result
```

Definition at line 38 of file adjacency\_matrix\_graph.h.

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

- graph/graphs/[adjacency\\_matrix\\_graph.h](#)

## 6.50 GLib::DFS< cur\_node, graph, visited\_nodes > Struct Template Reference

```
#include <dfs.h>
```

### Classes

- struct [IterateThroughChildren](#)
- struct [IterateThroughChildren< EmptyTypeList, cur\\_unvisited >](#)

### Public Types

- using [upd\\_visited](#) = typename [TL::Add< cur\\_node, 0, visited\\_nodes >::result](#)
- using [iterate\\_through\\_children](#) = [IterateThroughChildren< typename cur\\_node::children, upd\\_visited >](#)
- using [new\\_visited](#) = typename [iterate\\_through\\_children::new\\_visited](#)
- using [result](#) = typename [iterate\\_through\\_children::result](#)

### 6.50.1 Detailed Description

```
template<class cur_node, class graph, class visited_nodes = EmptyTypeList>
struct GLib::DFS< cur_node, graph, visited_nodes >
```

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>cur_nod</i>	Template parameter, starting node in <a href="#">DFS</a> .
<i>graph</i>	<a href="#">Graph</a> , where <a href="#">DFS</a> should be performed.
<i>visited_nodes</i>	Optional template parameter, nodes that are not allowed to be visited.

## Returns

Parameter result, [TypeList](#) of visited edges in chronological order. Also returns parameter new\_visited as a side effect, which is a [TypeList](#) of visited nodes.

Definition at line 22 of file dfs.h.

## 6.50.2 Member Typedef Documentation

### 6.50.2.1 iterate\_through\_children

```
template<class cur_node , class graph , class visited_nodes = EmptyTypeList>
using GLib::DFS< cur_node, graph, visited_nodes >::iterate_through_children = IterateThroughChildren<
typename cur_node::children, upd_visited >
```

Definition at line 67 of file dfs.h.

### 6.50.2.2 new\_visited

```
template<class cur_node , class graph , class visited_nodes = EmptyTypeList>
using GLib::DFS< cur_node, graph, visited_nodes >::new_visited = typename iterate_through_children::new_visit
```

Definition at line 71 of file dfs.h.

### 6.50.2.3 result

```
template<class cur_node , class graph , class visited_nodes = EmptyTypeList>
using GLib::DFS< cur_node, graph, visited_nodes >::result = typename iterate_through_children::result
```

Definition at line 72 of file dfs.h.

#### 6.50.2.4 upd\_visited

```
template<class cur_node , class graph , class visited_nodes = EmptyTypeList>
using GLib::DFS< cur_node, graph, visited_nodes >::upd_visited = typename TL::Add<cur_node,
0, visited_nodes>::result
```

Definition at line 25 of file dfs.h.

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

- graph/GLib/dfs.h

## 6.51 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.*

#### 6.51.1 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> <sub>↔</sub> _	Template parameter, starting vertex of an edge
<i>to</i> _	Template parameter, ending vertex of an edge
<i>weight</i> <sub>↔</sub> _	Template parameter, additional property of an edge

Definition at line 12 of file edge.h.

#### 6.51.2 Member Typedef Documentation

### 6.51.2.1 from

```
template<typename from_ , typename to_ , typename weight_ = NullType>
using Edge< from_ , to_ , weight_ >::from = from_
```

Starting vertex of an edge.

Definition at line 13 of file edge.h.

### 6.51.2.2 to

```
template<typename from_ , typename to_ , typename weight_ = NullType>
using Edge< from_ , to_ , weight_ >::to = to_
```

Ending vertex of an edge.

Definition at line 14 of file edge.h.

### 6.51.2.3 weight

```
template<typename from_ , typename to_ , typename weight_ = NullType>
using Edge< from_ , to_ , weight_ >::weight = weight_
```

Additional property of an edge.

Definition at line 15 of file edge.h.

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

- graph/[edge.h](#)

## 6.52 EdgeListGraph< nodes, edge\_list > Struct Template Reference

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

### Classes

- struct [ConvertTo](#)

### Public Types

- using [vertexes\\_](#) = nodes  
*TypeList of vertexes in graph.*
- using [edge\\_list\\_](#) = edge\_list  
*TypeList of edges.*

## Static Public Attributes

- constexpr static [GraphType](#) TYPE = EDGE\_LIST

### 6.52.1 Detailed Description

```
template<class nodes, class edge_list>
struct EdgeListGraph< nodes, 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, <a href="#">TypeList</a> of <a href="#">Edge</a>

Returns

Parameter result, resulting graph

Definition at line 20 of file `edge_list_graph.h`.

### 6.52.2 Member Typedef Documentation

#### 6.52.2.1 `edge_list_`

```
template<class nodes , class edge_list >
using EdgeListGraph< nodes, edge_list >::edge_list_ = edge_list
```

[TypeList](#) of edges.

Definition at line 26 of file `edge_list_graph.h`.

#### 6.52.2.2 `vertexes_`

```
template<class nodes , class edge_list >
using EdgeListGraph< nodes, edge_list >::vertexes_ = nodes
```

[TypeList](#) of vertexes in graph.

Definition at line 25 of file `edge_list_graph.h`.



## 6.52.3 Member Data Documentation

### 6.52.3.1 TYPE

```
template<class nodes , class edge_list >
constexpr static GraphType EdgeListGraph< nodes, edge_list >::TYPE = EDGE_LIST [static],
[constexpr]
```

Definition at line 21 of file edge\_list\_graph.h.

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

- graph/graphs/edge\_list\_graph.h

## 6.53 GLib::FindNodeByVertex< vertex, graph > Struct Template Reference

```
#include <find_node_by_vertex.h>
```

### Classes

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

### Public Types

- using [result](#) = typename [IterateThroughNodes](#)< typename graph::nodes\_ >::result

### 6.53.1 Detailed Description

```
template<typename vertex, class graph>
struct GLib::FindNodeByVertex< vertex, graph >
```

Finds node corresponding to this vertex.

#### Parameters

<i>vertex</i>	Template parameter, vertex, node of which to find.
<i>graph</i>	Template parameter, graph that should be passed.

### Returns

Parameter result, required node if found, [NullType](#) otherwise.

Definition at line 13 of file `find_node_by_vertex.h`.

## 6.53.2 Member Typedef Documentation

### 6.53.2.1 result

```
template<typename vertex , class graph >
using GLib::FindNodeByVertex< vertex, graph >::result = typename IterateThroughNodes<typename
graph::nodes_>::result
```

Definition at line 31 of file `find_node_by_vertex.h`.

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

- `graph/GLib/find_node_by_vertex.h`

## 6.54 GLib::FindNodeByVertex< vertex, EmptyTypeList > Struct Template Reference

```
#include <find_node_by_vertex.h>
```

### Public Types

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

### 6.54.1 Detailed Description

```
template<typename vertex>
struct GLib::FindNodeByVertex< vertex, EmptyTypeList >
```

Definition at line 35 of file `find_node_by_vertex.h`.

## 6.54.2 Member Typedef Documentation

### 6.54.2.1 result

```
template<typename vertex >
using GLib::FindNodeByVertex< vertex, EmptyTypeList >::result = NullType
```

Definition at line 36 of file find\_node\_by\_vertex.h.

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

- graph/GLib/find\_node\_by\_vertex.h

## 6.55 TL::FindParentTypeList< T, type\_list, type\_lists > Struct Template Reference

```
#include <find_parent_type_list.h>
```

### Public Types

- using [result](#) = typename [CheckFindParentTypeList](#)< [TL::IsBaseOf](#)< type\_list, T >::result, T, type\_list, type↵\_lists... >::result

### 6.55.1 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 <a href="#">TypeList</a> 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

Definition at line 35 of file find\_parent\_type\_list.h.

### 6.55.2 Member Typedef Documentation

### 6.55.2.1 result

```
template<typename T , class type_list , class ... type_lists>
using TL::FindParentTypeList< T, type_list, type_lists >::result = typename CheckFindParentTypeList<
TL::IsBaseOf<type_list, T>::result, T, type_list, type_lists... >::result
```

Definition at line 36 of file find\_parent\_type\_list.h.

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

- [TL/find\\_parent\\_type\\_list.h](#)

## 6.56 GLib::FindPath< graph\_raw, start, finish > Struct Template Reference

```
#include <find_path.h>
```

### Classes

- struct [IterateThroughEdges](#)
- struct [IterateThroughEdges< EmptyTypeList, wanted\\_node >](#)

### Public Types

- using [graph](#) = typename [ConvertGraph](#)< graph\_raw::TYPE, [POINTER\\_STRUCTURE](#), graph\_raw >::result
- using [start\\_node](#) = typename [FindNodeByVertex](#)< start, [graph](#) >::result
- using [finish\\_node](#) = typename [FindNodeByVertex](#)< finish, [graph](#) >::result
- using [dfs\\_search](#) = typename [DFS](#)< [start\\_node](#), [graph](#) >::result
- using [reversed](#) = typename [TL::Reverse](#)< [dfs\\_search](#) >::result
- using [iterate\\_through\\_edges](#) = [IterateThroughEdges](#)< [reversed](#), [finish\\_node](#) >
- using [reversed\\_path](#) = typename [iterate\\_through\\_edges](#)::path
- using [reversed\\_weights](#) = typename [iterate\\_through\\_edges](#)::weights
- using [path](#) = typename [TL::Add](#)< start, 0, typename [TL::Reverse](#)< [reversed\\_path](#) >::result >::result
- using [weights](#) = typename [TL::Reverse](#)< [reversed\\_weights](#) >::result

### 6.56.1 Detailed Description

```
template<class graph_raw, typename start, typename finish>
struct GLib::FindPath< graph_raw, start, finish >
```

Finds path in graph between vertexes start and finish.

See also

[DFS](#)

**Parameters**

<i>graph</i>	Template parameter
<i>start</i>	Template parameter
<i>finish</i>	Template parameter

**Returns**

Two parameters: path and weights. "path" is a [TypeList](#) of vertexes that make this path. "weights" is a [TypeList](#) of weights, that were on the edges in this path. If there's no path, path and weights are EmptyTypeList.

Definition at line 26 of file find\_path.h.

**6.56.2 Member Typedef Documentation****6.56.2.1 dfs\_search**

```
template<class graph_raw , typename start , typename finish >
using GLib::FindPath< graph_raw, start, finish >::dfs_search = typename DFS<start_node, graph>↵
::result
```

Definition at line 32 of file find\_path.h.

**6.56.2.2 finish\_node**

```
template<class graph_raw , typename start , typename finish >
using GLib::FindPath< graph_raw, start, finish >::finish_node = typename FindNodeByVertex<finish,
graph>::result
```

Definition at line 30 of file find\_path.h.

**6.56.2.3 graph**

```
template<class graph_raw , typename start , typename finish >
using GLib::FindPath< graph_raw, start, finish >::graph = typename ConvertGraph<graph_raw::↵
TYPE, POINTER_STRUCTURE, graph_raw>::result
```

Definition at line 27 of file find\_path.h.

#### 6.56.2.4 iterate\_through\_edges

```
template<class graph_raw , typename start , typename finish >
using GLib::FindPath< graph_raw, start, finish >::iterate_through_edges = IterateThroughEdges<reversed,
finish_node>
```

Definition at line 78 of file find\_path.h.

#### 6.56.2.5 path

```
template<class graph_raw , typename start , typename finish >
using GLib::FindPath< graph_raw, start, finish >::path = typename TL::Add< start, 0, typename
TL::Reverse<reversed_path>::result >::result
```

Definition at line 82 of file find\_path.h.

#### 6.56.2.6 reversed

```
template<class graph_raw , typename start , typename finish >
using GLib::FindPath< graph_raw, start, finish >::reversed = typename TL::Reverse<dfs_search>↔
::result
```

Definition at line 33 of file find\_path.h.

#### 6.56.2.7 reversed\_path

```
template<class graph_raw , typename start , typename finish >
using GLib::FindPath< graph_raw, start, finish >::reversed_path = typename iterate_through_edges::path
```

Definition at line 79 of file find\_path.h.

#### 6.56.2.8 reversed\_weights

```
template<class graph_raw , typename start , typename finish >
using GLib::FindPath< graph_raw, start, finish >::reversed_weights = typename iterate_through_edges::weights
```

Definition at line 80 of file find\_path.h.

### 6.56.2.9 start\_node

```
template<class graph_raw , typename start , typename finish >
using GLib::FindPath< graph_raw, start, finish >::start_node = typename FindNodeByVertex<start,
graph>::result
```

Definition at line 29 of file find\_path.h.

### 6.56.2.10 weights

```
template<class graph_raw , typename start , typename finish >
using GLib::FindPath< graph_raw, start, finish >::weights = typename TL::Reverse<reversed_weights>←
::result
```

Definition at line 87 of file find\_path.h.

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

- graph/GLib/find\_path.h

## 6.57 TL::FindTypeListByClass< T, type\_list, type\_lists > Struct Template Reference

```
#include <find_type_list_by_class.h>
```

### Public Types

- using [result](#) = typename [CheckFindTypeListByClass](#)< [TL::Contains](#)< type\_list, T >::result, T, type\_list, type\_lists... >::result

### 6.57.1 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 <a href="#">TypeList</a> among other TypeLists
<i>...type_lists</i>	Template parameter, other TypeLists to check

**Returns**

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

Definition at line 35 of file `find_type_list_by_class.h`.

**6.57.2 Member Typedef Documentation****6.57.2.1 result**

```
template<typename T , class type_list , class ... type_lists>
using TL::FindTypeListByClass< T, type_list, type_lists >::result = typename CheckFindTypeListByClass<
TL::Contains<type_list, T>::result, T, type_list, type_lists... >::result
```

Definition at line 36 of file `find_type_list_by_class.h`.

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

- [TL/find\\_type\\_list\\_by\\_class.h](#)

**6.58 Functor< ResultType, ArgTypes > Class Template Reference****6.58.1 Detailed Description**

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

Definition at line 7 of file `functor.h`.

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

- [functor.h](#)

**6.59 Functor< ResultType(ArgTypes...)> Class Template Reference**

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

**Public Member Functions**

- [Functor](#) ()=default
- `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)

**6.59.1 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

Definition at line 15 of file functor.h.

## 6.59.2 Constructor & Destructor Documentation

### 6.59.2.1 Functor() [1/4]

```
template<typename ResultType , typename ... ArgTypes>
Functor< ResultType(ArgTypes...)>::Functor ( ) [default]
```

### 6.59.2.2 Functor() [2/4]

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

Definition at line 20 of file functor.h.

### 6.59.2.3 Functor() [3/4]

```
template<typename ResultType , typename ... ArgTypes>
template<typename Function , class Class >
Functor< ResultType(ArgTypes...)>::Functor (
    Function Class::* function ) [inline]
```

Definition at line 23 of file functor.h.

### 6.59.2.4 Functor() [4/4]

```
template<typename ResultType , typename ... ArgTypes>
Functor< ResultType(ArgTypes...)>::Functor (
    const Functor< ResultType(ArgTypes...)> & other ) [inline]
```

Definition at line 25 of file functor.h.

