Vienna Graph Template Library

Version 1.0

Reference Manual

Hermann Schichl
University of Vienna, Department of Mathematics
A-1090 Wien, Austria
email: Hermann.Schichl@esi.ac.at

Generated by Doxygen 1.2.14

Contents

1 Introduction

The Vienna Graph Template Library (VGTL) is a generic graph library with generic programming structure. It uses STL containers like `map` and `vector` to organize the internal structure of the graphs.

A collection of walking algorithms for analyzing and working with the graphs has been implemented as generic algorithms. Similar to STL iterators, which are used to handle data in containers independently of the container implementation, for graphs the walker concept (see Section Walker) is introduced.

1.1 Walker

A **walker** is, like an STL iterator, a generalization of a pointer. It dereferences to the data a graph node stores.

There are two different kinds of walkers: **recursive** walker and **iterative** walker.

1.1.1 Recursive Walker

A recursive walker is a pointer to graph nodes, which can be moved around on the graph by changing the node it points to. Walkers can move along the edges of the graph to new nodes. The operators reserved for that are $<<$ for moving along in-edges and $>>$ for moving along out-edges. A recursive walker does not have an internal status, so the walking has to be done recursively.

1.1.2 Iterative Walker

An iterative walker (automatic walker) can walk through a graph without guidance. Simply using the operators `++` and `--`, the walker itself searches for the next node in the walk.

1.2 Trees and Forests

The first few of the collection of graph containers are the n-ary trees and forests. These trees come in various flavors: standard trees, labelled trees, with and without data hooks. Trees provide iterative walkers and recursive walkers.

1.3 Directed Graphs and DAGs

The next more complicated graphs are **directed graphs. There are two classes implemented. Standard directed graphs and directed acyclic graphs (DAGs). Directed graphs provide recursive walkers only.**

1.4 Generic Graphs

Generic graphs don't have directed edges. They are the most general class of graphs, and special walking algorithms are provided for them. Generic graphs only have recursive walkers.

2 Vienna Graph Template Library Module Index

2.1 Vienna Graph Template Library Modules

Here is a list of all modules:

3 Vienna Graph Template Library Hierarchical Index

3.1 Vienna Graph Template Library Class Hierarchy

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

__Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc > 68

_Tree_base< _Tp, _Ctr, _Iterator, _Node, _Alloc > 115

__Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc > 74

_Tree_alloc_base< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator, _ITree_node< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator >, _Alloc_traits< _Key, _ITree_node< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator >>::_S_instanceless >

_Tree_base< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator, _ITree_node< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator >, _Alloc > 115

__Tree_t< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator, _Key &, _ITree_node< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator >, _Alloc > 74

__ITree< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator, _Key &, _Alloc > 61

stree< _Key, _Compare, _AssocCtr, _PtrAlloc, _Alloc > 185

_Tree_alloc_base< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator, _Node, _ITree_node< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator >, std::_Alloc_traits< _Key, _ITree_node< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator >>::_S_instanceless >

_Tree_base< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator, _ITree_node< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator >, _Alloc > 115

_Tree_alloc_base< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator, _Node, _Tree_node< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator >, std::_Alloc_traits< _Key, _Tree_node< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator >>::_S_instanceless >

_Tree_base< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator, _Tree_node< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator >, _Alloc > 115

__Tree_t< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator, _Key &, _Tree_node< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator >,

4 Vienna Graph Template Library Compound Index

4.1 Vienna Graph Template Library Compound List

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

5 Vienna Graph Template Library File Index

5.1 Vienna Graph Template Library File List

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

6 Vienna Graph Template Library Module Documentation

6.1 Classes and types for external use

Compounds

- class **array_vector**
  STL vector wrapper for C *array.*
- class **atree**
  n-ary forest with labelled edges.
- class **dag**
  unlabeled directed acyclic graph (DAG).
- class **dgraph**
  unlabeled directed graph.
- class **ntree**
  n-ary forest.
- class **postorder_visitor**
  postorder visitor base class.
- class **preorder_visitor**
  preorder visitor base class.
- class **prepost_visitor**
  pre+postorder visitor base class.
- class **ratree**
  n-ary forest with labelled edges.

- **class rntree**

  n-ary forest.

- **class rstree**

  n-ary forest with unsorted edges.

- **class stree**

  n-ary forest with unsorted edges.

6.1.1 Detailed Description

The classes and types in this section are for external use.

6.2 Generic algorithms for external use

Functions

- **template<class _IterativeWalker, class _Function> _Function walk (_IterativeWalker __first, _IterativeWalker __last, _Function __f)**
- **template<class _PrePostWalker, class _Function> _Function pre_post_walk (_PrePostWalker __first, _PrePostWalker __last, _Function __f)**
- **template<class _PrePostWalker, class _Function1, class _Function2> _Function2 pre_post_walk (_PrePostWalker __first, _PrePostWalker __last, _Function1 __f1, _Function2 __f2)**
- **template<class _PrePostWalker, class _Function> _Function var_walk (_PrePostWalker __first, _PrePostWalker __last, _Function __f)**
- **template<class _PrePostWalker, class _Function1, class _Function2> _Function2 var_walk (_PrePostWalker __first, _PrePostWalker __last, _Function1 __f1, _Function2 __f2)**
- **template<class _PrePostWalker, class _Function, class _Predicate> _Function walk_if (_PrePostWalker __first, _PrePostWalker __last, _Function __f, _Predicate __pred)**
- **template<class _PrePostWalker, class _Function1, class _Function2, class _Predicate> _Function2 walk_if (_PrePostWalker __first, _PrePostWalker __last, _Function1 __f1, _Function2 __f2, _Predicate __pred)**
- **template<class _PrePostWalker, class _Function1, class _Function2, class _Predicate1, class _Predicate2> _Function2 walk_if (_PrePostWalker __first, _PrePostWalker __last, _Function1 __f1, _Function2 __f2, _Predicate1 __pred1, _Predicate2 __pred2)**
- **template<class _PrePostWalker, class _Function1, class _Function2, class _Predicate> _Function2 cached_walk_if (_PrePostWalker __first, _PrePostWalker __last, _Function1 __f1, _Function2 __f2, _Predicate __pred)**
- **template<class _PrePostWalker, class _Function1, class _Function2, class _Predicate> _Function2 multi_walk_if (_PrePostWalker __first, _PrePostWalker __last, _Function1 __f1, _Function2 __f2, _Predicate __pred)**
- **template<class _Walker, class _Function> _Function walk_up (_Walker __w, _Function __f)**
- **template<class _Walker, class _Function> _Function var_walk_up (_Walker __w, _Function __f)**
- **template<class _Walker, class _Function, class _Predicate> _Function walk_up_if (_Walker __w, _Function __f, _Predicate __p)**
- **template<class _Walker, class _Visitor> _Visitor::return_value recursive_preorder_walk (_Walker __w, _Visitor __f)**
- **template<class _Walker, class _Visitor> _Visitor::return_value recursive_postorder_walk (_Walker __w, _Visitor __f)**