### 6.59.3 Member Function Documentation

#### 6.59.3.1 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

Definition at line 36 of file functor.h.

#### 6.59.3.2 operator=()

```
template<typename ResultType , typename ... ArgTypes>
Functor& Functor< ResultType(ArgTypes...)>::operator= (
    const Functor< ResultType(ArgTypes...)> & other ) [inline]
```

Definition at line 27 of file functor.h.

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

- [functor.h](#)

## 6.60 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

#### 6.60.1 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

Definition at line 15 of file generate\_type\_lists.h.

## 6.60.2 Member Typedef Documentation

### 6.60.2.1 result

```
template<int n>
using TL::GenerateTypeLists< n >::result = typename Add< EmptyTypeList, 0, typename GenerateTypeLists<n
- 1>::result >::result
```

Definition at line 16 of file generate\_type\_lists.h.

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

- [TL/generate\\_type\\_lists.h](#)

## 6.61 TL::GenerateTypeLists< 0 > Struct Reference

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

### Public Types

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

### 6.61.1 Detailed Description

Definition at line 24 of file generate\_type\_lists.h.

### 6.61.2 Member Typedef Documentation

### 6.61.2.1 result

```
using TL::GenerateTypeLists< 0 >::result = EmptyTypeList
```

Definition at line 25 of file generate\_type\_lists.h.

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

- [TL/generate\\_type\\_lists.h](#)

## 6.62 GLib::GetReachedVertexes< graph, start > Struct Template Reference

```
#include <get_reached_vertexes.h>
```

### Classes

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

### Public Types

- using [start\\_node](#) = typename [FindNodeByVertex< start, graph >::result](#)
- using [dfs\\_search](#) = typename [DFS< start\\_node, graph >::result](#)
- using [result](#) = typename [TL::Add< start, 0, typename IterateThroughEdges< dfs\\_search >::result >::result](#)

### 6.62.1 Detailed Description

```
template<class graph, typename start>
struct GLib::GetReachedVertexes< graph, start >
```

Gets all vertexes that can be reached from vertex start.

See also

[DFS](#)

#### Parameters

<i>graph</i>	Template parameter, graph to process
<i>start</i>	Template parameter

#### Returns

Parameter result, all vertexes that are reached from start.

Definition at line 20 of file `get_reached_vertexes.h`.

## 6.62.2 Member Typedef Documentation

### 6.62.2.1 dfs\_search

```
template<class graph , typename start >
using GLib::GetReachedVertexes< graph, start >::dfs_search = typename DFS<start_node, graph>↵
::result
```

Definition at line 27 of file `get_reached_vertexes.h`.

### 6.62.2.2 result

```
template<class graph , typename start >
using GLib::GetReachedVertexes< graph, start >::result = typename TL::Add< start, 0, typename
IterateThroughEdges<dfs_search>::result >::result
```

Definition at line 46 of file `get_reached_vertexes.h`.

### 6.62.2.3 start\_node

```
template<class graph , typename start >
using GLib::GetReachedVertexes< graph, start >::start_node = typename FindNodeByVertex<start,
graph>::result
```

Definition at line 25 of file `get_reached_vertexes.h`.

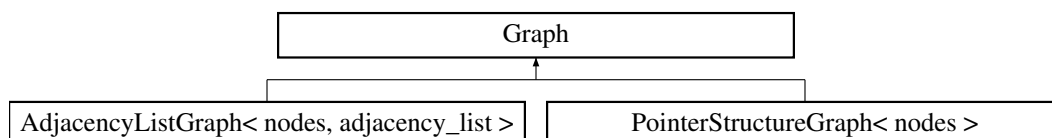
The documentation for this struct was generated from the following file:

- `graph/GLib/get_reached_vertexes.h`

## 6.63 Graph Struct Reference

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

Inheritance diagram for Graph:



### 6.63.1 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.

Definition at line 11 of file [graph.h](#).

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

- [graph/graphs/graph.h](#)

## 6.64 TL::HasDerivedAndConstructible< type\_list, T > Struct Template Reference

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

### Static Public Attributes

- constexpr static bool [result](#)

### 6.64.1 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

Definition at line 33 of file [has\\_derived\\_and\\_constructible.h](#).

### 6.64.2 Member Data Documentation

### 6.64.2.1 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
```

Definition at line 34 of file `has_derived_and_constructible.h`.

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

- [TL/has\\_derived\\_and\\_constructible.h](#)

## 6.65 TL::HasDerivedAndConstructible< EmptyTypeList, T > Struct Template Reference

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

### Static Public Attributes

- constexpr static bool [result](#) = false

### 6.65.1 Detailed Description

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

See also

[HasDerivedAndConstructible](#)

Definition at line 46 of file `has_derived_and_constructible.h`.

### 6.65.2 Member Data Documentation

#### 6.65.2.1 result

```
template<typename T >
constexpr static bool TL::HasDerivedAndConstructible< EmptyTypeList, T >::result = false
[static], [constexpr]
```

Definition at line 47 of file `has_derived_and_constructible.h`.

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

- [TL/has\\_derived\\_and\\_constructible.h](#)

## 6.66 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

### 6.66.1 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

Definition at line 15 of file index\_of.h.

## 6.66.2 Member Data Documentation

### 6.66.2.1 value

```
template<class type_list , typename T >
constexpr static int TL::IndexOf< type_list, T >::value = 1 + IndexOf<typename type_list::↵
Tail, T>::value [static], [constexpr]
```

Definition at line 16 of file index\_of.h.

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

- [TL/index\\_of.h](#)

## 6.67 TL::IndexOf< EmptyTypeList, T > Struct Template Reference

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



## Static Public Attributes

- constexpr static int [value](#) = INT32\_MIN

### 6.67.1 Detailed Description

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

See also

[IndexOf](#)

Definition at line 31 of file `index_of.h`.

### 6.67.2 Member Data Documentation

#### 6.67.2.1 value

```
template<typename T >
constexpr static int TL::IndexOf< EmptyTypeList, T >::value = INT32_MIN [static], [constexpr]
```

Definition at line 32 of file `index_of.h`.

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

- [TL/index\\_of.h](#)

## 6.68 TL::IndexOf< type\_list, typename type\_list::Head > Struct Template Reference

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

## Static Public Attributes

- constexpr static int [value](#) = 0

### 6.68.1 Detailed Description

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

See also

[IndexOf](#)

Definition at line 23 of file `index_of.h`.

## 6.68.2 Member Data Documentation

### 6.68.2.1 value

```
template<class type_list >
constexpr static int TL::IndexOf< type_list, typename type_list::Head >::value = 0 [static],
[constexpr]
```

Definition at line 24 of file index\_of.h.

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

- [TL/index\\_of.h](#)

## 6.69 Objects::Integer< integer > Struct Template Reference

```
#include <objects.h>
```

### Static Public Attributes

- constexpr static int [value](#) = integer

### 6.69.1 Detailed Description

```
template<int integer>
struct Objects::Integer< integer >
```

Definition at line 8 of file objects.h.

## 6.69.2 Member Data Documentation

### 6.69.2.1 value

```
template<int integer>
constexpr static int Objects::Integer< integer >::value = integer [static], [constexpr]
```

Definition at line 9 of file objects.h.

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

- [graph/objects.h](#)

## 6.70 TL::IsBaseOf< parent, derived > Struct Template Reference

```
#include <is_base_of.h>
```

### Static Public Attributes

- constexpr static bool [result](#)

### 6.70.1 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

Definition at line 38 of file `is_base_of.h`.

### 6.70.2 Member Data Documentation

#### 6.70.2.1 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
```

Definition at line 39 of file `is_base_of.h`.

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

- [TL/is\\_base\\_of.h](#)

## 6.71 TL::IsBaseOf< EmptyTypeList, derived > Struct Template Reference

```
#include <is_base_of.h>
```

### Static Public Attributes

- constexpr static bool [result](#) = false

#### 6.71.1 Detailed Description

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

Definition at line 52 of file [is\\_base\\_of.h](#).

#### 6.71.2 Member Data Documentation

##### 6.71.2.1 result

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

Definition at line 53 of file [is\\_base\\_of.h](#).

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

- [TL/is\\_base\\_of.h](#)

## 6.72 TL::IsBaseOf< EmptyTypeList, EmptyTypeList > Struct Reference

```
#include <is_base_of.h>
```

### Static Public Attributes

- constexpr static bool [result](#) = true

#### 6.72.1 Detailed Description

Definition at line 57 of file [is\\_base\\_of.h](#).

## 6.72.2 Member Data Documentation

### 6.72.2.1 result

```
constexpr static bool TL::IsBaseOf< EmptyTypeList, EmptyTypeList >::result = true [static],  
[constexpr]
```

Definition at line 58 of file is\_base\_of.h.

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

- [TL/is\\_base\\_of.h](#)

## 6.73 TL::IsBaseOf< parent, EmptyTypeList > Struct Template Reference

```
#include <is_base_of.h>
```

### Static Public Attributes

- constexpr static bool [result](#) = true

### 6.73.1 Detailed Description

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

Definition at line 47 of file is\_base\_of.h.

## 6.73.2 Member Data Documentation

### 6.73.2.1 result

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

Definition at line 48 of file is\_base\_of.h.

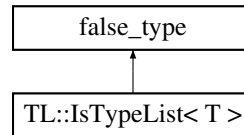
The documentation for this struct was generated from the following file:

- [TL/is\\_base\\_of.h](#)

## 6.74 TL::IsTypeList< T > Struct Template Reference

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

Inheritance diagram for TL::IsTypeList< T >:



### 6.74.1 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

Definition at line 14 of file `is_type_list.h`.

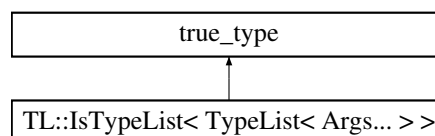
The documentation for this struct was generated from the following file:

- [TL/is\\_type\\_list.h](#)

## 6.75 TL::IsTypeList< TypeList< Args... > > Struct Template Reference

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

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



### 6.75.1 Detailed Description

```
template<class ... Args>
struct TL::IsTypeList< TypeList< Args... > >
```

Definition at line 17 of file is\_type\_list.h.

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

- [TL/is\\_type\\_list.h](#)

## 6.76 GLib::DFS< cur\_node, graph, visited\_nodes >::IterateThroughChildren< cur\_children, cur\_visited > Struct Template Reference

```
#include <dfs.h>
```

### Public Types

- using [cur\\_edge](#) = typename cur\_children::Head
- using [cur\\_child](#) = typename [GLib::FindNodeByVertex](#)< typename cur\_edge::to, graph >::result
- using [new\\_visited](#) = std::conditional\_t< [TL::Contains](#)< cur\_visited, [cur\\_child](#) >::result, [upd\\_visited](#), typename [DFS](#)< [cur\\_child](#), graph, [upd\\_visited](#) >::new\_visited >
- using [result](#) = std::conditional\_t< [TL::Contains](#)< [upd\\_visited](#), [cur\\_child](#) >::result, typename [IterateThroughChildren](#)< typename cur\_children::Tail, [new\\_visited](#) >::result, typename [TL::Add](#)< [cur\\_edge](#), 0, typename [TL::Concatenate](#)< typename [DFS](#)< [cur\\_child](#), graph, [upd\\_visited](#) >::result, typename [IterateThroughChildren](#)< typename cur\_children::Tail, [new\\_visited](#) >::result >::result >::result >

### 6.76.1 Detailed Description

```
template<class cur_node, class graph, class visited_nodes = EmptyTypeList>
template<class cur_children, class cur_visited>
struct GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< cur_children, cur_visited >
```

Definition at line 28 of file dfs.h.

### 6.76.2 Member Typedef Documentation

#### 6.76.2.1 cur\_child

```
template<class cur_node , class graph , class visited_nodes = EmptyTypeList>
template<class cur_children , class cur_visited >
using GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< cur_children,
cur_visited >::cur_child = typename GLib::FindNodeByVertex< typename cur_edge::to, graph >↔
::result
```

Definition at line 30 of file dfs.h.

### 6.76.2.2 cur\_edge

```
template<class cur_node , class graph , class visited_nodes = EmptyTypeList>
template<class cur_children , class cur_visited >
using GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< cur_children,
cur_visited >::cur_edge = typename cur_children::Head
```

Definition at line 29 of file dfs.h.

### 6.76.2.3 new\_visited

```
template<class cur_node , class graph , class visited_nodes = EmptyTypeList>
template<class cur_children , class cur_visited >
using GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< cur_children,
cur_visited >::new_visited = std::conditional_t< TL::Contains<cur_visited, cur_child>::result,
upd_visited, typename DFS<cur_child, graph, upd_visited>::new_visited >
```

Definition at line 35 of file dfs.h.

### 6.76.2.4 result

```
template<class cur_node , class graph , class visited_nodes = EmptyTypeList>
template<class cur_children , class cur_visited >
using GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< cur_children,
cur_visited >::result = std::conditional_t< TL::Contains<upd_visited, cur_child>::result,
typename IterateThroughChildren< typename cur_children::Tail, new_visited >::result, typename
TL::Add< cur_edge, 0, typename TL::Concatenate< typename DFS<cur_child, graph, upd_visited>::
::result, typename IterateThroughChildren< typename cur_children::Tail, new_visited >::result
>::result >::result >
```

Definition at line 41 of file dfs.h.

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

- graph/GLib/dfs.h

## 6.77 GLib::DFS< cur\_node, graph, visited\_nodes >::IterateThroughChildren< EmptyTypeList, cur\_unvisited > Struct Template Reference

```
#include <dfs.h>
```

### Public Types

- using `result` = `EmptyTypeList`
- using `new_visited` = `upd_visited`



### 6.77.1 Detailed Description

```
template<class cur_node, class graph, class visited_nodes = EmptyTypeList>
template<class cur_unvisited>
struct GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< EmptyTypeList, cur_unvisited >
```

Definition at line 62 of file dfs.h.

### 6.77.2 Member Typedef Documentation

#### 6.77.2.1 new\_visited

```
template<class cur_node , class graph , class visited_nodes = EmptyTypeList>
template<class cur_unvisited >
using GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< EmptyTypeList,
cur_unvisited >::new_visited = upd_visited
```

Definition at line 64 of file dfs.h.

#### 6.77.2.2 result

```
template<class cur_node , class graph , class visited_nodes = EmptyTypeList>
template<class cur_unvisited >
using GLib::DFS< cur_node, graph, visited_nodes >::IterateThroughChildren< EmptyTypeList,
cur_unvisited >::result = EmptyTypeList
```

Definition at line 63 of file dfs.h.

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

- graph/GLib/dfs.h

## 6.78 GLib::FindPath< graph\_raw, start, finish >::IterateThroughEdges< cur\_edges, wanted\_node > Struct Template Reference

```
#include <find_path.h>
```

## Public Types

- using `cur_edge` = typename cur\_edges::Head
- using `path` = typename std::conditional\_t< `found`, typename `TL::Add`< typename wanted\_node::vertex, 0, typename `IterateThroughEdges`< typename cur\_edges::Tail, typename `FindNodeByVertex`< typename cur\_edge::from, `graph`>::result >::path >::result, typename `IterateThroughEdges`< typename cur\_edges↵::Tail, wanted\_node >::path >
- using `weights` = typename std::conditional\_t< `found`, typename `TL::Add`< typename cur\_edge::weight, 0, typename `IterateThroughEdges`< typename cur\_edges::Tail, typename `FindNodeByVertex`< typename cur\_edge::from, `graph`>::result >::weights >::result, typename `IterateThroughEdges`< typename cur↵edges::Tail, wanted\_node >::weights >

## Static Public Attributes

- constexpr static bool `found`

### 6.78.1 Detailed Description

```
template<class graph_raw, typename start, typename finish>
template<class cur_edges, class wanted_node>
struct GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< cur_edges, wanted_node >
```

Definition at line 36 of file find\_path.h.

### 6.78.2 Member Typedef Documentation

#### 6.78.2.1 cur\_edge

```
template<class graph_raw , typename start , typename finish >
template<class cur_edges , class wanted_node >
using GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< cur_edges, wanted_node
>::cur_edge = typename cur_edges::Head
```

Definition at line 37 of file find\_path.h.

#### 6.78.2.2 path

```
template<class graph_raw , typename start , typename finish >
template<class cur_edges , class wanted_node >
using GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< cur_edges, wanted_node
>::path = typename std::conditional_t<found, typename TL::Add< typename wanted_node::vertex,
0, typename IterateThroughEdges< typename cur_edges::Tail, typename FindNodeByVertex< typename
cur_edge::from, graph>::result >::path >::result, typename IterateThroughEdges<typename
cur_edges::Tail, wanted_node>::path >
```

Definition at line 43 of file find\_path.h.

### 6.78.2.3 weights

```
template<class graph_raw , typename start , typename finish >
template<class cur_edges , class wanted_node >
using GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< cur_edges, wanted_node
>::weights = typename std::conditional_t<found, typename TL::Add< typename cur_edge::weight,
0, typename IterateThroughEdges< typename cur_edges::Tail, typename FindNodeByVertex< typename
cur_edge::from, graph >::result >::weights >::result, typename IterateThroughEdges<typename
cur_edges::Tail, wanted_node>::weights >
```

Definition at line 57 of file find\_path.h.

## 6.78.3 Member Data Documentation

### 6.78.3.1 found

```
template<class graph_raw , typename start , typename finish >
template<class cur_edges , class wanted_node >
constexpr static bool GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< cur_↵
edges, wanted_node >::found [static], [constexpr]
```

#### Initial value:

```
= std::is_same<
    typename cur_edge::to,
    typename wanted_node::vertex
>::value
```

Definition at line 38 of file find\_path.h.

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

- graph/GLib/find\_path.h

## 6.79 GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< cur\_edges > Struct Template Reference

```
#include <get_reached_vertexes.h>
```

### Public Types

- using [cur\\_edge](#) = typename cur\_edges::Head
- using [result](#) = typename TL::Add< typename cur\_edge::to, 0, typename IterateThroughEdges< typename cur\_edges::Tail >::result >::result

### 6.79.1 Detailed Description

```
template<class graph, typename start>
template<class cur_edges>
struct GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< cur_edges >
```

Definition at line 31 of file `get_reached_vertexes.h`.

### 6.79.2 Member Typedef Documentation

#### 6.79.2.1 cur\_edge

```
template<class graph , typename start >
template<class cur_edges >
using GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< cur_edges >::cur_edge =
typename cur_edges::Head
```

Definition at line 32 of file `get_reached_vertexes.h`.

#### 6.79.2.2 result

```
template<class graph , typename start >
template<class cur_edges >
using GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< cur_edges >::result =
typename TL::Add< typename cur_edges::to, 0, typename IterateThroughEdges<typename cur_edges::to, 0, typename cur_edges::Tail>::result >::result
```

Definition at line 33 of file `get_reached_vertexes.h`.

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

- [graph/GLib/get\\_reached\\_vertexes.h](#)

## 6.80 ConvertGraph< EDGE\_LIST, ADJACENCY\_LIST, graph >::IterateThroughEdges< edge\_list > Struct Template Reference

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

### Public Types

- using `result` = typename `GLib::AddEdge< ADJACENCY_LIST, typename IterateThroughEdges< typename edge_list::Tail >::result, typename edge_list::Head >::result`
- using `result` = typename `GLib::AddEdge< ADJACENCY_LIST, typename IterateThroughEdges< typename edge_list::Tail >::result, typename edge_list::Head >::result`

### 6.80.1 Detailed Description

```
template<class graph>
template<class edge_list>
struct ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< edge_list >
```

Definition at line 25 of file convert\_from\_edge\_list.h.

### 6.80.2 Member Typedef Documentation

#### 6.80.2.1 result [1/2]

```
template<class graph >
template<class edge_list >
using ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< edge_list >↔
::result = typename GLib::AddEdge< ADJACENCY_LIST, typename IterateThroughEdges<typename
edge_list::Tail>::result, typename edge_list::Head >::result
```

Definition at line 26 of file convert\_from\_edge\_list.h.

#### 6.80.2.2 result [2/2]

```
template<class graph >
template<class edge_list >
using ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< edge_list >↔
::result = typename GLib::AddEdge< ADJACENCY_LIST, typename IterateThroughEdges<typename
edge_list::Tail>::result, typename edge_list::Head >::result
```

Definition at line 26 of file convert\_to\_adjacency\_list.h.

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

- graph/graphs/convert\_from\_edge\_list.h
- graph/graphs/convert\_to\_adjacency\_list.h

## 6.81 GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< EmptyTypeList > Struct Reference

```
#include <get_reached_vertexes.h>
```

### Public Types

- using result = EmptyTypeList

### 6.81.1 Detailed Description

```
template<class graph, typename start>
struct GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< EmptyTypeList >
```

Definition at line 41 of file `get_reached_vertexes.h`.

### 6.81.2 Member Typedef Documentation

#### 6.81.2.1 result

```
template<class graph , typename start >
using GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< EmptyTypeList >::result
= EmptyTypeList
```

Definition at line 42 of file `get_reached_vertexes.h`.

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

- `graph/GLib/get_reached_vertexes.h`

## 6.82 ConvertGraph< EDGE\_LIST, ADJACENCY\_LIST, graph >::IterateThroughEdges< EmptyTypeList > Struct Reference

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

### Public Types

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

### 6.82.1 Detailed Description

```
template<class graph>
struct ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< EmptyTypeList >
```

Definition at line 34 of file `convert_from_edge_list.h`.

### 6.82.2 Member Typedef Documentation

### 6.82.2.1 result [1/2]

```
template<class graph >
using ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< EmptyTypeList
>::result = AdjacencyListGraph< typename graph::vertexes_, typename TL::GenerateTypeLists<TL::Size<typename
graph::vertexes_>::size>::result >
```

Definition at line 35 of file convert\_from\_edge\_list.h.

### 6.82.2.2 result [2/2]

```
template<class graph >
using ConvertGraph< EDGE_LIST, ADJACENCY_LIST, graph >::IterateThroughEdges< EmptyTypeList
>::result = AdjacencyListGraph< typename graph::vertexes_, typename TL::GenerateTypeLists<TL::Size<typename
graph::vertexes_>::size>::result >
```

Definition at line 35 of file convert\_to\_adjacency\_list.h.

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

- graph/graphs/[convert\\_from\\_edge\\_list.h](#)
- graph/graphs/[convert\\_to\\_adjacency\\_list.h](#)

## 6.83 GLib::FindPath< graph\_raw, start, finish >::IterateThroughEdges< EmptyTypeList, wanted\_node > Struct Template Reference

```
#include <find_path.h>
```

### Public Types

- using [path](#) = [EmptyTypeList](#)
- using [weights](#) = [EmptyTypeList](#)

### 6.83.1 Detailed Description

```
template<class graph_raw, typename start, typename finish>
template<class wanted_node>
struct GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< EmptyTypeList, wanted_node >
```

Definition at line 73 of file find\_path.h.

### 6.83.2 Member Typedef Documentation

### 6.83.2.1 path

```
template<class graph_raw , typename start , typename finish >
template<class wanted_node >
using GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< EmptyTypeList, wanted←
_node >::path = EmptyTypeList
```

Definition at line 74 of file find\_path.h.

### 6.83.2.2 weights

```
template<class graph_raw , typename start , typename finish >
template<class wanted_node >
using GLib::FindPath< graph_raw, start, finish >::IterateThroughEdges< EmptyTypeList, wanted←
_node >::weights = EmptyTypeList
```

Definition at line 75 of file find\_path.h.

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

- graph/GLib/[find\\_path.h](#)

## 6.84 TL::Reverse< type\_list >::IterateThroughElements< cur\_type\_list, cur\_result > Struct Template Reference

```
#include <reverse.h>
```

### Public Types

- using [result](#) = typename [IterateThroughElements](#)< typename cur\_type\_list::Tail, typename [TL::Add](#)< type-name cur\_type\_list::Head, 0, cur\_result >::[result](#) >::[result](#)

### 6.84.1 Detailed Description

```
template<class type_list>
template<class cur_type_list, class cur_result>
struct TL::Reverse< type_list >::IterateThroughElements< cur_type_list, cur_result >
```

Definition at line 14 of file reverse.h.

### 6.84.2 Member Typedef Documentation



### 6.84.2.1 result

```
template<class type_list >
template<class cur_type_list , class cur_result >
using TL::Reverse< type_list >::IterateThroughElements< cur_type_list, cur_result >::result
= typename IterateThroughElements < typename cur_type_list::Tail, typename TL::Add< typename
cur_type_list::Head, 0, cur_result >::result >::result
```

Definition at line 15 of file reverse.h.

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

- [TL/reverse.h](#)

## 6.85 TL::Reverse< type\_list >::IterateThroughElements< EmptyTypeList, cur\_result > Struct Template Reference

```
#include <reverse.h>
```

### Public Types

- using [result](#) = cur\_result

### 6.85.1 Detailed Description

```
template<class type_list>
template<class cur_result>
struct TL::Reverse< type_list >::IterateThroughElements< EmptyTypeList, cur_result >
```

Definition at line 26 of file reverse.h.

### 6.85.2 Member Typedef Documentation

#### 6.85.2.1 result

```
template<class type_list >
template<class cur_result >
using TL::Reverse< type_list >::IterateThroughElements< EmptyTypeList, cur_result >::result =
cur_result
```

Definition at line 27 of file reverse.h.

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

- [TL/reverse.h](#)

## 6.86 GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< cur\_nodes > Struct Template Reference

```
#include <find_node_by_vertex.h>
```

### Public Types

- using [cur\\_node](#) = typename cur\_nodes::Head
- using [result](#) = std::conditional\_t< std::is\_same< vertex, typename cur\_node::vertex >::value, [cur\\_node](#), typename [IterateThroughNodes](#)< typename cur\_nodes::Tail >::result >

### 6.86.1 Detailed Description

```
template<typename vertex, class graph>
template<class cur_nodes>
struct GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< cur_nodes >
```

Definition at line 17 of file [find\\_node\\_by\\_vertex.h](#).

### 6.86.2 Member Typedef Documentation

#### 6.86.2.1 cur\_node

```
template<typename vertex , class graph >
template<class cur_nodes >
using GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< cur_nodes >::cur_node =
typename cur_nodes::Head
```

Definition at line 18 of file [find\\_node\\_by\\_vertex.h](#).

#### 6.86.2.2 result

```
template<typename vertex , class graph >
template<class cur_nodes >
using GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< cur_nodes >::result =
std::conditional_t< std::is_same<vertex, typename cur_node::vertex>::value, cur\_node, typename
IterateThroughNodes<typename cur_nodes::Tail>::result >
```

Definition at line 19 of file [find\\_node\\_by\\_vertex.h](#).

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

- [graph/GLib/find\\_node\\_by\\_vertex.h](#)

## 6.87 GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< EmptyTypeList > Struct Reference

```
#include <find_node_by_vertex.h>
```

### Public Types

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

#### 6.87.1 Detailed Description

```
template<typename vertex, class graph>  
struct GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< EmptyTypeList >
```

Definition at line 27 of file `find_node_by_vertex.h`.

#### 6.87.2 Member Typedef Documentation

##### 6.87.2.1 result

```
template<typename vertex , class graph >  
using GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< EmptyTypeList >::result =  
NullType
```

Definition at line 28 of file `find_node_by_vertex.h`.

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

- `graph/GLib/find_node_by_vertex.h`

## 6.88 TL::Concatenate< front, back >::IterateThroughReversedFront< elements, current > Struct Template Reference

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

### Public Types

- using [added](#) = typename [Add](#)< typename elements::Head, 0, current >::result
- using [result](#) = typename [IterateThroughReversedFront](#)< typename elements::Tail, [added](#) >::result

### 6.88.1 Detailed Description

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

Definition at line 24 of file concatenate.h.

### 6.88.2 Member Typedef Documentation

#### 6.88.2.1 added

```
template<class front , class back >
template<class elements , class current >
using TL::Concatenate< front, back >::IterateThroughReversedFront< elements, current >::added
= typename Add< typename elements::Head, 0, current >::result
```

Definition at line 25 of file concatenate.h.

#### 6.88.2.2 result

```
template<class front , class back >
template<class elements , class current >
using TL::Concatenate< front, back >::IterateThroughReversedFront< elements, current >↵
::result = typename IterateThroughReversedFront< typename elements::Tail, added >::result
```

Definition at line 31 of file concatenate.h.

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

- [TL/concatenate.h](#)

## 6.89 TL::Concatenate< front, back >::IterateThroughReversedFront< EmptyTypeList, current > Struct Template Reference

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

### Public Types

- using [result](#) = current

### 6.89.1 Detailed Description

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

Definition at line 38 of file concatenate.h.

### 6.89.2 Member Typedef Documentation

#### 6.89.2.1 result

```
template<class front , class back >
template<class current >
using TL::Concatenate< front, back >::IterateThroughReversedFront< EmptyTypeList, current >↔
::result = current
```

Definition at line 39 of file concatenate.h.

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

- TL/[concatenate.h](#)

## 6.90 ConvertGraph< ADJACENCY\_LIST, POINTER\_STRUCTURE, graph >::MakePointerStructureGraph< current\_vertexes, current\_adjacency\_list > Struct Template Reference

```
#include <convert_to_pointer_structure.h>
```

### Public Types

- using [type\\_list\\_without\\_first](#) = typename MakePointerStructureGraph< typename current\_vertexes::Tail, typename current\_adjacency\_list::Tail >::result
- using [result](#) = typename TL::Add< [PointerStructureNode](#)< typename current\_vertexes::Head, typename current\_adjacency\_list::Head >, 0, [type\\_list\\_without\\_first](#) >::result

### 6.90.1 Detailed Description

```
template<class graph>
template<class current_vertexes, class current_adjacency_list>
struct ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< current_vertexes, current_adjacency_list >
```

Definition at line 28 of file convert\_to\_pointer\_structure.h.