- template<class _Walker, class _Visitor> _Visitor::return_value recursive_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_preorder_walk_if (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_preorder_walk_if (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_postorder_walk_if (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_walk_if (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_cached_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_multi_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor, class _Predicate1, class _Predicate2> _Visitor::return_value recursive_walk_if (_Walker __w, _Visitor __f, _Predicate1 __p1, _Predicate2 __p2)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_cached_walk (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_multi_walk (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_preorder_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_preorder_walk_up_if (_Walker __w, _Visitor __f)
- ∗recursive walk towards the root node (i.e.come back!) template< class _Walker
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_postorder_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_postorder_walk_up_if (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_walk_up_if (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor, class _Predicate1, class _Predicate2> _Visitor::return_value recursive_walk_up_if (_Walker __w, _Visitor __f, _Predicate1 __p1, _Predicate2 __p2)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_cached_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_multi_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_cached_walk_up (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_multi_walk_up (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor> _Visitor::return_value general_directed_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value general_directed_walk_down (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value general_directed_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_general_directed_walk (_Walker __w, _Visitor __f)

- template<class _Walker, class _Visitor> _Visitor::return_value recursive_general_directed_walk_down (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_general_directed_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value general_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_general_walk (_Walker __w, _Visitor __f)
- template<class _BidirIter, class _Tp> _BidirIter rfind (_BidirIter __first, _BidirIter __last, const _Tp &__val)
- template<class _BidirIter, class _Predicate> _BidirIter rfind_if (_BidirIter __first, _BidirIter __last, _Predicate __pred)

6.2.1 Detailed Description

The generic functions in this section are for external use.

6.2.2 Function Documentation

6.2.2.1 template<class _PrePostWalker, class _Function1, class _Function2, class _Predicate> _Function2 cached_walk_if (_PrePostWalker __*first*, _PrePostWalker __*last*, _Function1 __*f1*, _Function2 __*f2*, _Predicate __*pred*)

this tree walk is a pre+post walk, calling two functions at every node, one in the preorder and the other in the postorder visit. If the function returns true, the status of the walker is flipped from pre to post (or vice versa). If the status is changed from pre to post, the subtree originating from the current position is not visited, if the status change is the other way round, it is revisited. This allows for cached or partially multi pass walks.

Definition at line 393 of file vgtl_algo.h.

6.2.2.2 template<class _Walker, class _Visitor> _Visitor::return_value general_directed_walk (_Walker __*w*, _Visitor __*f*)

perform a general directed walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `analyze` is called before walking for every virtual node. While this function returns `true`, the walk goes on.
- `preorder` is called before a walk direction is being decided.
- `postorder` is called after the walk dircetion has been found.
- `walk_up` shall return whether the next step of the walk is upwards or downwards.
- `up` is called for an upwards step and decides which in-edge to take.
- `down` is called for a downwards step and decides which out-edge to take.
- `value` is called to compute the return value for this node.

Definition at line 2390 of file vgtl_algo.h.

6.2.2.3 template<class _Walker, class _Visitor> _Visitor::return_value general_directed_walk_down (_Walker __*w*, _Visitor __*f*)

perform a general directed walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `analyze` is called before walking for every virtual node. While this function returns `true`, the walk goes on.
- `preorder` is called before a walk direction is being decided.
- `postorder` is called after the walk dircetion has been found.
- `down` is called to decide which out-edge to take.
- `value` is called to compute the return value for this node.

Definition at line 2419 of file vgtl_algo.h.

6.2.2.4 template<class _Walker, class _Visitor> _Visitor::return_value general_directed_walk_up (_Walker __w, _Visitor __f)

perform a general directed walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `analyze` is called before walking for every virtual node. While this function returns `true`, the walk goes on.
- `preorder` is called before a walk direction is being decided.
- `postorder` is called after the walk dircetion has been found.
- `up` is called to decide which in-edge to take.
- `value` is called to compute the return value for this node.

Definition at line 2446 of file vgtl_algo.h.

6.2.2.5 template<class _Walker, class _Visitor> _Visitor::return_value general_walk (_Walker __w, _Visitor __f)

perform a general walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `analyze` is called before walking for every virtual node. While this function returns `true`, the walk goes on.
- `preorder` is called before a walk direction is being decided.
- `postorder` is called after the walk dircetion has been found.
- `next` is called to decide which edge to follow.
- `value` is called to compute the return value for this node.

Definition at line 2558 of file vgtl_algo.h.

6.2.2.6 template<class _PrePostWalker, class _Function1, class _Function2, class _Predicate> _Function2 multi_walk_if (_PrePostWalker __*first*, _PrePostWalker __*last*, _Function1 __*f1*, _Function2 __*f2*, _Predicate __*pred*)

this tree walk is a pre+post walk, calling two functions at every node, one in the preorder and the other in the postorder visit. If the function returns true, the status of the walker is flipped from pre to post (or vice versa). If the status is changed from pre to post, the subtree originating from the current position is not visited, if the status change is the other way round, it is revisited. This allows for cached or partially multi pass walks.

Definition at line 426 of file vgtl_algo.h.

6.2.2.7 ∗ recursive walk towards the root node (i.e.come back!)

perform a recursive preorder walk towards the root starting at __w. At every node various methods of the visitor __f are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited. Then the predicate is called. If this predicate returns** `true`**, the children are visited. Otherwise, the node is treated as if it was a terminal node.**
- `collect` **is called everytime a child has finished**
- `value` **is called to compute the return value for this node**

6.2.2.8 template<class _PrePostWalker, class _Function1, class _Function2> _Function2 pre_post_walk (_PrePostWalker __*first*, _PrePostWalker __*last*, _Function1 __*f1*, _Function2 __*f2*)

make a pre and post order tree walk, calling two different functions, one in the preorder step, the other in the postorder step.

Definition at line 223 of file vgtl_algo.h.

6.2.2.9 template<class _PrePostWalker, class _Function> _Function pre_post_walk (_PrePostWalker __*first*, _PrePostWalker __*last*, _Function __*f*)

make a pre and post order tree walk, calling a function for every node.

Definition at line 205 of file vgtl_algo.h.

6.2.2.10 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_cached_walk (_Walker __*w*, _Visitor __*f*, _Predicate __*p*)

perform a recursive pre+post order walk starting at __w. At every node various methods of the visitor __f are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited. If then predicate __p returns** `true`**, the children are visited. If it returns** `false`**, the children are ignored**
- `collect` **is called everytime a child has finished**
- `postorder` **is called after the children have been visited.**
- `value` **is called to compute the return value for this node**

Definition at line 1296 of file vgtl_algo.h.

6.2.2.11 template<class _Walker, class _Visitor> _Visitor::return_value recursive_cached_walk (_Walker __*w*, _Visitor __*f*)

perform a recursive pre+post order walk starting at __w. At every node various methods of the visitor __f are called:

- `vinit` **is called before walking for every virtual node**

- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. If it returns `true`, the children are visited. If it returns `false`, the children are ignored
- `collect` is called everytime a child has finished
- `postorder` is called after the children have been visited.
- `value` is called to compute the return value for this node

Definition at line 1047 of file vgtl_algo.h.

6.2.2.12 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_cached_walk_up (_Walker __w, _Visitor __f, _Predicate __p)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. If then predicate `__p` returns `true`, the children are visited. If it returns `false`, the children are ignored
- `collect` is called everytime a child has finished
- `postorder` is called after the children have been visited.
- `value` is called to compute the return value for this node

Definition at line 2224 of file vgtl_algo.h.

6.2.2.13 template<class _Walker, class _Visitor> _Visitor::return_value recursive_cached_walk_up (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. If it returns `true`, the children are visited. If it returns `false`, the children are ignored
- `collect` is called everytime a child has finished
- `postorder` is called after the children have been visited.
- `value` is called to compute the return value for this node

Definition at line 2066 of file vgtl_algo.h.