## 6.90.2 Member Typedef Documentation

### 6.90.2.1 result

```
template<class graph >
template<class current_vertexes , class current_adjacency_list >
using ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph<
current_vertexes, current_adjacency_list >::result = typename TL::Add< PointerStructureNode<
typename current_vertexes::Head, typename current_adjacency_list::Head >, 0, type_list_without_first
>::result
```

Definition at line 34 of file `convert_to_pointer_structure.h`.

### 6.90.2.2 type\_list\_without\_first

```
template<class graph >
template<class current_vertexes , class current_adjacency_list >
using ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph<
current_vertexes, current_adjacency_list >::type_list_without_first = typename MakePointer↵
StructureGraph< typename current_vertexes::Tail, typename current_adjacency_list::Tail >↵
::result
```

Definition at line 29 of file `convert_to_pointer_structure.h`.

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

- `graph/graphs/convert_to_pointer_structure.h`

## 6.91 ConvertGraph< ADJACENCY\_LIST, POINTER\_STRUCTURE, graph >::MakePointerStructureGraph< EmptyTypeList, EmptyTypeList > Struct Reference

```
#include <convert_to_pointer_structure.h>
```

### Public Types

- using `result` = `EmptyTypeList`

### 6.91.1 Detailed Description

```
template<class graph>
struct ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph< EmptyTypeList,
EmptyTypeList >
```

Definition at line 45 of file `convert_to_pointer_structure.h`.

## 6.91.2 Member Typedef Documentation

### 6.91.2.1 result

```
template<class graph >
using ConvertGraph< ADJACENCY_LIST, POINTER_STRUCTURE, graph >::MakePointerStructureGraph<
EmptyTypeList, EmptyTypeList >::result = EmptyTypeList
```

Definition at line 46 of file convert\_to\_pointer\_structure.h.

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

- graph/graphs/[convert\\_to\\_pointer\\_structure.h](#)

## 6.92 TL::MostDerived< type\_list, T > Struct Template Reference

```
#include <most_derived.h>
```

### Public Types

- using [result](#) = typename [CheckMostDerived](#)< type\_list, T, std::is\_base\_of< T, typename type\_list::Head >::value >::[result](#)

### 6.92.1 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

Definition at line 35 of file most\_derived.h.

## 6.92.2 Member Typedef Documentation

### 6.92.2.1 result

```
template<class type_list , typename T >
using TL::MostDerived< type_list, T >::result = typename CheckMostDerived< type_list, T,
std::is_base_of<T, typename type_list::Head>::value >::result
```

Definition at line 36 of file most\_derived.h.

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

- [TL/most\\_derived.h](#)

## 6.93 TL::MostDerived< EmptyTypeList, T > Struct Template Reference

```
#include <most_derived.h>
```

### Public Types

- using [result](#) = T

### 6.93.1 Detailed Description

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

Definition at line 44 of file most\_derived.h.

### 6.93.2 Member Typedef Documentation

#### 6.93.2.1 result

```
template<typename T >
using TL::MostDerived< EmptyTypeList, T >::result = T
```

Definition at line 45 of file most\_derived.h.

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

- [TL/most\\_derived.h](#)



## 6.94 TL::MostDerivedAndConstructible< type\_list, T > Struct Template Reference

```
#include <most_derived_and_constructible.h>
```

### 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](#)

### 6.94.1 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

Definition at line 33 of file `most_derived_and_constructible.h`.

### 6.94.2 Member Typedef Documentation

#### 6.94.2.1 result

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

Definition at line 34 of file `most_derived_and_constructible.h`.

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

- [TL/most\\_derived\\_and\\_constructible.h](#)

## 6.95 TL::MostDerivedAndConstructible< EmptyTypeList, T > Struct Template Reference

```
#include <most_derived_and_constructible.h>
```

### Public Types

- using [result](#) = T

#### 6.95.1 Detailed Description

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

Definition at line 43 of file `most_derived_and_constructible.h`.

#### 6.95.2 Member Typedef Documentation

##### 6.95.2.1 result

```
template<typename T >
using TL::MostDerivedAndConstructible< EmptyTypeList, T >::result = T
```

Definition at line 44 of file `most_derived_and_constructible.h`.

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

- [TL/most\\_derived\\_and\\_constructible.h](#)

## 6.96 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 >

#### 6.96.1 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

Definition at line 11 of file no\_duplicates.h.

## 6.96.2 Member Typedef Documentation

### 6.96.2.1 result

```
template<class type_list >
using TL::NoDuplicates< type_list >::result = TypeList< typename type_list::Head, typename
NoDuplicates< typename RemoveAll< typename type_list::Tail, typename type_list::Head >↵
::result >::result >
```

Definition at line 12 of file no\_duplicates.h.

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

- [TL/no\\_duplicates.h](#)

## 6.97 TL::NoDuplicates< EmptyTypeList > Struct Reference

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

### Public Types

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

### 6.97.1 Detailed Description

See also

[NoDuplicates](#)

Definition at line 26 of file no\_duplicates.h.

## 6.97.2 Member Typedef Documentation

### 6.97.2.1 result

```
using TL::NoDuplicates< EmptyTypeList >::result = EmptyTypeList
```

Definition at line 27 of file no\_duplicates.h.

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

- [TL/no\\_duplicates.h](#)

## 6.98 NullType Struct Reference

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

### 6.98.1 Detailed Description

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

Definition at line 7 of file null\_type.h.

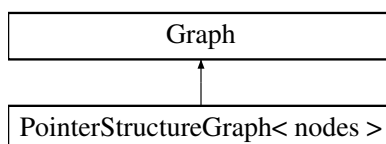
The documentation for this struct was generated from the following file:

- [TL/null\\_type.h](#)

## 6.99 PointerStructureGraph< nodes > Struct Template Reference

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

Inheritance diagram for PointerStructureGraph< nodes >:



### Classes

- struct [ConvertTo](#)

## Public Types

- using `nodes_` = nodes  
*All accounted vertexes in this graph.*

## Static Public Attributes

- constexpr static `GraphType TYPE = POINTER_STRUCTURE`

### 6.99.1 Detailed Description

```
template<class nodes>
struct PointerStructureGraph< nodes >
```

Represents graph as a structure with pointers Every vertex must be contained within node. Node must have a [TypeList](#) "children", which is a [TypeList](#) of Edges, showing who can be reached from this vertex. Also node must have a field "vertex" ~— the vertex this node contains.

#### Parameters

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

Definition at line 21 of file `pointer_structure_graph.h`.

### 6.99.2 Member Typedef Documentation

#### 6.99.2.1 nodes\_

```
template<class nodes >
using PointerStructureGraph< nodes >::nodes_ = nodes
```

All accounted vertexes in this graph.

Definition at line 25 of file `pointer_structure_graph.h`.

### 6.99.3 Member Data Documentation

### 6.99.3.1 TYPE

```
template<class nodes >
constexpr static GraphType PointerStructureGraph< nodes >::TYPE = POINTER_STRUCTURE [static],
[constexpr]
```

Definition at line 22 of file pointer\_structure\_graph.h.

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

- graph/graphs/[pointer\\_structure\\_graph.h](#)

## 6.100 PointerStructureNode< vertex\_, children\_ > Struct Template Reference

```
#include <pointer_structure_node.h>
```

### Public Types

- using [vertex](#) = vertex\_
- using [children](#) = children\_

### 6.100.1 Detailed Description

```
template<class vertex_, class children_>
struct PointerStructureNode< vertex_, children_ >
```

Default version of a suitable class for [PointerStructureGraph](#). It's not necessary to use this one. In fact, it's encouraged to make your objects suitable to [PointerStructureGraph](#). It can be done by adding field "children" to every vertex in the graph.

#### Parameters

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

Definition at line 13 of file pointer\_structure\_node.h.

### 6.100.2 Member Typedef Documentation

### 6.100.2.1 children

```
template<class vertex_ , class children_ >
using PointerStructureNode< vertex_, children_ >::children = children_
```

Definition at line 16 of file pointer\_structure\_node.h.

### 6.100.2.2 vertex

```
template<class vertex_ , class children_ >
using PointerStructureNode< vertex_, children_ >::vertex = vertex_
```

Definition at line 15 of file pointer\_structure\_node.h.

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

- graph/graphs/[pointer\\_structure\\_node.h](#)

## 6.101 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 >

### 6.101.1 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

Definition at line 11 of file remove.h.

## 6.101.2 Member Typedef Documentation

### 6.101.2.1 result

```
template<class type_list , typename T >
using TL::Remove< type_list, T >::result = TypeList<typename type_list::Head, typename Remove<typename
type_list::Tail, T>::result>
```

Definition at line 12 of file remove.h.

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

- [TL/remove.h](#)

## 6.102 TL::Remove< EmptyTypeList, T > Struct Template Reference

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

### Public Types

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

### 6.102.1 Detailed Description

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

See also

[Remove](#)

Definition at line 28 of file remove.h.

## 6.102.2 Member Typedef Documentation

### 6.102.2.1 result

```
template<typename T >
using TL::Remove< EmptyTypeList, T >::result = EmptyTypeList
```

Definition at line 29 of file remove.h.

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

- [TL/remove.h](#)



## 6.103 TL::Remove< type\_list, typename type\_list::Head > Struct Template Reference

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

### Public Types

- using [result](#) = typename type\_list::Tail

#### 6.103.1 Detailed Description

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

See also

[Remove](#)

Definition at line 20 of file remove.h.

#### 6.103.2 Member Typedef Documentation

##### 6.103.2.1 result

```
template<class type_list >
using TL::Remove< type_list, typename type_list::Head >::result = typename type_list::Tail
```

Definition at line 21 of file remove.h.

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

- TL/[remove.h](#)

## 6.104 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 >

#### 6.104.1 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

Definition at line 39 of file remove.h.

## 6.104.2 Member Typedef Documentation

### 6.104.2.1 result

```
template<class type_list , class T >
using TL::RemoveAll< type_list, T >::result = TypeList<typename type_list::Head, typename
RemoveAll< typename type_list::Tail, T >::result >
```

Definition at line 40 of file remove.h.

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

- [TL/remove.h](#)

## 6.105 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

### 6.105.1 Detailed Description

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

See also

[RemoveAll](#)

Definition at line 51 of file remove.h.

## 6.105.2 Member Typedef Documentation

### 6.105.2.1 result

```
template<class type_list >
using TL::RemoveAll< type_list, typename type_list::Head >::result = typename RemoveAll<
typename type_list::Tail, typename type_list::Head >::result
```

Definition at line 52 of file remove.h.

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

- [TL/remove.h](#)

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

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

### 6.106.1 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 <a href="#">TypeList</a>
<i>ind</i>	Number of this position
<i>TypeList&lt;Arg,Args...&gt;</i>	This <a href="#">TypeList</a>

#### Returns

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

Definition at line 14 of file replace.h.

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

- [TL/replace.h](#)

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

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

## Public Types

- using `result` = `TypeList`< T, Args... >

### 6.107.1 Detailed Description

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

See also

[Replace](#)

Definition at line 34 of file `replace.h`.

### 6.107.2 Member Typedef Documentation

#### 6.107.2.1 `result`

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

Definition at line 35 of file `replace.h`.

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

- [TL/replace.h](#)

## 6.108 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`

### 6.108.1 Detailed Description

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

See also

[Replace](#)

Definition at line 20 of file replace.h.

### 6.108.2 Member Typedef Documentation

#### 6.108.2.1 end

```
template<typename T , size_t ind, class Arg , class ... Args>
using TL::Replace< T, ind, TypeList< Arg, Args... > >::end = typename Replace< T, ind - 1,
TypeList<Args...> >::result
```

Definition at line 21 of file replace.h.

#### 6.108.2.2 result

```
template<typename T , size_t ind, class Arg , class ... Args>
using TL::Replace< T, ind, TypeList< Arg, Args... > >::result = typename Add<Arg, 0, end>↵
::result
```

Definition at line 27 of file replace.h.

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

- [TL/replace.h](#)

## 6.109 TL::Reverse< type\_list > Struct Template Reference

```
#include <reverse.h>
```

### Classes

- struct [IterateThroughElements](#)
- struct [IterateThroughElements< EmptyTypeList, cur\\_result >](#)

### Public Types

- using [result](#) = typename [IterateThroughElements< type\\_list, EmptyTypeList >::result](#)

### 6.109.1 Detailed Description

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

Reverses type\_list

## Parameters

<code>type_list</code>	Template parameter
------------------------	--------------------

## Returns

Parameter result, reversed `type_list`

Definition at line 12 of file `reverse.h`.

## 6.109.2 Member Typedef Documentation

### 6.109.2.1 result

```
template<class type_list >
using TL::Reverse< type_list >::result = typename IterateThroughElements<type_list, EmptyTypeList>←
::result
```

Definition at line 30 of file `reverse.h`.

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

- [TL/reverse.h](#)

## 6.110 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

### 6.110.1 Detailed Description

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

Gets length of a [TypeList](#)

## Parameters

<a href="#">TypeList</a>	Template parameter
--------------------------	--------------------

### Returns

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

Definition at line 12 of file size.h.

## 6.110.2 Member Data Documentation

### 6.110.2.1 size

```
template<class type_list >
constexpr static size_t TL::Size< type_list >::size = 1 + Size<typename type_list::Tail>←
::size [static], [constexpr]
```

Definition at line 14 of file size.h.

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

- TL/[size.h](#)

## 6.111 TL::Size< EmptyTypeList > Struct Reference

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

### Static Public Attributes

- constexpr static size\_t [size](#) = 0

### 6.111.1 Detailed Description

See also

[Size](#)

Definition at line 21 of file size.h.

## 6.111.2 Member Data Documentation

### 6.111.2.1 size

```
constexpr static size_t TL::Size< EmptyTypeList >::size = 0 [static], [constexpr]
```

Definition at line 22 of file size.h.

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

- [TL/size.h](#)

## 6.112 Stream< T > Class Template Reference

```
#include <stream.h>
```

### Public Member Functions

- [Stream](#) (std::vector< T > stream)
- template<class Consumer >  
void [ForEach](#) (Consumer consumer)
- template<class Predicate >  
[Stream](#) & [Filter](#) (Predicate predicate)
- template<class NewType , class Function >  
[Stream](#)< NewType > [Map](#) (Function function)
- template<class NewType , class BinaryOperation >  
NewType [Reduce](#) (BinaryOperation binary\_operation, NewType default\_value)
- std::vector< T > [Collect](#) ()

### 6.112.1 Detailed Description

```
template<typename T>  
class Stream< T >
```

Represents an analogue of std::ranges. Unlike std::ranges, this class is compatible with earlier standards. It was made specifically for indexes of vertexes, but it can be used for other purposes. If this is a stream of indexes of vertexes, then it's recommended to use functors in FunctorStreamType.

#### See also

FunctorStreamType  
java.util.stream.Stream

Definition at line 14 of file stream.h.

### 6.112.2 Constructor & Destructor Documentation



### 6.112.2.1 Stream()

```
template<typename T >
Stream< T >::Stream (
    std::vector< T > stream ) [inline], [explicit]
```

Definition at line 19 of file stream.h.

## 6.112.3 Member Function Documentation

### 6.112.3.1 Collect()

```
template<typename T >
std::vector<T> Stream< T >::Collect ( ) [inline]
```

Collects stream into a vector.

#### Returns

Contents of a stream in a vector.

Definition at line 78 of file stream.h.

### 6.112.3.2 Filter()

```
template<typename T >
template<class Predicate >
Stream& Stream< T >::Filter (
    Predicate predicate ) [inline]
```

Removes elements from stream where predicate returns false.

#### Parameters

<i>predicate</i>	Predicate, that takes object and returns true or false.
------------------	---

#### Returns

This stream, after modification.

Definition at line 38 of file stream.h.

### 6.112.3.3 ForEach()

```
template<typename T >
template<class Consumer >
void Stream< T >::ForEach (
    Consumer consumer ) [inline]
```

Performs passed action on every index in stream.

#### Parameters

<i>consumer</i>	Consumer, that takes object and performs an action.
-----------------	---

Definition at line 26 of file stream.h.

### 6.112.3.4 Map()

```
template<typename T >
template<class NewType , class Function >
Stream<NewType> Stream< T >::Map (
    Function function ) [inline]
```

Converts every object into another one.

#### Parameters

<i>function</i>	Function, that takes object and returns another object.
-----------------	---

#### Returns

New stream, after modification.

Definition at line 52 of file stream.h.

### 6.112.3.5 Reduce()

```
template<typename T >
template<class NewType , class BinaryOperation >
NewType Stream< T >::Reduce (
    BinaryOperation binary_operation,
    NewType default_value ) [inline]
```

Adds up all objects by using specific BinaryOperation.

## Parameters

<i>binary_operation</i>	BinaryOperation, that takes result and current object, and returns result.
<i>default_value</i>	Starting result of binary_operation.

## Returns

Final result of binary\_operation

Definition at line 67 of file stream.h.

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

- graph/[stream.h](#)

## 6.113 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](#)

### 6.113.1 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](#)

Definition at line 15 of file type\_at.h.

### 6.113.2 Member Typedef Documentation

### 6.113.2.1 value

```
template<class type_list , size_t ind>
using TL::TypeAt< type_list, ind >::value = typename TypeAt<typename type_list::Tail, ind -
1>::value
```

Definition at line 18 of file type\_at.h.

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

- [TL/type\\_at.h](#)

## 6.114 TL::TypeAt< type\_list, 0 > Struct Template Reference

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

### Public Types

- using [value](#) = typename type\_list::Head

### 6.114.1 Detailed Description

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

See also

[TypeAt](#)

Definition at line 25 of file type\_at.h.

### 6.114.2 Member Typedef Documentation

#### 6.114.2.1 value

```
template<class type_list >
using TL::TypeAt< type_list, 0 >::value = typename type_list::Head
```

Definition at line 26 of file type\_at.h.

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

- [TL/type\\_at.h](#)

## 6.115 `TypeList< Args >` Struct Template Reference

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

### Public Types

- using `Head` = `NullType`
- using `Tail` = `TypeList<>`

### 6.115.1 Detailed Description

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

See also

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

Definition at line 9 of file `type_list.h`.

### 6.115.2 Member Typedef Documentation

#### 6.115.2.1 Head

```
template<typename ... Args>  
using TypeList< Args >::Head = NullType
```

Definition at line 10 of file `type_list.h`.

#### 6.115.2.2 Tail

```
template<typename ... Args>  
using TypeList< Args >::Tail = TypeList<>
```

Definition at line 11 of file `type_list.h`.

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

- [TL/type\\_list.h](#)

## 6.116 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.*

### 6.116.1 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

Definition at line 35 of file `type_list.h`.

### 6.116.2 Member Typedef Documentation

#### 6.116.2.1 Head

```
template<typename H , typename ... T>
using TypeList< H, T... >::Head = H
```

First type in a type list.

Definition at line 36 of file `type_list.h`.

#### 6.116.2.2 Tail

```
template<typename H , typename ... T>
using TypeList< H, T... >::Tail = TypeList<T...>
```

[TypeList](#) of other types.

Definition at line 37 of file `type_list.h`.

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

- [TL/type\\_list.h](#)

## 6.117 `TypeList< T >` Struct Template Reference

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

### Public Types

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

### 6.117.1 Detailed Description

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

See also

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

Definition at line 24 of file `type_list.h`.

### 6.117.2 Member Typedef Documentation

#### 6.117.2.1 Head

```
template<typename T >  
using TypeList< T >::Head = T
```

Definition at line 25 of file `type_list.h`.

#### 6.117.2.2 Tail

```
template<typename T >  
using TypeList< T >::Tail = EmptyTypeList
```

Definition at line 26 of file `type_list.h`.

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

- [TL/type\\_list.h](#)

## 6.118 VertexStream< stream, graph > Class Template Reference

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

### Static Public Member Functions

- static std::vector< size\_t > [MapVertexesToReversedIndexes](#) ()
- static [Stream](#)< size\_t > [MapVertexesToIndexes](#) ()

### 6.118.1 Detailed Description

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

Represents a stream of vertexes of a graph

#### Parameters

<i>stream</i>	Template parameter, <a href="#">TypeList</a> of vertexes of a graph
<i>graph</i>	Template parameter

Definition at line 19 of file vertex\_stream.h.

### 6.118.2 Member Function Documentation

#### 6.118.2.1 MapVertexesToIndexes()

```
template<class stream , class graph >  
static Stream<size_t> VertexStream< stream, graph >::MapVertexesToIndexes ( ) [inline],  
[static]
```

Converts given stream into vector of indexes of vertexes.

#### Returns

Vector of indexes of vertexes

Definition at line 38 of file vertex\_stream.h.



### 6.118.2.2 MapVertexesToReversedIndexes()

```
template<class stream , class graph >
static std::vector<size_t> VertexStream< stream, graph >::MapVertexesToReversedIndexes ( )
[inline], [static]
```

Converts given stream into reversed vector of indexes of vertexes. Also it's based on a variation of "Chain of a responsibility" pattern

#### Returns

Vector of reversed indexes of vertexes

Definition at line 26 of file vertex\_stream.h.

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

- graph/[vertex\\_stream.h](#)

## 6.119 VertexStream< EmptyTypeList, graph > Struct Template Reference

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

### Static Public Member Functions

- static std::vector< size\_t > [MapVertexesToReversedIndexes](#) ()

### 6.119.1 Detailed Description

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

See also

[VertexStream](#)

Definition at line 50 of file vertex\_stream.h.

### 6.119.2 Member Function Documentation

#### 6.119.2.1 MapVertexesToReversedIndexes()

```
template<class graph >
static std::vector<size_t> VertexStream< EmptyTypeList, graph >::MapVertexesToReversedIndexes
( ) [inline], [static]
```

Definition at line 51 of file vertex\_stream.h.

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

- graph/[vertex\\_stream.h](#)



## Chapter 7

# File Documentation

### 7.1 Debug/CodeAnalysisResultManifest.txt File Reference

### 7.2 Debug/library.vcxproj.FileListAbsolute.txt File Reference

### 7.3 functor.h File Reference

```
#include <cassert>
#include <memory>
```

#### Classes

- class [Functor< ResultType\(ArgTypes...\)>](#)

### 7.4 graph/edge.h File Reference

```
#include "../TL/null_type.h"
```

#### Classes

- struct [Edge< from\\_, to\\_, weight\\_ >](#)

### 7.5 graph/examples/graph\_examples.cpp File Reference

```
#include "../graphs/adjacency_list_graph.h"
#include "../graphs/adjacency_matrix_graph.h"
#include "../graphs/edge_list_graph.h"
#include "../graphs/pointer_structure_graph.h"
#include "../edge.h"
#include "../objects.h"
#include "../graphs/pointer_structure_node.h"
```

## Functions

- int [main](#) ()

### 7.5.1 Function Documentation

#### 7.5.1.1 main()

```
int main ( )
```

Definition at line 12 of file graph\_examples.cpp.

## 7.6 graph/examples/vertex\_stream\_example.cpp File Reference

```
#include <functional>
#include <iostream>
#include "../edge.h"
#include "../objects.h"
#include "../stream.h"
#include "../graphs/edge_list_graph.h"
#include "../stream_functor_type.h"
#include "../vertex_stream.h"
```

## Functions

- template<class graph >  
bool [IsAmongFirst3](#) (size\_t vertex\_ind)
- template<class graph >  
int [Add1](#) (size\_t vertex\_ind)
- int [main](#) ()

### 7.6.1 Function Documentation

#### 7.6.1.1 Add1()

```
template<class graph >
int Add1 (
    size_t vertex_ind )
```

Definition at line 21 of file vertex\_stream\_example.cpp.

### 7.6.1.2 IsAmongFirst3()

```
template<class graph >
bool IsAmongFirst3 (
    size_t vertex_ind )
```

Definition at line 16 of file vertex\_stream\_example.cpp.

### 7.6.1.3 main()

```
int main ( )
```

Definition at line 25 of file vertex\_stream\_example.cpp.

## 7.7 graph/GLib/add\_edge.h File Reference

```
#include "../TL/replace.h"
#include "../TL/type_list.h"
#include "../TL/type_at.h"
#include "../TL/index_of.h"
#include "../graphs/graph_type.h"
#include "../graphs/adjacency_list_graph.h"
```

### Classes

- struct [GLib::AddEdge< ADJACENCY\\_LIST, graph, edge >](#)

### Namespaces

- [GLib](#)

## 7.8 graph/GLib/dfs.h File Reference

```
#include "../TL/contains.h"
#include "../graphs/pointer_structure_graph.h"
```

### Classes

- struct [GLib::DFS< cur\\_node, graph, visited\\_nodes >](#)
- struct [GLib::DFS< cur\\_node, graph, visited\\_nodes >::IterateThroughChildren< cur\\_children, cur\\_visited >](#)
- struct [GLib::DFS< cur\\_node, graph, visited\\_nodes >::IterateThroughChildren< EmptyTypeList, cur\\_unvisited >](#)

## Namespaces

- [GLib](#)

## 7.9 graph/GLib/find\_node\_by\_vertex.h File Reference

```
#include "../graphs/pointer_structure_graph.h"
```

## Classes

- struct [GLib::FindNodeByVertex](#)< vertex, graph >
- struct [GLib::FindNodeByVertex](#)< vertex, graph >::IterateThroughNodes< cur\_nodes >
- struct [GLib::FindNodeByVertex](#)< vertex, graph >::IterateThroughNodes< EmptyTypeList >
- struct [GLib::FindNodeByVertex](#)< vertex, EmptyTypeList >

## Namespaces

- [GLib](#)

## 7.10 graph/GLib/find\_path.h File Reference

```
#include <type_traits>
#include "../../TL/concatenate.h"
#include "../../TL/reverse.h"
#include "find_node_by_vertex.h"
#include "dfs.h"
#include "../graphs/convert_graph.h"
#include "../graphs/convert_to_pointer_structure.h"
```

## Classes

- struct [GLib::FindPath](#)< graph\_raw, start, finish >
- struct [GLib::FindPath](#)< graph\_raw, start, finish >::IterateThroughEdges< cur\_edges, wanted\_node >
- struct [GLib::FindPath](#)< graph\_raw, start, finish >::IterateThroughEdges< EmptyTypeList, wanted\_node >

## Namespaces

- [GLib](#)

## 7.11 graph/GLib/get\_reached\_vertexes.h File Reference

```
#include <type_traits>
#include "find_node_by_vertex.h"
#include "dfs.h"
#include "../graphs/convert_graph.h"
#include "../graphs/convert_to_pointer_structure.h"
```

### Classes

- struct [GLib::GetReachedVertexes< graph, start >](#)
- struct [GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< cur\\_edges >](#)
- struct [GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< EmptyTypeList >](#)

### Namespaces

- [GLib](#)

## 7.12 graph/GLib/map\_indexes\_to\_vertexes.h File Reference

## 7.13 graph/graphs/adjacency\_list\_graph.h File Reference

```
#include "graph.h"
#include "../../TL/add.h"
#include "../../TL/contains.h"
#include "../../TL/index_of.h"
#include "../../TL/is_type_list.h"
#include "../../TL/size.h"
#include "../../TL/type_at.h"
#include "../../TL/type_list.h"
#include "convert_graph.h"
```

### Classes

- struct [AdjacencyListGraph< nodes, adjacency\\_list >](#)
- struct [AdjacencyListGraph< nodes, adjacency\\_list >::ConvertTo< type >](#)

## 7.14 graph/graphs/adjacency\_matrix\_graph.h File Reference

```
#include "graph.h"
#include "../../TL/is_type_list.h"
#include "../../TL/generate_type_lists.h"
#include "../GLib/add_edge.h"
#include "convert_graph.h"
```

## Classes

- struct [AdjacencyMatrixGraph< nodes, matrix >](#)
- struct [AdjacencyMatrixGraph< nodes, matrix >::ConvertTo< type >](#)

## 7.15 graph/graphs/convert\_from\_edge\_list.h File Reference

```
#include "convert_graph.h"
#include "../../TL/type_list.h"
#include "../../TL/generate_type_lists.h"
#include "../../TL/size.h"
#include "../GLib/add_edge.h"
#include "adjacency_list_graph.h"
#include "adjacency_matrix_graph.h"
#include "edge_list_graph.h"
#include "pointer_structure_graph.h"
#include "graph_type.h"
```

## Classes

- struct [ConvertGraph< EDGE\\_LIST, ADJACENCY\\_LIST, graph >](#)
- struct [ConvertGraph< EDGE\\_LIST, ADJACENCY\\_LIST, graph >::IterateThroughEdges< edge\\_list >](#)
- struct [ConvertGraph< EDGE\\_LIST, ADJACENCY\\_LIST, graph >::IterateThroughEdges< EmptyTypeList >](#)

## 7.16 graph/graphs/convert\_from\_pointer\_structure.h File Reference

```
#include "convert_graph.h"
#include "../../TL/type_list.h"
#include "../../TL/generate_type_lists.h"
#include "../../TL/size.h"
#include "../add_edge.h"
#include "adjacency_list_graph.h"
#include "adjacency_matrix_graph.h"
#include "edge_list_graph.h"
#include "pointer_structure_graph.h"
#include "graph_type.h"
```

## Classes

- struct [ConvertGraph< POINTER\\_STRUCTURE, EDGE\\_LIST, graph >](#)

## 7.17 graph/graphs/convert\_graph.h File Reference

```
#include "graph_type.h"
```



## Classes

- struct [ConvertGraph< type, type, graph >](#)