6.2.2.14 template<class _Walker, class _Visitor> _Visitor::return_value recursive_general_directed_walk (_Walker __w, _Visitor __f)

perform a recursive general directed walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `preorder` is called before any child is visited

- `analyze` is called everytime before a child node might be vistied. While this function returns `true`, the walk goes on at this node.
- `collect` is called everytime a child has finished.
- `postorder` is called after the walk dircetion has been found.
- `walk_up` shall return whether the next step of the walk is upwards or downwards.
- `up` is called for an upwards step and decides which in-edge to take.
- `down` is called for a downwards step and decides which out-edge to take.
- `value` is called to compute the return value for this node.

Definition at line 2479 of file vgtl_algo.h.

6.2.2.15 template<class _Walker, class _Visitor> _Visitor::return_value recursive_general_directed_walk_down (_Walker __w, _Visitor __f)

perform a recursive general directed walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `preorder` is called before any child is visited
- `analyze` is called everytime before a child node might be vistied. While this function returns `true`, the walk goes on at this node.
- `collect` is called everytime a child has finished.
- `postorder` is called after the walk dircetion has been found.
- `down` is called to decide which out-edge to take.
- `value` is called to compute the return value for this node.

Definition at line 2509 of file vgtl_algo.h.

6.2.2.16 template<class _Walker, class _Visitor> _Visitor::return_value recursive_general_directed_walk_up (_Walker __w, _Visitor __f)

perform a recursive general directed walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `preorder` is called before any child is visited
- `analyze` is called everytime before a child node might be vistied. While this function returns `true`, the walk goes on at this node.
- `collect` is called everytime a child has finished.
- `postorder` is called after the walk dircetion has been found.
- `up` is called to decide which in-edge to take.
- `value` is called to compute the return value for this node.

Definition at line 2534 of file vgtl_algo.h.

6.2.2.17 template<class _Walker, class _Visitor> _Visitor::return_value recursive_general_walk (_Walker __w, _Visitor __f)

perform a recursive general walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `preorder` is called before any child is visited
- `analyze` is called everytime before a child node might be vistied. While this function returns `true`, the walk goes on at this node.

- `collect` is called everytime a child has finished.
- `postorder` is called after the walk dircetion has been found.
- `next` is called to decide which edge to follow.
- `value` is called to compute the return value for this node.

Definition at line 2585 of file vgtl_algo.h.

6.2.2.18 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_multi_walk (_Walker __w, _Visitor __f, _Predicate __p)

perform a recursive pre+post order walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node.
- `vcollect` is called after a child of a virtual node has finished.
- `vvalue` is called to compute the return value of a virtual node.
- `preorder` is called before the children are visited.
- `collect` is called everytime a child has finished.
- `postorder` is called after the children have been visited. If the predicate `__p` returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.
- `value` is called to compute the return value for this node.

Definition at line 1375 of file vgtl_algo.h.

6.2.2.19 template<class _Walker, class _Visitor> _Visitor::return_value recursive_multi_walk (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node.
- `vcollect` is called after a child of a virtual node has finished.
- `vvalue` is called to compute the return value of a virtual node.
- `preorder` is called before the children are visited.
- `collect` is called everytime a child has finished.
- `postorder` is called after the children have been visited. If it returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.
- `value` is called to compute the return value for this node.

Definition at line 1123 of file vgtl_algo.h.

6.2.2.20 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_multi_walk_up (_Walker __w, _Visitor __f, _Predicate __p)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node.
- `vcollect` is called after a child of a virtual node has finished.

- `vvalue` is called to compute the return value of a virtual node.
- `preorder` is called before the children are visited.
- `collect` is called everytime a child has finished.
- `postorder` is called after the children have been visited. If the predicate `__p` returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.
- `value` is called to compute the return value for this node.

Definition at line 2303 of file vgtl_algo.h.

6.2.2.21 template<class _Walker, class _Visitor> _Visitor::return_value recursive_multi_walk_up (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node.
- `vcollect` is called after a child of a virtual node has finished.
- `vvalue` is called to compute the return value of a virtual node.
- `preorder` is called before the children are visited.
- `collect` is called everytime a child has finished.
- `postorder` is called after the children have been visited. If it returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.
- `value` is called to compute the return value for this node.

Definition at line 2143 of file vgtl_algo.h.

6.2.2.22 template<class _Walker, class _Visitor> _Visitor::return_value recursive_postorder_walk (_Walker __w, _Visitor __f)

perform a recursive postorder walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `init` is called before the children are visited
- `collect` is called everytime a child has finished
- `postorder` is called after all children have finished
- `value` is called to compute the return value for this node

Definition at line 595 of file vgtl_algo.h.

6.2.2.23 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_postorder_walk_if (_Walker __w, _Visitor __f, _Predicate __p)

perform a recursive postorder walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node

- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `init` is called before the children are visited. Then the predicate is called. If this predicate returns `true`, the children are visited. Otherwise, the node is treated as if it was a terminal node.
- `postorder` is called after all children have been visited.
- `collect` is called everytime a child has finished.
- `value` is called to compute the return value for this node.

Definition at line 880 of file vgtl_algo.h.

6.2.2.24 template<class _Walker, class _Visitor> _Visitor::return_value recursive_postorder_walk_up (_Walker __w, _Visitor __f)

perform a recursive postorder walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `init` is called before the children are visited
- `collect` is called everytime a child has finished
- `postorder` is called after all children have finished
- `value` is called to compute the return value for this node

Definition at line 1669 of file vgtl_algo.h.

6.2.2.25 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_postorder_walk_up_if (_Walker __w, _Visitor __f, _Predicate __p)

perform a recursive postorder walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `init` is called before the children are visited. Then the predicate is called. If this predicate returns `true`, the children are visited. Otherwise, the node is treated as if it was a terminal node.
- `postorder` is called after all children have been visited.
- `collect` is called everytime a child has finished.
- `value` is called to compute the return value for this node.

Definition at line 1740 of file vgtl_algo.h.

6.2.2.26 template<class _Walker, class _Visitor> _Visitor::return_value recursive_preorder_walk (_Walker __w, _Visitor __f)

perform a recursive preorder walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node

- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited
- `collect` is called everytime a child has finished
- `value` is called to compute the return value for this node

Definition at line 530 of file vgtl_algo.h.

6.2.2.27 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_preorder_walk_if (_Walker __w, _Visitor __f, _Predicate __p)

perform a recursive preorder walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. Then the predicate is called. If this predicate returns `true`, the children are visited. Otherwise, the node is treated as if it was a terminal node.
- `collect` is called everytime a child has finished
- `value` is called to compute the return value for this node

Definition at line 803 of file vgtl_algo.h.

6.2.2.28 template<class _Walker, class _Visitor> _Visitor::return_value recursive_preorder_walk_if (_Walker __w, _Visitor __f)

perform a recursive preorder walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. If this function returns `true`, the children are visited. Otherwise, the node is treated as if it was a terminal node.
- `collect` is called everytime a child has finished
- `value` is called to compute the return value for this node

Definition at line 730 of file vgtl_algo.h.

6.2.2.29 template<class _Walker, class _Visitor> _Visitor::return_value recursive_preorder_walk_up (_Walker __w, _Visitor __f)

perform a recursive preorder walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited

- `collect` is called everytime a child has finished
- `value` is called to compute the return value for this node

Definition at line 1455 of file vgtl_algo.h.

6.2.2.30 template<class _Walker, class _Visitor> _Visitor::return_value recursive_preorder_walk_up_if (_Walker __w, _Visitor __f)

perform a recursive preorder walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. If this function returns `true`, the children are visited. Otherwise, the node is treated as if it was a terminal node.
- `collect` is called everytime a child has finished
- `value` is called to compute the return value for this node

Definition at line 1521 of file vgtl_algo.h.

6.2.2.31 template<class _Walker, class _Visitor> _Visitor::return_value recursive_walk (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited
- `collect` is called everytime a child has finished
- `postorder` is called after all children have been visited
- `value` is called to compute the return value for this node

Definition at line 663 of file vgtl_algo.h.

6.2.2.32 template<class _Walker, class _Visitor, class _Predicate1, class _Predicate2> _Visitor::return_value recursive_walk_if (_Walker __w, _Visitor __f, _Predicate1 __p1, _Predicate2 __p2)

perform a recursive pre+post order walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. If then predicate `p1` returns `true`, the children are visited. If it returns `false`, the children are ignored
- `collect` is called everytime a child has finished

- `postorder` is called after the children have been visited. If then predicate `p2` returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.
- `value` is called to compute the return value for this node

Definition at line 1205 of file vgtl_algo.h.

6.2.2.33 template<class _Walker, class _Visitor> _Visitor::return_value recursive_walk_if (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node.
- `vcollect` is called after a child of a virtual node has finished.
- `vvalue` is called to compute the return value of a virtual node.
- `preorder` is called before the children are visited. If it returns `true`, the children are visited. If it returns `false`, the children are ignored.
- `collect` is called everytime a child has finished.
- `postorder` is called after the children have been visited. If it returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.
- `value` is called to compute the return value for this node.

Definition at line 962 of file vgtl_algo.h.

6.2.2.34 template<class _Walker, class _Visitor> _Visitor::return_value recursive_walk_up (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited
- `collect` is called everytime a child has finished
- `postorder` is called after all children have been visited
- `value` is called to compute the return value for this node

Definition at line 1816 of file vgtl_algo.h.

6.2.2.35 template<class _Walker, class _Visitor, class _Predicate1, class _Predicate2> _Visitor::return_value recursive_walk_up_if (_Walker __w, _Visitor __f, _Predicate1 __p1, _Predicate2 __p2)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished

- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. If then predicate `p1` returns `true`, the children are visited. If it returns `false`, the children are ignored
- `collect` is called everytime a child has finished
- `postorder` is called after the children have been visited. If then predicate `p2` returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.
- `value` is called to compute the return value for this node

Definition at line 1975 of file vgtl_algo.h.

6.2.2.36 template<class _Walker, class _Visitor> _Visitor::return_value recursive_walk_up_if (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node.
- `vcollect` is called after a child of a virtual node has finished.
- `vvalue` is called to compute the return value of a virtual node.
- `preorder` is called before the children are visited. If it returns `true`, the children are visited. If it returns `false`, the children are ignored.
- `collect` is called everytime a child has finished.
- `postorder` is called after the children have been visited. If it returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.
- `value` is called to compute the return value for this node.

Definition at line 1887 of file vgtl_algo.h.

6.2.2.37 template<class _BidirIter, class _Tp> _BidirIter rfind (_BidirIter __first, _BidirIter __last, const _Tp & __val) `[inline]`

Find the last occurrence of a value in a sequence.

Parameters:

__first An input iterator.

__last An input iterator.

__val The value to find.

Returns:

The last iterator `i` in the range `[__first, __last)` such that `*i == val`, or `__last` if no such iterator exists.

Definition at line 191 of file vgtl_helpers.h.

6.2.2.38 template<class _BidirIter, class _Predicate> _BidirIter rfind_if (_BidirIter __first, _BidirIter __last, _Predicate __pred) `[inline]`

Find the last element in a sequence for which a predicate is true.

Parameters:

__first An input iterator.

__last An input iterator.

__pred A predicate.

Returns:

The last iterator `i` in the range `[__first, __last)` such that `__pred(*i)` is true, or `__last` if no such iterator exists.

Definition at line 207 of file vgtl_helpers.h.

6.2.2.39 template<class _PrePostWalker, class _Function1, class _Function2> _Function2 var_walk (_PrePostWalker *__first*, _PrePostWalker *__last*, _Function1 *__f1*, _Function2 *__f2*)

this tree walk is a pre+post walk, calling two functions at every node, one in the preorder and the other in the postorder step. If the function returns true, the status of the walker is flipped from pre to post (or vice versa). If the status is changed from pre to post, the subtree originating from the current position is not visited, if the status change is the other way round, it is revisited. This allows for cached or partially multi pass walks.

Definition at line 270 of file vgtl_algo.h.

6.2.2.40 template<class _PrePostWalker, class _Function> _Function var_walk (_PrePostWalker *__first*, _PrePostWalker *__last*, _Function *__f*)

this tree walk is a pre+post walk, calling a function at every node. If the function returns `true`, the status of the walker is flipped from pre to post (or vice versa). If the status is changed from pre to post, the subtree originating from the current position is not visited, if the status change is the other way round, it is revisited. This allows for cached or partially multi pass walks.

Definition at line 247 of file vgtl_algo.h.

6.2.2.41 template<class _Walker, class _Function> _Function var_walk_up (_Walker *__w*, _Function *__f*)

this tree walk is a pre+post walk towards the root, calling a function at every node. If the function returns `true`, the status of the walker is flipped from pre to post (or vice versa). If the status is changed from pre to post, the subtree originating from the current position is not visited, if the status change is the other way round, it is revisited. This allows for cached or partially multi pass walks.

Definition at line 475 of file vgtl_algo.h.

6.2.2.42 template<class _IterativeWalker, class _Function> _Function walk (_IterativeWalker *__first*, _IterativeWalker *__last*, _Function *__f*)

make a pre or post order tree walk, calling a function for every node it is also possible to perform a pre+post order walk. In that case the function `_f` must distinguish between the two calls by itself.

Definition at line 190 of file vgtl_algo.h.

6.2.2.43 template<class _PrePostWalker, class _Function1, class _Function2, class _Predicate1, class _Predicate2> _Function2 walk_if (_PrePostWalker *__first*, _PrePostWalker *__last*, _Function1 *__f1*, _Function2 *__f2*, _Predicate1 *__pred1*, _Predicate2 *__pred2*)

this tree walk is a pre+post walk, calling two functions at every node, one in the preorder and the other in the postorder visit. If the predicates return true, the status of the walker is flipped from pre to post (or vice versa). If the status is changed from pre to post, the subtree originating from the current position is not visited, if the status change is the other way round, it is revisited. This allows for cached or partially multi pass walks. Predicate pred1 is called in the preorder phase, predicate pred2 in the postorder phase.

Definition at line 355 of file vgtl_algo.h.

6.2.2.44 template<class _PrePostWalker, class _Function1, class _Function2, class _Predicate> _Function2 walk_if (_PrePostWalker __*first*, _PrePostWalker __*last*, _Function1 __*f1*, _Function2 __*f2*, _Predicate __*pred*)

this tree walk is a pre+post walk, calling two functions at every node, one in the preorder and the other in the postorder visit. If the predicate returns true, the status of the walker is flipped from pre to post (or vice versa). If the status is changed from pre to post, the subtree originating from the current position is not visited, if the status change is the other way round, it is revisited. This allows for cached or partially multi pass walks.