## 7.18 graph/graphs/convert\_to\_adjacency\_list.h File Reference

```
#include "convert_graph.h"
#include "../../TL/type_list.h"
#include "../../TL/generate_type_lists.h"
#include "../../TL/size.h"
#include "../../GLib/add_edge.h"
#include "adjacency_list_graph.h"
#include "adjacency_matrix_graph.h"
#include "edge_list_graph.h"
#include "pointer_structure_graph.h"
#include "graph_type.h"
```

## Classes

- struct [ConvertGraph< EDGE\\_LIST, ADJACENCY\\_LIST, graph >](#)
- struct [ConvertGraph< EDGE\\_LIST, ADJACENCY\\_LIST, graph >::IterateThroughEdges< edge\\_list >](#)
- struct [ConvertGraph< EDGE\\_LIST, ADJACENCY\\_LIST, graph >::IterateThroughEdges< EmptyTypeList >](#)

## 7.19 graph/graphs/convert\_to\_pointer\_structure.h File Reference

```
#include "convert_graph.h"
#include "../../TL/add.h"
#include "../../TL/generate_type_lists.h"
#include "../../TL/size.h"
#include "../../TL/type_list.h"
#include "../../GLib/add_edge.h"
#include "adjacency_list_graph.h"
#include "adjacency_matrix_graph.h"
#include "edge_list_graph.h"
#include "pointer_structure_graph.h"
#include "pointer_structure_node.h"
#include "graph_type.h"
```

## Classes

- struct [ConvertGraph< ADJACENCY\\_LIST, POINTER\\_STRUCTURE, graph >](#)
- struct [ConvertGraph< ADJACENCY\\_LIST, POINTER\\_STRUCTURE, graph >::MakePointerStructureGraph< current\\_vertexes >](#)
- struct [ConvertGraph< ADJACENCY\\_LIST, POINTER\\_STRUCTURE, graph >::MakePointerStructureGraph< EmptyTypeList, >](#)
- struct [ConvertGraph< EDGE\\_LIST, POINTER\\_STRUCTURE, graph >](#)

## 7.20 graph/graphs/edge\_list\_graph.h File Reference

```
#include "graph.h"
#include "../../TL/is_type_list.h"
#include "../../TL/generate_type_lists.h"
#include "../GLib/add_edge.h"
#include "convert_graph.h"
```

### Classes

- struct [EdgeListGraph](#)< nodes, edge\_list >
- struct [EdgeListGraph](#)< nodes, edge\_list >::ConvertTo< type >

## 7.21 graph/graphs/graph.h File Reference

```
#include "graph_type.h"
```

### Classes

- struct [Graph](#)

## 7.22 graph/graphs/graph\_type.h File Reference

### Enumerations

- enum [GraphType](#) { [ADJACENCY\\_MATRIX](#), [ADJACENCY\\_LIST](#), [EDGE\\_LIST](#), [POINTER\\_STRUCTURE](#) }

### 7.22.1 Enumeration Type Documentation

#### 7.22.1.1 GraphType

enum [GraphType](#)

Types, by which graph can be created. For more details, see corresponding file.

#### See also

[AdjacencyMatrixGraph](#)  
[AdjacencyListGraph](#)  
[EdgeListGraph](#)  
[PointerStructureGraph](#)

## Enumerator

ADJACENCY_MATRIX	<a href="#">Graph</a> is represented as an adjacency matrix of booleans.
ADJACENCY_LIST	<a href="#">Graph</a> is represented as an adjacency list ( <a href="#">TypeList</a> of <a href="#">TypeLists</a> of edges).
EDGE_LIST	<a href="#">Graph</a> is represented by a collection of edges.
POINTER_STRUCTURE	<a href="#">Graph</a> is represented by a pointer structure.

Definition at line 11 of file graph\_type.h.

## 7.23 graph/graphs/pointer\_structure\_graph.h File Reference

```
#include "graph.h"
#include "../../TL/concatenate.h"
#include "../../TL/is_type_list.h"
#include "../../TL/remove.h"
#include "../../GLib/find_node_by_vertex.h"
#include "pointer_structure_node.h"
```

### Classes

- struct [PointerStructureGraph< nodes >](#)
- struct [PointerStructureGraph< nodes >::ConvertTo< type >](#)

## 7.24 graph/graphs/pointer\_structure\_node.h File Reference

```
#include "../../TL/is_type_list.h"
```

### Classes

- struct [PointerStructureNode< vertex\\_, children\\_ >](#)

## 7.25 graph/objects.h File Reference

### Classes

- struct [Objects::Integer< integer >](#)
- struct [Objects::Boolean< boolean >](#)

### Namespaces

- [Objects](#)

## 7.26 graph/stream.h File Reference

```
#include <algorithm>
#include <vector>
```

### Classes

- class [Stream< T >](#)

## 7.27 graph/stream\_functor\_type.h File Reference

```
#include "../functor.h"
```

### Namespaces

- [StreamFunctorType](#)

### Typedefs

- template<class graph >  
using [StreamFunctorType::Consumer](#) = [Functor](#)< void(size\_t)>
- template<class graph >  
using [StreamFunctorType::Predicate](#) = [Functor](#)< bool(size\_t)>
- template<class graph , class ResultType >  
using [StreamFunctorType::Function](#) = [Functor](#)< ResultType(size\_t)>
- template<class graph , class ResultType >  
using [StreamFunctorType::BinaryOperation](#) = [Functor](#)< ResultType(ResultType, size\_t)>

## 7.28 graph/vertex\_stream.h File Reference

```
#include <ranges>
#include <vector>
#include "../TL/add.h"
#include "../TL/contains.h"
#include "../TL/index_of.h"
#include "../TL/type_list.h"
#include "stream.h"
```

### Classes

- class [VertexStream< stream, graph >](#)
- struct [VertexStream< EmptyTypeList, graph >](#)

## 7.29 TL/add.h File Reference

```
#include "type_list.h"
```

### Classes

- struct [TL::Add< T, ind, TypeList< Arg, Args... > >](#)
- struct [TL::Add< T, 0, TypeList< Arg, Args... > >](#)
- struct [TL::Add< T, 0, TypeList< Args... > >](#)

### Namespaces

- [TL](#)

## 7.30 TL/concatenate.h File Reference

```
#include "add.h"  
#include "reverse.h"  
#include "size.h"  
#include "is_type_list.h"
```

### Classes

- struct [TL::Concatenate< front, back >](#)
- struct [TL::Concatenate< front, back >::IterateThroughReversedFront< elements, current >](#)
- struct [TL::Concatenate< front, back >::IterateThroughReversedFront< EmptyTypeList, current >](#)

### Namespaces

- [TL](#)

## 7.31 TL/contains.h File Reference

```
#include "index_of.h"  
#include "type_list.h"
```

### Classes

- struct [TL::Contains< type\\_list, T >](#)

## Namespaces

- [TL](#)

## 7.32 TL/contains\_constructible\_parent.h File Reference

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

## Classes

- struct [CheckContainsConstructibleParent](#)< type\_list, T, is\_parent >
- struct [CheckContainsConstructibleParent](#)< type\_list, T, false >
- struct [CheckContainsConstructibleParent](#)< type\_list, T, true >
- struct [TL::ContainsConstructibleParent](#)< type\_list, T >
- struct [TL::ContainsConstructibleParent](#)< EmptyTypeList, T >

## Namespaces

- [TL](#)

## 7.33 TL/contains\_parent.h File Reference

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

## Classes

- struct [CheckContainsParent](#)< type\_list, T, is\_parent >
- struct [CheckContainsParent](#)< type\_list, T, false >
- struct [CheckContainsParent](#)< type\_list, T, true >
- struct [TL::ContainsParent](#)< type\_list, T >
- struct [TL::ContainsParent](#)< EmptyTypeList, T >

## Namespaces

- [TL](#)

## 7.34 TL/find\_parent\_type\_list.h File Reference

```
#include "is_base_of.h"
#include "type_list.h"
```

## Classes

- struct [CheckFindParentTypeList< contains\\_class, T, type\\_list, type\\_lists >](#)
- struct [CheckFindParentTypeList< true, T, type\\_list, type\\_lists... >](#)
- struct [CheckFindParentTypeList< false, T, type\\_list, type\\_lists... >](#)
- struct [TL::FindParentTypeList< T, type\\_list, type\\_lists >](#)

## Namespaces

- [TL](#)

## 7.35 TL/find\_type\_list\_by\_class.h File Reference

```
#include "contains.h"
#include "type_list.h"
```

## Classes

- struct [CheckFindTypeListByClass< contains\\_class, T, type\\_list, type\\_lists >](#)
- struct [CheckFindTypeListByClass< true, T, type\\_list, type\\_lists... >](#)
- struct [CheckFindTypeListByClass< false, T, type\\_list, type\\_lists... >](#)
- struct [TL::FindTypeListByClass< T, type\\_list, type\\_lists >](#)

## Namespaces

- [TL](#)

## 7.36 TL/generate\_type\_lists.h File Reference

```
#include "add.h"
#include "type_list.h"
```

## Classes

- struct [TL::GenerateTypeLists< n >](#)
- struct [TL::GenerateTypeLists< 0 >](#)

## Namespaces

- [TL](#)

## 7.37 TL/has\_derived\_and\_constructible.h File Reference

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

### Classes

- struct [CheckHasDerivedAndConstructible< type\\_list, T, is\\_head\\_parent\\_of\\_T >](#)
- struct [CheckHasDerivedAndConstructible< type\\_list, T, true >](#)
- struct [CheckHasDerivedAndConstructible< type\\_list, T, false >](#)
- struct [TL::HasDerivedAndConstructible< type\\_list, T >](#)
- struct [TL::HasDerivedAndConstructible< EmptyTypeList, T >](#)

### Namespaces

- [TL](#)

## 7.38 TL/index\_of.h File Reference

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

### Classes

- struct [TL::IndexOf< type\\_list, T >](#)
- struct [TL::IndexOf< type\\_list, typename type\\_list::Head >](#)
- struct [TL::IndexOf< EmptyTypeList, T >](#)

### Namespaces

- [TL](#)

## 7.39 TL/is\_base\_of.h File Reference

```
#include <type_traits>
#include "contains_parent.h"
#include "type_list.h"
```



## Classes

- struct [CheckIsBaseOf< has\\_parent, parent, derived >](#)
- struct [CheckIsBaseOf< false, parent, derived >](#)
- struct [CheckIsBaseOf< true, parent, derived >](#)
- struct [TL::IsBaseOf< parent, derived >](#)
- struct [TL::IsBaseOf< parent, EmptyTypeList >](#)
- struct [TL::IsBaseOf< EmptyTypeList, derived >](#)
- struct [TL::IsBaseOf< EmptyTypeList, EmptyTypeList >](#)

## Namespaces

- [TL](#)

## 7.40 TL/is\_type\_list.h File Reference

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

## Classes

- struct [TL::IsTypeList< T >](#)
- struct [TL::IsTypeList< TypeList< Args... > >](#)

## Namespaces

- [TL](#)

## 7.41 TL/most\_derived.h File Reference

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

## Classes

- struct [CheckMostDerived< type\\_list, T, is\\_head\\_parent\\_of\\_T >](#)
- struct [CheckMostDerived< type\\_list, T, true >](#)
- struct [CheckMostDerived< type\\_list, T, false >](#)
- struct [TL::MostDerived< type\\_list, T >](#)
- struct [TL::MostDerived< EmptyTypeList, T >](#)

## Namespaces

- [TL](#)

## 7.42 TL/most\_derived\_and\_constructible.h File Reference

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

### Classes

- struct [CheckMostDerivedAndConstructible< type\\_list, T, is\\_head\\_parent\\_of\\_T >](#)
- struct [CheckMostDerivedAndConstructible< type\\_list, T, true >](#)
- struct [CheckMostDerivedAndConstructible< type\\_list, T, false >](#)
- struct [TL::MostDerivedAndConstructible< type\\_list, T >](#)
- struct [TL::MostDerivedAndConstructible< EmptyTypeList, T >](#)

### Namespaces

- [TL](#)

## 7.43 TL/no\_duplicates.h File Reference

```
#include "remove.h"
#include "type_list.h"
```

### Classes

- struct [TL::NoDuplicates< type\\_list >](#)
- struct [TL::NoDuplicates< EmptyTypeList >](#)

### Namespaces

- [TL](#)

## 7.44 TL/null\_type.h File Reference

### Classes

- struct [NullType](#)

## 7.45 TL/remove.h File Reference

```
#include "type_list.h"
```

## Classes

- struct [TL::Remove< type\\_list, T >](#)
- struct [TL::Remove< type\\_list, typename type\\_list::Head >](#)
- struct [TL::Remove< EmptyTypeList, T >](#)
- struct [TL::RemoveAll< type\\_list, T >](#)
- struct [TL::RemoveAll< type\\_list, typename type\\_list::Head >](#)

## Namespaces

- [TL](#)

## 7.46 TL/replace.h File Reference

```
#include "add.h"
#include "type_list.h"
```

## Classes

- struct [TL::Replace< T, ind, Arg, Args >](#)
- struct [TL::Replace< T, ind, TypeList< Arg, Args... > >](#)
- struct [TL::Replace< T, 0, TypeList< Arg, Args... > >](#)

## Namespaces

- [TL](#)

## 7.47 TL/reverse.h File Reference

```
#include "add.h"
```

## Classes

- struct [TL::Reverse< type\\_list >](#)
- struct [TL::Reverse< type\\_list >::IterateThroughElements< cur\\_type\\_list, cur\\_result >](#)
- struct [TL::Reverse< type\\_list >::IterateThroughElements< EmptyTypeList, cur\\_result >](#)

## Namespaces

- [TL](#)

## 7.48 TL/size.h File Reference

```
#include "is_type_list.h"  
#include "type_list.h"
```

### Classes

- struct [TL::Size< type\\_list >](#)
- struct [TL::Size< EmptyTypeList >](#)

### Namespaces

- [TL](#)

## 7.49 TL/type\_at.h File Reference

```
#include "is_type_list.h"  
#include "size.h"  
#include "type_list.h"
```

### Classes

- struct [TL::TypeAt< type\\_list, ind >](#)
- struct [TL::TypeAt< type\\_list, 0 >](#)

### Namespaces

- [TL](#)

## 7.50 TL/type\_list.h File Reference

```
#include "null_type.h"
```

### Classes

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

### Namespaces

- [TL](#)

## Typedefs

- using [EmptyTypeList](#) = [TypeList](#)<>
- template<typename ... Args>  
using [Typelist](#) = [TypeList](#)< Args... >

### 7.50.1 Typedef Documentation

#### 7.50.1.1 EmptyTypeList

```
using EmptyTypeList = TypeList<>
```

Represents [TypeList](#) with no data

See also

[TypeList](#)

Definition at line 18 of file type\_list.h.

#### 7.50.1.2 Typelist

```
template<typename ... Args>  
using Typelist = TypeList<Args...>
```

See also

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

Definition at line 44 of file type\_list.h.



## Chapter 8

# Example Documentation

### 8.1 `get_reached_vertexes.h`

An example of how to use DFS.

### 8.2 `graph_examples.cpp`

An example of how graph be created.

### 8.3 `vertex_stream_example.cpp`

An example of how to use [VertexStream](#).