Definition at line 322 of file vgtl_algo.h.

6.2.2.45 template<class _PrePostWalker, class _Function, class _Predicate> _Function walk_if (_PrePostWalker __*first*, _PrePostWalker __*last*, _Function __*f*, _Predicate __*pred*)

this tree walk is a pre+post walk, calling a function at every node. If the predicate returns true, the status of the walker is flipped from pre to post (or vice versa). If the status is changed from pre to post, the subtree originating from the current position is not visited, if the status change is the other way round, it is revisited. This allows for cached or partially multi pass walks.

Definition at line 295 of file vgtl_algo.h.

6.2.2.46 template<class _Walker, class _Function> _Function walk_up (_Walker __*w*, _Function __*f*)

make a pre or post order tree walk towards the root node, calling a function for every node it is also possible to perform a pre+post order walk. In that case the function `_f` must distinguish between the two calls by itself.

Definition at line 455 of file vgtl_algo.h.

6.2.2.47 template<class _Walker, class _Function, class _Predicate> _Function walk_up_if (_Walker __*w*, _Function __*f*, _Predicate __*p*)

this tree walk is a pre+post walk towards the root, calling a function at every node. If the predicate returns true, the status of the walker is flipped from pre to post (or vice versa). If the status is changed from pre to post, the subtree originating from the current position is not visited, if the status change is the other way round, it is revisited. This allows for cached or partially multi pass walks.

Definition at line 496 of file vgtl_algo.h.

6.3 Classes and types for internal use

Compounds

- class __DG

Directed graph base class.

- class __ITree

  Tree base class with data hooks.

- class __one_iterator

  make an iterator out of one pointer.

- class __Tree

  Tree base class without data hooks.

- class __Tree_t

  Tree base class.

- class _DG_alloc_base

  Directed graph base class for general standard-conforming allocators.

- class _DG_alloc_base< _Tp, _Ctr, _I, _Allocator, true >

  Directed graph base class specialization for instanceless allocators.

- class _DG_base

  Directed graph base class for allocator encapsulation.

- class _DG_iterator

  iterator through the directed graph.

- class _DG_node

  directed graph node.

- class _DG_walker

  recursive directed graph walkers.

- class _G_compare_adaptor

  Adaptor for data comparison in graph nodes.

- class _ITree_node

  tree node for trees with data hooks.

- class _RTree_walker

  recursive tree walkers.

- class _Tree_alloc_base

  Tree base class for general standard-conforming allocators.

- class _Tree_alloc_base< _Tp, _Ctr, _I, _Node, _Allocator, true >

  Tree base class specialization for instanceless allocators.

- class _Tree_base

  Tree base class for allocator encapsulation.

- class _Tree_iterator

  iterator through the tree.

- class _Tree_node

  tree node for trees w/o data hooks.

- class _Tree_walker

  automatic tree walkers.

- class _Tree_walker_base

  base class for all tree walkers.

- class child_data_iterator

  Iterator which iterates through the data hooks of all children.

- class pair_adaptor

  adaptor for an iterator over a pair to an iterator returning the second element.

- class pointer_adaptor

  adaptor transforming a comparison predicate to pointers.

6.3.1 Detailed Description

The classes and types in this section are used VDBL internally.

6.4 Generic algorithms for internal use

Functions

- **template<class _Walker, class _Visitor> _Visitor::return_value _recursive_preorder_walk (_Walker __w, _Visitor __f)**
- **template<class _Walker, class _Visitor> _Visitor::return_value _recursive_postorder_walk (_Walker __w, _Visitor __f)**
- **template<class _Walker, class _Visitor> _Visitor::return_value _recursive_walk (_Walker __w, _Visitor __f)**
- **template<class _Walker, class _Visitor> _Visitor::return_value _recursive_preorder_walk_if (_Walker __w, _Visitor __f)**
- **template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_preorder_walk_if (_Walker __w, _Visitor __f, _Predicate __p)**
- **template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_postorder_walk_if (_Walker __w, _Visitor __f, _Predicate __p)**
- **template<class _Walker, class _Visitor> _Visitor::return_value _recursive_walk_if (_Walker __w, _Visitor __f)**
- **template<class _Walker, class _Visitor> _Visitor::return_value _recursive_cached_walk (_Walker __w, _Visitor __f)**
- **template<class _Walker, class _Visitor> _Visitor::return_value _recursive_multi_walk (_Walker __w, _Visitor __f)**
- **template<class _Walker, class _Visitor, class _Predicate1, class _Predicate2> _Visitor::return_value _recursive_walk_if (_Walker __w, _Visitor __f, _Predicate1 __p1, _Predicate2 __p2)**

- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_cached_walk (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_multi_walk (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_preorder_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_preorder_walk_up_if (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_postorder_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_postorder_walk_up_if (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_walk_up_if (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor, class _Predicate1, class _Predicate2> _Visitor::return_value _recursive_walk_up_if (_Walker __w, _Visitor __f, _Predicate1 __p1, _Predicate2 __p2)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_cached_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_multi_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_cached_walk_up (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_multi_walk_up (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _BidirIter, class _Tp> _BidirIter rfind (_BidirIter __first, _BidirIter __last, const _Tp &__val, std::bidirectional_iterator_tag)
- template<class _BidirIter, class _Predicate> _BidirIter rfind_if (_BidirIter __first, _BidirIter __last, _Predicate __pred, std::bidirectional_iterator_tag)
- template<class _RandomAccessIter, class _Tp> _RandomAccessIter rfind (_RandomAccessIter __first, _RandomAccessIter __last, const _Tp &__val, std::random_access_iterator_tag)
- template<class _RandomAccessIter, class _Predicate> _RandomAccessIter rfind_if (_RandomAccessIter __first, _RandomAccessIter __last, _Predicate __pred, std::random_access_iterator_tag)

Variables

- _Visitor _Walker
- _Visitor _Visitor
- _Visitor __w
- _Visitor __f
- _Visitor __it
- _Visitor __e

6.4.1 Detailed Description

The generic functions in this section are used by other generic algorithms and are not intended for external use.

6.4.2 Function Documentation

6.4.2.1 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_cached_walk (_Walker __w, _Visitor __f, _Predicate __p)

perform a recursive pre+post order walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited. If then predicate** `p` **returns** `true`**, the children are visited. If it returns** `false`**, the children are ignored**
- `collect` **is called everytime a child has finished**
- `postorder` **is called after the children have been visited.**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual ground node.**

Definition at line 1341 of file vgtl_algo.h.

6.4.2.2 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_cached_walk (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited. If it returns** `true`**, the children are visited. If it returns** `false`**, the children are ignored**
- `collect` **is called everytime a child has finished**
- `postorder` **is called after the children have been visited.**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual ground node.**

Definition at line 1090 of file vgtl_algo.h.

6.4.2.3 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_cached_walk_up (_Walker __w, _Visitor __f, _Predicate __p)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited. If then predicate** `__p` **returns** `true`**, the children are visited. If it returns** `false`**, the children are ignored**
- `collect` **is called everytime a child has finished**
- `postorder` **is called after the children have been visited.**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual sky node.**

Definition at line 2269 of file vgtl_algo.h.

6.4.2.4 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_cached_walk_up (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited. If it returns `true`, the children are visited. If it returns `false`, the children are ignored**
- `collect` **is called everytime a child has finished**
- `postorder` **is called after the children have been visited.**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual sky node.**

Definition at line 2110 of file vgtl_algo.h.

6.4.2.5 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_multi_walk (_Walker __w, _Visitor __f, _Predicate __p)

perform a recursive pre+post order walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node.**
- `vcollect` **is called after a child of a virtual node has finished.**
- `vvalue` **is called to compute the return value of a virtual node.**
- `preorder` **is called before the children are visited.**
- `collect` **is called everytime a child has finished.**
- `postorder` **is called after the children have been visited. If the predicate `__p` returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.**
- `value` **is called to compute the return value for this node. this function does not check for hitting the virtual ground node.**

Definition at line 1423 of file vgtl_algo.h.

6.4.2.6 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_multi_walk (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node.**
- `vcollect` **is called after a child of a virtual node has finished.**
- `vvalue` **is called to compute the return value of a virtual node.**
- `preorder` **is called before the children are visited.**
- `collect` **is called everytime a child has finished.**
- `postorder` **is called after the children have been visited. If it returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.**
- `value` **is called to compute the return value for this node. this function does not check for hitting the virtual ground node.**

Definition at line 1169 of file vgtl_algo.h.

6.4.2.7 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_multi_walk_up (_Walker __w, _Visitor __f, _Predicate __p)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node.
- `vcollect` is called after a child of a virtual node has finished.
- `vvalue` is called to compute the return value of a virtual node.
- `preorder` is called before the children are visited.
- `collect` is called everytime a child has finished.
- `postorder` is called after the children have been visited. If the predicate `__p` returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.
- `value` is called to compute the return value for this node. this function does not check for hitting the virtual sky node.

Definition at line 2352 of file vgtl_algo.h.

6.4.2.8 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_multi_walk_up (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node.
- `vcollect` is called after a child of a virtual node has finished.
- `vvalue` is called to compute the return value of a virtual node.
- `preorder` is called before the children are visited.
- `collect` is called everytime a child has finished.
- `postorder` is called after the children have been visited. If it returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.
- `value` is called to compute the return value for this node. this function does not check for hitting the virtual sky node.

Definition at line 2190 of file vgtl_algo.h.

6.4.2.9 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_postorder_walk (_Walker __w, _Visitor __f)

perform a recursive postorder walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `init` is called before the children are visited
- `collect` is called everytime a child has finished
- `postorder` is called after all children have finished
- `value` is called to compute the return value for this node this function does not check for hitting the virtual ground node.

Definition at line 635 of file vgtl_algo.h.

6.4.2.10 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_postorder_walk_if (_Walker __w, _Visitor __f, _Predicate __p)

perform a recursive postorder walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `init` **is called before the children are visited. Then the predicate is called. If this predicate returns** `true`**, the children are visited. Otherwise, the node is treated as if it was a terminal node.**
- `postorder` **is called after all children have been visited.**
- `collect` **is called everytime a child has finished.**
- `value` **is called to compute the return value for this node. this function does not check for hitting the virtual ground node.**

Definition at line 926 of file vgtl_algo.h.

6.4.2.11 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_postorder_walk_up (_Walker __w, _Visitor __f)

perform a recursive postorder walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `init` **is called before the children are visited**
- `collect` **is called everytime a child has finished**
- `postorder` **is called after all children have finished**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual sky node.**

Definition at line 1709 of file vgtl_algo.h.

6.4.2.12 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_postorder_walk_up_if (_Walker __w, _Visitor __f, _Predicate __p)

perform a recursive postorder walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `init` **is called before the children are visited. Then the predicate is called. If this predicate returns** `true`**, the children are visited. Otherwise, the node is treated as if it was a terminal node.**
- `postorder` **is called after all children have been visited.**
- `collect` **is called everytime a child has finished.**
- `value` **is called to compute the return value for this node.**

Definition at line 1785 of file vgtl_algo.h.

6.4.2.13 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_preorder_walk (_Walker __w, _Visitor __f)

perform a recursive preorder walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited**
- `collect` **is called everytime a child has finished**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual ground node.**

Definition at line 568 of file vgtl_algo.h.

6.4.2.14 template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_preorder_walk_if (_Walker __w, _Visitor __f, _Predicate __p)

perform a recursive preorder walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited. Then the predicate is called. If this predicate returns `true`, the children are visited. Otherwise, the node is treated as if it was a terminal node.**
- `collect` **is called everytime a child has finished**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual ground node.**

Definition at line 847 of file vgtl_algo.h.

6.4.2.15 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_preorder_walk_if (_Walker __w, _Visitor __f)

perform a recursive preorder walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited. If this function returns `true`, the children are visited. Otherwise, the node is treated as if it was a terminal node.**
- `collect` **is called everytime a child has finished**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual ground node.**

Definition at line 772 of file vgtl_algo.h.

6.4.2.16 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_preorder_walk_up (_Walker __w, _Visitor __f)

perform a recursive preorder walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited**
- `collect` **is called everytime a child has finished**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual sky node.**

Definition at line 1493 of file vgtl_algo.h.

6.4.2.17 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_preorder_walk_up_if (_Walker __w, _Visitor __f)

perform a recursive preorder walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited. If this function returns `true`, the children are visited. Otherwise, the node is treated as if it was a terminal node.**
- `collect` **is called everytime a child has finished**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual sky node.**

Definition at line 1563 of file vgtl_algo.h.

6.4.2.18 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_walk (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited**
- `collect` **is called everytime a child has finished**
- `postorder` **is called after all children have been visited**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual ground node.**

Definition at line 702 of file vgtl_algo.h.

6.4.2.19 template<class _Walker, class _Visitor, class _Predicate1, class _Predicate2> _Visitor::return_value _recursive_walk_if (_Walker __w, _Visitor __f, _Predicate1 __p1, _Predicate2 __p2)

perform a recursive pre+post order walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited. If then predicate `p1` returns `true`, the children are visited. If it returns `false`, the children are ignored**
- `collect` **is called everytime a child has finished**
- `postorder` **is called after the children have been visited. If then predicate `p2` returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual ground node.**

Definition at line 1258 of file vgtl_algo.h.

6.4.2.20 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_walk_if (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited. If it returns `true`, the children are visited. If it returns `false`, the children are ignored**
- `collect` **is called everytime a child has finished**
- `postorder` **is called after the children have been visited. If it returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual ground node.**

Definition at line 1012 of file vgtl_algo.h.

6.4.2.21 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_walk_up (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited**
- `collect` **is called everytime a child has finished**

- `postorder` is called after all children have been visited
- `value` is called to compute the return value for this node this function does not check for hitting the virtual sky node.

Definition at line 1855 of file vgtl_algo.h.

6.4.2.22 template<class _Walker, class _Visitor, class _Predicate1, class _Predicate2> _Visitor::return_value _recursive_walk_up_if (_Walker __w, _Visitor __f, _Predicate1 __p1, _Predicate2 __p2)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. If then predicate `p1` returns `true`, the children are visited. If it returns `false`, the children are ignored
- `collect` is called everytime a child has finished
- `postorder` is called after the children have been visited. If then predicate `p2` returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.
- `value` is called to compute the return value for this node this function does not check for hitting the virtual sky node.

Definition at line 2028 of file vgtl_algo.h.