# Index

Add1  
    vertex\_stream\_example.cpp, 130  
added  
    TL::Concatenate< front, back >::IterateThroughReverseOrder< T, type\_list, type\_lists...>, 98  
ADJACENCY\_LIST  
    graph\_type.h, 137  
adjacency\_list  
    ConvertGraph< EDGE\_LIST, POINTER\_STRUCTURE, T, type\_list, type\_lists...>, 52  
adjacency\_list\_  
    AdjacencyListGraph< nodes, adjacency\_list >, 24  
ADJACENCY\_MATRIX  
    graph\_type.h, 137  
AdjacencyListGraph< nodes, adjacency\_list >, 22  
    adjacency\_list\_, 24  
    GetVertexIndex, 24  
    HasEdge, 24  
    TYPE, 25  
    vertexes\_, 24  
AdjacencyListGraph< nodes, adjacency\_list >::ConvertTo< T, type\_list, type\_lists...>  
    type >, 56  
    result, 57  
AdjacencyMatrixGraph< nodes, matrix >, 25  
    matrix\_, 26  
    TYPE, 27  
    vertexes\_, 26  
AdjacencyMatrixGraph< nodes, matrix >::ConvertTo< T, type\_list, type\_lists...>  
    type >, 57  
    result, 58  
adjacent\_vertexes  
    GLib::AddEdge< ADJACENCY\_LIST, graph, edge >, 21  
  
BinaryOperation  
    StreamFunctorType, 14  
  
CheckContainsConstructibleParent< type\_list, T, false >, 28  
    result, 29  
CheckContainsConstructibleParent< type\_list, T, is\_parent >, 28  
CheckContainsConstructibleParent< type\_list, T, true >, 29  
    result, 29  
CheckContainsParent< type\_list, T, false >, 30  
    result, 31  
CheckContainsParent< type\_list, T, is\_parent >, 30  
CheckContainsParent< type\_list, T, true >, 31  
    result, 31  
CheckFindParentTypeList< contains\_class, T, type\_list, type\_lists >, 32  
    result, 32  
CheckFindParentTypeList< false, T, type\_list, type\_lists... >, 33  
    result, 33  
CheckFindParentTypeList< true, T, type\_list, type\_lists... >, 33  
    result, 34  
CheckFindTypeListByClass< contains\_class, T, type\_list, type\_lists >, 34  
    result, 34  
CheckFindTypeListByClass< false, T, type\_list, type\_lists... >, 35  
    result, 35  
CheckFindTypeListByClass< true, T, type\_list, type\_lists... >, 36  
    result, 36  
CheckHasDerivedAndConstructible< type\_list, T, false >, 37  
    result, 37  
CheckHasDerivedAndConstructible< type\_list, T, is\_head\_parent\_of\_T >, 36  
CheckHasDerivedAndConstructible< type\_list, T, true >, 37  
    result, 38  
CheckIsBaseOf< false, parent, derived >, 38  
    result, 39  
CheckIsBaseOf< has\_parent, parent, derived >, 38  
CheckIsBaseOf< true, parent, derived >, 39  
    result, 39  
CheckMostDerived< type\_list, T, false >, 41  
    result, 41  
CheckMostDerived< type\_list, T, is\_head\_parent\_of\_T >, 40  
    result, 40  
CheckMostDerived< type\_list, T, true >, 41  
    result, 42  
CheckMostDerivedAndConstructible< type\_list, T, false >, 42  
    result, 43  
CheckMostDerivedAndConstructible< type\_list, T, is\_head\_parent\_of\_T >, 42  
CheckMostDerivedAndConstructible< type\_list, T, true >, 43  
    result, 43  
children  
    PointerStructureNode< vertex\_, children\_ >, 108  
Collect

- Stream< T >, 119
- Consumer
  - StreamFunctorType, 14
- ConvertGraph< ADJACENCY\_LIST, POINTER\_STRUCTURE, graph\_type.h, 137
  - graph >, 50
  - result, 50
- ConvertGraph< ADJACENCY\_LIST, POINTER\_STRUCTURE, graph >::MakePointerStructureGraph< current\_vertexes, current\_adjacency\_list >, 99
  - result, 100
  - type\_list\_without\_first, 100
- ConvertGraph< ADJACENCY\_LIST, POINTER\_STRUCTURE, graph >::MakePointerStructureGraph< EmptyTypeList, EmptyTypeList >, 100
  - result, 101
- ConvertGraph< EDGE\_LIST, ADJACENCY\_LIST, graph >, 51
  - result, 51, 52
- ConvertGraph< EDGE\_LIST, ADJACENCY\_LIST, graph >::IterateThroughEdges< edge\_list >, 90
  - result, 91
- ConvertGraph< EDGE\_LIST, ADJACENCY\_LIST, graph >::IterateThroughEdges< EmptyTypeList >, 92
  - result, 92, 93
- ConvertGraph< EDGE\_LIST, POINTER\_STRUCTURE, graph >, 52
  - adjacency\_list, 52
  - result, 53
- ConvertGraph< From, To, graph >, 49
- ConvertGraph< POINTER\_STRUCTURE, EDGE\_LIST, graph >, 53
- ConvertGraph< type, type, graph >, 53
  - result, 54
- cur\_child
  - GLib::DFS< cur\_node, graph, visited\_nodes >::IterateThroughChildren< cur\_children, cur\_visited >, 85
- cur\_edge
  - GLib::DFS< cur\_node, graph, visited\_nodes >::IterateThroughChildren< cur\_children, cur\_visited >, 85
  - GLib::FindPath< graph\_raw, start, finish >::IterateThroughEdges< cur\_edges, wanted\_node >, 88
  - GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< cur\_edges >, 90
- cur\_node
  - GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< cur\_nodes >, 96
- Debug/CodeAnalysisResultManifest.txt, 129
- Debug/library.vcxproj.FileListAbsolute.txt, 129
- dfs\_search
  - GLib::FindPath< graph\_raw, start, finish >, 67
  - GLib::GetReachedVertexes< graph, start >, 75
- Edge< from\_, to\_, weight\_ >, 60
  - from, 60
  - to, 61
  - weight, 61
- EDGE\_LIST
  - edge\_list\_
    - EdgeListGraph< nodes, edge\_list >, 62
    - EdgeListGraph< nodes, edge\_list >, 61
    - edge\_list\_, 62
    - TYPE, 63
    - vertexes\_, 62
  - EdgeListGraph< nodes, edge\_list >::ConvertTo< type >, 54
    - result, 55
  - EmptyTypeList
    - type\_list.h, 147
  - end
    - TL::Add< T, ind, TypeList< Arg, Args... > >, 19
    - TL::Replace< T, ind, TypeList< Arg, Args... > >, 115
- Filter
  - Stream< T >, 119
- finish\_node
  - GLib::FindPath< graph\_raw, start, finish >, 67
- ForEach
  - Stream< T >, 119
- found
  - GLib::FindPath< graph\_raw, start, finish >::IterateThroughEdges< cur\_edges, wanted\_node >, 89
- from
  - Edge< from\_, to\_, weight\_ >, 60
- Function
  - StreamFunctorType, 14
- Functor
  - Functor< ResultType(ArgTypes...)>, 71
  - Functor< ResultType(ArgTypes...)>, 70
  - Functor, 71
  - operator(), 72
  - operator=, 72
  - Functor< ResultType, ArgTypes >, 70
  - functor.h, 129
- GetVertexIndex
  - AdjacencyListGraph< nodes, adjacency\_list >, 24
  - GLib, 13
  - GLib::AddEdge< ADJACENCY\_LIST, graph, edge >, 20
    - adjacent\_vertexes, 21
    - new\_adjacency\_list, 21
    - new\_adjacent\_vertexes, 21
    - result, 22
    - vertex\_num, 22
  - GLib::AddEdge< GraphType, graph, edge >, 20
  - GLib::DFS< cur\_node, graph, visited\_nodes >, 58
    - iterate\_through\_children, 59
    - new\_visited, 59
    - result, 59
    - upd\_visited, 59

- GLib::DFS< cur\_node, graph, visited\_nodes >::IterateThroughEdges<
  - cur\_children, cur\_visited >, 85
  - cur\_child, 85
  - cur\_edge, 85
  - new\_visited, 86
  - result, 86
- GLib::DFS< cur\_node, graph, visited\_nodes >::IterateThroughEdges<
  - EmptyTypeList, cur\_unvisited >, 86
  - new\_visited, 87
  - result, 87
- GLib::FindNodeByVertex< vertex, EmptyTypeList >, 64
  - result, 64
- GLib::FindNodeByVertex< vertex, graph >, 63
  - result, 64
- GLib::FindNodeByVertex< vertex, graph >::IterateThroughEdges<
  - cur\_nodes >, 96
  - cur\_node, 96
  - result, 96
- GLib::FindNodeByVertex< vertex, graph >::IterateThroughEdges<
  - EmptyTypeList >, 97
  - result, 97
- GLib::FindPath< graph\_raw, start, finish >, 66
  - dfs\_search, 67
  - finish\_node, 67
  - graph, 67
  - iterate\_through\_edges, 67
  - path, 68
  - reversed, 68
  - reversed\_path, 68
  - reversed\_weights, 68
  - start\_node, 68
  - weights, 69
- GLib::FindPath< graph\_raw, start, finish >::IterateThroughEdges<
  - cur\_edges, wanted\_node >, 87
  - cur\_edge, 88
  - found, 89
  - path, 88
  - weights, 88
- GLib::FindPath< graph\_raw, start, finish >::IterateThroughEdges<
  - EmptyTypeList, wanted\_node >, 93
  - path, 93
  - weights, 94
- GLib::GetReachedVertexes< graph, start >, 74
  - dfs\_search, 75
  - result, 75
  - start\_node, 75
- GLib::GetReachedVertexes< graph, start >::IterateThroughEdges<
  - cur\_edges >, 89
  - cur\_edge, 90
  - result, 90
- GLib::GetReachedVertexes< graph, start >::IterateThroughEdges<
  - EmptyTypeList >, 91
  - result, 92
- Graph, 75
- graph
  - GLib::FindPath< graph\_raw, start, finish >, 67
- graph/edge.h, 129
- graph/examples/graph\_examples.cpp, 129
- graph/examples/vertex\_stream\_example.cpp, 130
- graph/GLib/add\_edge.h, 131
- graph/GLib/dfs.h, 131
- graph/GLib/find\_node\_by\_vertex.h, 132
- graph/GLib/find\_path.h, 132
- graph/GLib/get\_reached\_vertexes.h, 133
- graph/GLib/map\_indexes\_to\_vertexes.h, 133
- graph/graphs/adjacency\_list\_graph.h, 133
- graph/graphs/adjacency\_matrix\_graph.h, 133
- graph/graphs/convert\_from\_edge\_list.h, 134
- graph/graphs/convert\_from\_pointer\_structure.h, 134
- graph/graphs/convert\_graph.h, 134
- graph/graphs/convert\_to\_adjacency\_list.h, 135
- graph/graphs/convert\_to\_pointer\_structure.h, 135
- graph/graphs/edge\_list\_graph.h, 136
- graph/graphs/graph.h, 136
- graph/graphs/graph\_type.h, 136
- graph/graphs/pointer\_structure\_graph.h, 137
- graph/graphs/pointer\_structure\_node.h, 137
- graph/objects.h, 137
- graph/stream.h, 138
- graph/stream\_functor\_type.h, 138
- graph/vertex\_stream.h, 138
- graph\_examples.cpp
  - main, 130
- graph\_type.h
  - ADJACENCY\_LIST, 137
  - ADJACENCY\_MATRIX, 137
  - EDGE\_LIST, 137
  - GraphType, 136
  - POINTER\_STRUCTURE, 137
- GraphType
  - graph\_type.h, 136
- HasEdge
  - AdjacencyListGraph< nodes, adjacency\_list >, 24
- Head
  - TypeList< Args >, 123
  - TypeList< H, T... >, 124
  - TypeList< T >, 125
- IsAmongFirst3
  - vertex\_stream\_example.cpp, 130
- iterate\_through\_children
  - GLib::DFS< cur\_node, graph, visited\_nodes >, 59
- iterate\_through\_edges
  - GLib::FindPath< graph\_raw, start, finish >, 67
- main
  - graph\_examples.cpp, 130
  - vertex\_stream\_example.cpp, 131
- Map
  - Stream< T >, 120
- MapVertexesToIndexes
  - VertexStream< stream, graph >, 126
- MapVertexesToReversedIndexes
  - VertexStream< EmptyTypeList, graph >, 127
  - VertexStream< stream, graph >, 126
- matrix\_