6.4.2.23 template<class _Walker, class _Visitor> _Visitor::return_value _recursive_walk_up_if (_Walker __w, _Visitor __f)

perform a recursive pre+post order walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node.
- `vcollect` is called after a child of a virtual node has finished.
- `vvalue` is called to compute the return value of a virtual node.
- `preorder` is called before the children are visited. If it returns `true`, the children are visited. If it returns `false`, the children are ignored.
- `collect` is called everytime a child has finished.
- `postorder` is called after the children have been visited. If it returns `true`, the walk is continued by switching back to preorder mode for this node. If it returns `false`, the walk is over for this node.
- `value` is called to compute the return value for this node. this function does not check for hitting the virtual sky node.

Definition at line 1937 of file vgtl_algo.h.

6.4.2.24 template<class _RandomAccessIter, class _Tp> _RandomAccessIter rfind (_RandomAccessIter __first, _RandomAccessIter __last, const _Tp & __val, std::random_access_iterator_tag)

This is an overload used by `rfind()` (reverse find) for the Random Access Iterator case. `rfind()` works like the STL `find()` algorithm, just backwards.

Definition at line 86 of file vgtl_helpers.h.

6.4.2.25 template<class _BidirIter, class _Tp> _BidirIter rfind (_BidirIter __*first*, _BidirIter __*last*, const _Tp & __*val*, std::bidirectional_iterator_tag) [inline]

This is an overload used by `rfind()` (reverse find) for the Bidirectional Iterator case. `rfind()` works like the STL `find()` algorithm, just backwards.

Definition at line 44 of file vgtl_helpers.h.

6.4.2.26 template<class _RandomAccessIter, class _Predicate> _RandomAccessIter rfind_if (_RandomAccessIter __*first*, _RandomAccessIter __*last*, _Predicate __*pred*, std::random_access_iterator_tag)

This is an overload used by `rfind_if()` (reverse find if) for the Random Access Iterator case. `rfind_if()` works like the STL `find_if()` algorithm, just backwards.

Definition at line 136 of file vgtl_helpers.h.

6.4.2.27 template<class _BidirIter, class _Predicate> _BidirIter rfind_if (_BidirIter __*first*, _BidirIter __*last*, _Predicate __*pred*, std::bidirectional_iterator_tag) [inline]

This is an overload used by `rfind_if()` (reverse find if) for the Bidirectional Iterator case. `rfind_if()` works like the STL `find_if()` algorithm, just backwards.

Definition at line 64 of file vgtl_helpers.h.

6.4.3 Variable Documentation

6.4.3.1 _Visitor __e

perform a recursive preorder walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. Then the predicate is called. If this predicate returns `true`, the children are visited. Otherwise, the node is treated as if it was a terminal node.
- `collect` is called everytime a child has finished
- `value` is called to compute the return value for this node this function does not check for hitting the virtual sky node.

6.4.3.2 _Visitor __f

perform a recursive preorder walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. Then the predicate is called. If this predicate returns `true`, the children are visited. Otherwise, the node is treated as if it was a terminal node.
- `collect` is called everytime a child has finished

- `value` is called to compute the return value for this node this function does not check for hitting the virtual sky node.

6.4.3.3 _Visitor __it

perform a recursive preorder walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. Then the predicate is called. If this predicate returns `true`, the children are visited. Otherwise, the node is treated as if it was a terminal node.
- `collect` is called everytime a child has finished
- `value` is called to compute the return value for this node this function does not check for hitting the virtual sky node.

6.4.3.4 __f preorder * __w

perform a recursive preorder walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. Then the predicate is called. If this predicate returns `true`, the children are visited. Otherwise, the node is treated as if it was a terminal node.
- `collect` is called everytime a child has finished
- `value` is called to compute the return value for this node this function does not check for hitting the virtual sky node.

Definition at line 1646 of file vgtl_algo.h.

6.4.3.5 _Visitor _Visitor

perform a recursive preorder walk towards the root starting at `__w`. At every node various methods of the visitor `__f` are called:

- `vinit` is called before walking for every virtual node
- `vcollect` is called after a child of a virtual node has finished
- `vvalue` is called to compute the return value of a virtual node
- `preorder` is called before the children are visited. Then the predicate is called. If this predicate returns `true`, the children are visited. Otherwise, the node is treated as if it was a terminal node.
- `collect` is called everytime a child has finished
- `value` is called to compute the return value for this node this function does not check for hitting the virtual sky node.

6.4.3.6 _Visitor _Walker

perform a recursive preorder walk towards the root starting at __w. At every node various methods of the visitor __f are called:

- `vinit` **is called before walking for every virtual node**
- `vcollect` **is called after a child of a virtual node has finished**
- `vvalue` **is called to compute the return value of a virtual node**
- `preorder` **is called before the children are visited. Then the predicate is called. If this predicate returns** `true`**, the children are visited. Otherwise, the node is treated as if it was a terminal node.**
- `collect` **is called everytime a child has finished**
- `value` **is called to compute the return value for this node this function does not check for hitting the virtual sky node.**

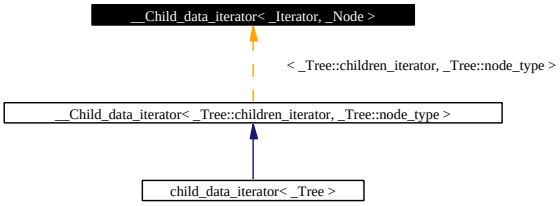
7 Vienna Graph Template Library Class Documentation

7.1 __Child_data_iterator Class Template Reference

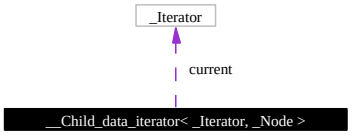
iterator adapter for iterating through children data hooks.

`#include <vgtl_algo.h>`

Inheritance diagram for __Child_data_iterator:



Collaboration diagram for __Child_data_iterator:



Public Types

- **typedef ctree_data_hook value_type**
- **typedef value_type * pointer**
- **typedef value_type & reference**

Public Methods

- **__Child_data_iterator (const _Self &__x)**
  standard destructor.
- **iterator_type base () const**
  return the 'unwrapped' iterator.
- **reference operator * () const**
  dereference to the `data_hook`.
- **_Self & operator= (const iterator_type &it)**
  assignment operator.

- **__Child_data_iterator ()**
  standard constructors.
- **__Child_data_iterator (iterator_type __x)**
  standard constructors.

- **bool operator== (const _Self &__x) const**
  standard comparison operator.
- **bool operator!= (const _Self &__x) const**
  standard comparison operator.

- **_Self & operator++ ()**
  standard in(de)crement operator.
- **_Self & operator++ (int)**
  standard in(de)crement operator.
- **_Self & operator– ()**
  standard in(de)crement operator.
- **_Self & operator– (int)**
  standard in(de)crement operator.

- **_Self operator+ (difference_type __n) const**
  additional operator for random access iterators.
- **_Self & operator+= (difference_type __n)**
  additional operator for random access iterators.
- **_Self operator- (difference_type __n) const**
  additional operator for random access iterators.

- _Self & operator-= (difference_type __n)

  additional operator for random access iterators.

- reference operator[ ] (difference_type __n) const

  additional operator for random access iterators.

Protected Attributes

- _Iterator current

  that's where we are.

7.1.1 Detailed Description

template<class _Iterator, class _Node> class __Child_data_iterator< _Iterator, _Node >

@addgroup internal This class is an iterator adapter for iterating through the data hooks of all children of a given node

Definition at line 50 of file vgtl_algo.h.

7.1.2 Member Typedef Documentation

7.1.2.1 template<class _Iterator, class _Node> typedef value_type* __Child_data_iterator::pointer

standard iterator definitions

Definition at line 63 of file vgtl_algo.h.