- AdjacencyMatrixGraph< nodes, matrix >, 26
- new\_adjacency\_list
  - GLib::AddEdge< ADJACENCY\_LIST, graph, edge >, 21
- new\_adjacent\_vertexes
  - GLib::AddEdge< ADJACENCY\_LIST, graph, edge >, 21
- new\_visited
  - GLib::DFS< cur\_node, graph, visited\_nodes >, 59
  - GLib::DFS< cur\_node, graph, visited\_nodes >::IterateThroughChildren< cur\_children, cur\_visited >, 86
  - GLib::DFS< cur\_node, graph, visited\_nodes >::IterateThroughChildren< EmptyTypeList, cur\_unvisited >, 87
- nodes\_
  - PointerStructureGraph< nodes >, 107
- NullType, 106
- Objects, 13
- Objects::Boolean< boolean >, 27
  - value, 27
- Objects::Integer< integer >, 80
  - value, 80
- operator()
  - Functor< ResultType(ArgTypes...) >, 72
- operator=
  - Functor< ResultType(ArgTypes...) >, 72
- path
  - GLib::FindPath< graph\_raw, start, finish >, 68
  - GLib::FindPath< graph\_raw, start, finish >::IterateThroughEdges< cur\_edges, wanted\_node >, 88
  - GLib::FindPath< graph\_raw, start, finish >::IterateThroughEdges< EmptyTypeList, wanted\_node >, 93
- POINTER\_STRUCTURE
  - graph\_type.h, 137
- PointerStructureGraph< nodes >, 106
  - nodes\_, 107
  - TYPE, 107
- PointerStructureGraph< nodes >::ConvertTo< type >, 55
  - result, 56
- PointerStructureNode< vertex\_, children\_ >, 108
  - children, 108
  - vertex, 109
- Predicate
  - StreamFunctorType, 14
- Reduce
  - Stream< T >, 120
- result
  - AdjacencyListGraph< nodes, adjacency\_list >::ConvertTo< type >, 57
  - AdjacencyMatrixGraph< nodes, matrix >::ConvertTo< type >, 58
  - CheckContainsConstructibleParent< type\_list, T, false >, 29
  - CheckContainsConstructibleParent< type\_list, T, true >, 29
  - CheckContainsParent< type\_list, T, false >, 31
  - CheckContainsParent< type\_list, T, true >, 31
  - CheckFindParentTypeList< contains\_class, T, type\_list, type\_lists >, 32
  - CheckFindParentTypeList< false, T, type\_list, type\_lists... >, 33
  - CheckFindParentTypeList< true, T, type\_list, type\_lists... >, 34
  - CheckFindTypeListByClass< contains\_class, T, type\_list, type\_lists >, 34
  - CheckFindTypeListByClass< false, T, type\_list, type\_lists... >, 35
  - CheckFindTypeListByClass< true, T, type\_list, type\_lists... >, 36
  - CheckHasDerivedAndConstructible< type\_list, T, false >, 37
  - CheckHasDerivedAndConstructible< type\_list, T, true >, 38
  - CheckIsBaseOf< false, parent, derived >, 39
  - CheckIsBaseOf< true, parent, derived >, 39
  - CheckMostDerived< type\_list, T, false >, 41
  - CheckMostDerived< type\_list, T, is\_head\_parent\_of\_T >, 40
  - CheckMostDerived< type\_list, T, true >, 42
  - CheckMostDerivedAndConstructible< type\_list, T, false >, 43
  - CheckMostDerivedAndConstructible< type\_list, T, true >, 43
  - ConvertGraph< ADJACENCY\_LIST, POINTER\_STRUCTURE, graph >, 50
  - ConvertGraph< ADJACENCY\_LIST, POINTER\_STRUCTURE, graph >::MakePointerStructureGraph< current\_vertexes, current\_adjacency\_list >, 100
  - ConvertGraph< ADJACENCY\_LIST, POINTER\_STRUCTURE, graph >::MakePointerStructureGraph< EmptyTypeList, EmptyTypeList >, 101
  - ConvertGraph< EDGE\_LIST, ADJACENCY\_LIST, graph >, 51, 52
  - ConvertGraph< EDGE\_LIST, ADJACENCY\_LIST, graph >::IterateThroughEdges< edge\_list >, 91
  - ConvertGraph< EDGE\_LIST, ADJACENCY\_LIST, graph >::IterateThroughEdges< EmptyTypeList >, 92, 93
  - ConvertGraph< EDGE\_LIST, POINTER\_STRUCTURE, graph >, 53
  - ConvertGraph< type, type, graph >, 54
  - EdgeListGraph< nodes, edge\_list >::ConvertTo< type >, 55
  - GLib::AddEdge< ADJACENCY\_LIST, graph, edge >, 22
  - GLib::DFS< cur\_node, graph, visited\_nodes >, 59
  - GLib::DFS< cur\_node, graph, visited\_nodes >::IterateThroughChildren< cur\_children, cur\_visited >, 86
  - GLib::DFS< cur\_node, graph, visited\_nodes

- [>::IterateThroughChildren< EmptyTypeList, cur\\_unvisited >, 87](#)
- [GLib::FindNodeByVertex< vertex, EmptyTypeList >, 64](#)
- [GLib::FindNodeByVertex< vertex, graph >, 64](#)
- [GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< cur\\_nodes >, 96](#)
- [GLib::FindNodeByVertex< vertex, graph >::IterateThroughNodes< EmptyTypeList >, 97](#)
- [GLib::GetReachedVertexes< graph, start >, 75](#)
- [GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< cur\\_edges >, 90](#)
- [GLib::GetReachedVertexes< graph, start >::IterateThroughEdges< EmptyTypeList >, 92](#)
- [PointerStructureGraph< nodes >::ConvertTo< type >, 56](#)
- [TL::Add< T, 0, TypeList< Arg, Args... > >, 18](#)
- [TL::Add< T, 0, TypeList< Args... > >, 18](#)
- [TL::Add< T, ind, TypeList< Arg, Args... > >, 19](#)
- [TL::Concatenate< front, back >, 44](#)
- [TL::Concatenate< front, back >::IterateThroughReversedFront< elements, current >, 98](#)
- [TL::Concatenate< front, back >::IterateThroughReversedFront< EmptyTypeList, current >, 99](#)
- [TL::Contains< type\\_list, T >, 46](#)
- [TL::ContainsConstructibleParent< EmptyTypeList, T >, 47](#)
- [TL::ContainsConstructibleParent< type\\_list, T >, 46](#)
- [TL::ContainsParent< EmptyTypeList, T >, 49](#)
- [TL::ContainsParent< type\\_list, T >, 48](#)
- [TL::FindParentTypeList< T, type\\_list, type\\_lists >, 65](#)
- [TL::FindTypeListByClass< T, type\\_list, type\\_lists >, 70](#)
- [TL::GenerateTypeLists< 0 >, 73](#)
- [TL::GenerateTypeLists< n >, 73](#)
- [TL::HasDerivedAndConstructible< EmptyTypeList, T >, 77](#)
- [TL::HasDerivedAndConstructible< type\\_list, T >, 76](#)
- [TL::IsBaseOf< EmptyTypeList, derived >, 82](#)
- [TL::IsBaseOf< EmptyTypeList, EmptyTypeList >, 83](#)
- [TL::IsBaseOf< parent, derived >, 81](#)
- [TL::IsBaseOf< parent, EmptyTypeList >, 83](#)
- [TL::MostDerived< EmptyTypeList, T >, 102](#)
- [TL::MostDerived< type\\_list, T >, 101](#)
- [TL::MostDerivedAndConstructible< EmptyTypeList, T >, 104](#)
- [TL::MostDerivedAndConstructible< type\\_list, T >, 103](#)
- [TL::NoDuplicates< EmptyTypeList >, 106](#)
- [TL::NoDuplicates< type\\_list >, 105](#)
- [TL::Remove< EmptyTypeList, T >, 110](#)
- [TL::Remove< type\\_list, T >, 110](#)
- [TL::Remove< type\\_list, typename type\\_list::Head >, 111](#)
- [TL::RemoveAll< type\\_list, T >, 112](#)
- [TL::RemoveAll< type\\_list, typename type\\_list::Head >, 113](#)
- [TL::Replace< T, 0, TypeList< Arg, Args... > >, 114](#)
- [TL::Replace< T, ind, TypeList< Arg, Args... > >, 115](#)
- [TL::Reverse< type\\_list >, 116](#)
- [TL::Reverse< type\\_list >::IterateThroughElements< cur\\_type\\_list, cur\\_result >, 94](#)
- [TL::Reverse< type\\_list >::IterateThroughElements< EmptyTypeList, cur\\_result >, 95](#)
- [reversed\\_edges](#)
- [GLib::FindPath< graph\\_raw, start, finish >, 68](#)
- [reversed\\_front](#)
- [TL::Concatenate< front, back >, 45](#)
- [reversed\\_path](#)
- [GLib::FindPath< graph\\_raw, start, finish >, 68](#)
- [reversed\\_weights](#)
- [GLib::FindPath< graph\\_raw, start, finish >, 68](#)
- [size](#)
- [TL::Size< EmptyTypeList >, 117](#)
- [TL::Size< type\\_list >, 117](#)
- [start\\_node](#)
- [GLib::FindPath< graph\\_raw, start, finish >, 68](#)
- [GLib::GetReachedVertexes< graph, start >, 75](#)
- [Stream](#)
- [Stream< T >, 118](#)
- [Stream< T >, 118](#)
- [Collect, 119](#)
- [Filter, 119](#)
- [ForEach, 119](#)
- [Map, 120](#)
- [Reduce, 120](#)
- [Stream, 118](#)
- [StreamFunctorType, 13](#)
- [BinaryOperation, 14](#)
- [Consumer, 14](#)
- [Function, 14](#)
- [Predicate, 14](#)
- [Tail](#)
- [TypeList< Args >, 123](#)
- [TypeList< H, T... >, 124](#)
- [TypeList< T >, 125](#)
- [TL, 15](#)
- [TL/add.h, 139](#)
- [TL/concatenate.h, 139](#)
- [TL/contains.h, 139](#)
- [TL/contains\\_constructible\\_parent.h, 140](#)
- [TL/contains\\_parent.h, 140](#)
- [TL/find\\_parent\\_type\\_list.h, 140](#)
- [TL/find\\_type\\_list\\_by\\_class.h, 141](#)
- [TL/generate\\_type\\_lists.h, 141](#)
- [TL/has\\_derived\\_and\\_constructible.h, 142](#)
- [TL/index\\_of.h, 142](#)
- [TL/is\\_base\\_of.h, 142](#)
- [TL/is\\_type\\_list.h, 143](#)

- TL/most\_derived.h, 143
- TL/most\_derived\_and\_constructible.h, 144
- TL/no\_duplicates.h, 144
- TL/null\_type.h, 144
- TL/remove.h, 144
- TL/replace.h, 145
- TL/reverse.h, 145
- TL/size.h, 146
- TL/type\_at.h, 146
- TL/type\_list.h, 146
- TL::Add< T, 0, TypeList< Arg, Args... > >, 17
  - result, 18
- TL::Add< T, 0, TypeList< Args... > >, 18
  - result, 18
- TL::Add< T, ind, Arg, Args >, 17
- TL::Add< T, ind, TypeList< Arg, Args... > >, 19
  - end, 19
  - result, 19
- TL::Concatenate< front, back >, 44
  - result, 44
  - reversed\_front, 45
- TL::Concatenate< front, back >::IterateThroughReversedFront
  - elements, current >, 97
  - added, 98
  - result, 98
- TL::Concatenate< front, back >::IterateThroughReversedFront
  - EmptyTypeList, current >, 98
  - result, 99
- TL::Contains< type\_list, T >, 45
  - result, 46
- TL::ContainsConstructibleParent< EmptyTypeList, T >, 47
  - result, 47
- TL::ContainsConstructibleParent< type\_list, T >, 46
  - result, 46
- TL::ContainsParent< EmptyTypeList, T >, 49
  - result, 49
- TL::ContainsParent< type\_list, T >, 48
  - result, 48
- TL::FindParentTypeList< T, type\_list, type\_lists >, 65
  - result, 65
- TL::FindTypeListByClass< T, type\_list, type\_lists >, 69
  - result, 70
- TL::GenerateTypeLists< 0 >, 73
  - result, 73
- TL::GenerateTypeLists< n >, 72
  - result, 73
- TL::HasDerivedAndConstructible< EmptyTypeList, T >, 77
  - result, 77
- TL::HasDerivedAndConstructible< type\_list, T >, 76
  - result, 76
- TL::IndexOf< EmptyTypeList, T >, 78
  - value, 79
- TL::IndexOf< type\_list, T >, 78
  - value, 78
- TL::IndexOf< type\_list, typename type\_list::Head >, 79
  - value, 80
- TL::IsBaseOf< EmptyTypeList, derived >, 82
  - result, 82
- TL::IsBaseOf< EmptyTypeList, EmptyTypeList >, 82
  - result, 83
- TL::IsBaseOf< parent, derived >, 81
  - result, 81
- TL::IsBaseOf< parent, EmptyTypeList >, 83
  - result, 83
- TL::IsTypeList< T >, 84
- TL::IsTypeList< TypeList< Args... > >, 84
- TL::MostDerived< EmptyTypeList, T >, 102
  - result, 102
- TL::MostDerived< type\_list, T >, 101
  - result, 101
- TL::MostDerivedAndConstructible< EmptyTypeList, T >, 104
  - result, 104
- TL::MostDerivedAndConstructible< type\_list, T >, 103
  - result, 103
- TL::NoDuplicates< EmptyTypeList >, 105
  - result, 106
- TL::NoDuplicates< type\_list >, 104
  - result, 105
- TL::Remove< EmptyTypeList, T >, 110
  - result, 110
- TL::Remove< type\_list, T >, 109
  - result, 110
- TL::Remove< type\_list, typename type\_list::Head >, 111
  - result, 111
- TL::RemoveAll< type\_list, T >, 111
  - result, 112
- TL::RemoveAll< type\_list, typename type\_list::Head >, 112
  - result, 113
- TL::Replace< T, 0, TypeList< Arg, Args... > >, 113
  - result, 114
- TL::Replace< T, ind, Arg, Args >, 113
- TL::Replace< T, ind, TypeList< Arg, Args... > >, 114
  - end, 115
  - result, 115
- TL::Reverse< type\_list >, 115
  - result, 116
- TL::Reverse< type\_list >::IterateThroughElements<
  - cur\_type\_list, cur\_result >, 94
  - result, 94
- TL::Reverse< type\_list >::IterateThroughElements<
  - EmptyTypeList, cur\_result >, 95
  - result, 95
- TL::Size< EmptyTypeList >, 117
  - size, 117
- TL::Size< type\_list >, 116
  - size, 117
- TL::TypeAt< type\_list, 0 >, 122
  - value, 122
- TL::TypeAt< type\_list, ind >, 121
  - value, 121
- to



- Edge< from\_, to\_, weight\_ >, [61](#)
- TYPE
  - AdjacencyListGraph< nodes, adjacency\_list >, [25](#)
  - AdjacencyMatrixGraph< nodes, matrix >, [27](#)
  - EdgeListGraph< nodes, edge\_list >, [63](#)
  - PointerStructureGraph< nodes >, [107](#)
- type\_list.h
  - EmptyTypeList, [147](#)
  - Typelist, [147](#)
- type\_list\_without\_first
  - ConvertGraph< ADJACENCY\_LIST, POINTER\_STRUCTURE, graph >::MakePointerStructureGraph< current\_vertexes, current\_adjacency\_list >, [100](#)
- Typelist
  - type\_list.h, [147](#)
- TypeList< Args >, [123](#)
  - Head, [123](#)
  - Tail, [123](#)
- TypeList< H, T... >, [124](#)
  - Head, [124](#)
  - Tail, [124](#)
- TypeList< T >, [125](#)
  - Head, [125](#)
  - Tail, [125](#)
- upd\_visited
  - GLib::DFS< cur\_node, graph, visited\_nodes >, [59](#)
- value
  - Objects::Boolean< boolean >, [27](#)
  - Objects::Integer< integer >, [80](#)
  - TL::IndexOf< EmptyTypeList, T >, [79](#)
  - TL::IndexOf< type\_list, T >, [78](#)
  - TL::IndexOf< type\_list, typename type\_list::Head >, [80](#)
  - TL::TypeAt< type\_list, 0 >, [122](#)
  - TL::TypeAt< type\_list, ind >, [121](#)
- vertex
  - PointerStructureNode< vertex\_, children\_ >, [109](#)
- vertex\_num
  - GLib::AddEdge< ADJACENCY\_LIST, graph, edge >, [22](#)
- vertex\_stream\_example.cpp
  - Add1, [130](#)
  - IsAmongFirst3, [130](#)
  - main, [131](#)
- vertexes\_
  - AdjacencyListGraph< nodes, adjacency\_list >, [24](#)
  - AdjacencyMatrixGraph< nodes, matrix >, [26](#)
  - EdgeListGraph< nodes, edge\_list >, [62](#)
- VertexStream< EmptyTypeList, graph >, [127](#)
  - MapVertexesToReversedIndexes, [127](#)
- VertexStream< stream, graph >, [126](#)
  - MapVertexesToIndexes, [126](#)
  - MapVertexesToReversedIndexes, [126](#)
- weight
  - Edge< from\_, to\_, weight\_ >, [61](#)
- weights