7.1.2.2 template<class _Iterator, class _Node> typedef value_type& __Child_data_iterator::reference

standard iterator definitions

Definition at line 64 of file vgtl_algo.h.

7.1.2.3 template<class _Iterator, class _Node> typedef ctree_data_hook __Child_data_iterator::value_type

standard iterator definitions

Definition at line 62 of file vgtl_algo.h.

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

- vgtl_algo.h

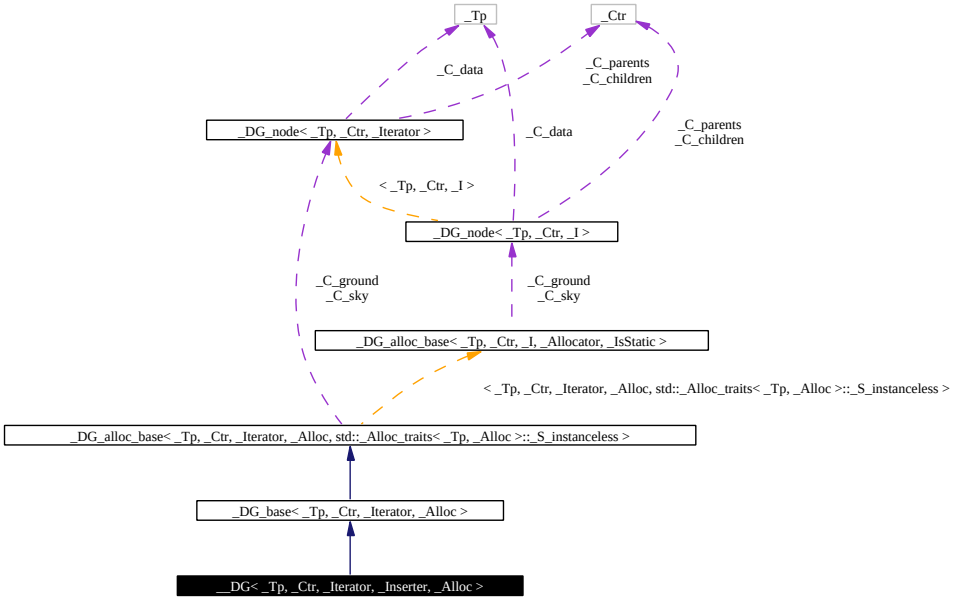
7.2 __DG Class Template Reference

Directed graph base class.

`#include <vgtl_dag.h>`

Inheritance diagram for __DG:

Collaboration diagram for __DG:



Public Types

- **typedef _Ctr container_type**
- **typedef _Iterator children_iterator**
- **typedef _Iterator parents_iterator**
- **typedef _Base::allocator_type allocator_type**
- **typedef _DG_iterator< _Tp, _Tp &, _Tp *, container_type, children_iterator > iterator**
- **typedef _DG_iterator< _Tp, const _Tp &, const _Tp *, container_type, children_iterator > const_iterator**
- **typedef std::reverse_iterator< const_iterator > const_reverse_iterator**
- **typedef std::reverse_iterator< iterator > reverse_iterator**
- **typedef _DG_walker< _Tp, _Tp &, _Tp *, container_type, children_iterator > walker**
- **typedef _DG_walker< _Tp, const _Tp &, const _Tp *, container_type, children_iterator > const_walker**
- **typedef std::pair< walker, walker > edge**
- **typedef std::pair< edge, bool > enhanced_edge**

- **typedef _Tp value_type**

- typedef _Node node_type
- typedef value_type * pointer
- typedef const value_type * const_pointer
- typedef value_type & reference
- typedef const value_type & const_reference
- typedef size_t size_type
- typedef ptrdiff_t difference_type

Public Methods

- allocator_type get_allocator () const
- __DG (const allocator_type &__a=allocator_type())
- walker ground ()
- walker sky ()
- const_walker ground () const
- const_walker sky () const
- children_iterator root_begin ()
- children_iterator root_end ()
- parents_iterator leaf_begin ()
- parents_iterator leaf_end ()
- bool empty () const
- size_type size () const
- size_type max_size () const
- void swap (_Self &__x)
- walker insert_node_in_graph (_Node *__n, const walker &__parent, const walker &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- walker insert_in_graph (const _Tp &__x, const walker &__parent, const walker &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- walker insert_in_graph (const walker &__parent, const walker &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- void insert_subgraph (_Self &__subgraph, const walker &__parent, const walker &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> walker insert_node_in_graph (_Node *__node, const __SequenceCtr1< walker, _Allocator1 > &__parents, const __SequenceCtr2< walker, _Allocator2 > &__children)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> walker insert_in_graph (const _Tp &__x, const __SequenceCtr1< walker, _Allocator1 > &__parents, const __SequenceCtr2< walker, _Allocator2 > &__children)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> walker insert_in_graph (const __SequenceCtr1< walker, _Allocator1 > &__parents, const __SequenceCtr2< walker, _Allocator2 > &__children)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker insert_node_in_graph (_Node *__node, const walker &__parent, const container_insert_arg &__pref, const __SequenceCtr< walker, _Allocator > &__children)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker insert_in_graph (const _Tp &__x, const walker &__parent, const container_insert_arg &__pref, const __SequenceCtr< walker, _Allocator > &__children)

- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker insert_in_graph (const walker &__parent, const container_insert_arg &__pref, const __SequenceCtr< walker, _Allocator > &__children)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker insert_node_in_graph (_Node *__node, const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const container_insert_arg &__cref)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker insert_in_graph (const _Tp &__x, const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const container_insert_arg &__cref)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker insert_in_graph (const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const container_insert_arg &__cref)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> void insert_subgraph (_Self &__subgraph, const __SequenceCtr1< walker, _Allocator1 > &__parents, const __SequenceCtr2< walker, _Allocator2 > &__children)
- void add_edge (const edge &__edge, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- void add_edge (const walker &__parent, const walker &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- void replace_edge_to_child (const walker &__parent, const walker &__child_old, const walker &__child_new)
- void replace_edge_to_parent (const walker &__parent_old, const walker &__parent_new, const walker &__child)
- void remove_edge (const edge &__edge)
- void remove_edge_and_deattach (const walker &__parent, const walker &__child)
- void remove_edge (const walker &__parent, const walker &__child)
- template<class Compare> void sort_child_edges (walker __position, children_iterator first, children_iterator last, Compare comp)
- template<class Compare> void sort_parent_edges (walker __position, parents_iterator first, parents_iterator last, Compare comp)
- template<class Compare> void sort_child_edges (walker __position, Compare comp)
- template<class Compare> void sort_parent_edges (walker __position, Compare comp)
- walker insert_node (_Node *__node, const walker &__position, const container_insert_arg &__It)
- walker insert_node (const _Tp &__x, const walker &__position, const container_insert_arg &__It)
- walker insert_node (const walker &__position, const container_insert_arg &__It)
- walker insert_node_before (_Node *__node, const walker &__position, const container_insert_arg &__It)
- void insert_node_before (const _Tp &__x, const walker &__position, const container_insert_arg &__It)
- void insert_node_before (const walker &__position, const container_insert_arg &__It)
- void merge (const walker &__position, const walker &__second, bool merge_parent_edges=true, bool merge_child_edges=true)
- void erase (const walker &__position)
- void clear_erased_part (erased_part &_ep)
- erased_part erase_maximal_subgraph (const walker &__position)
- erased_part erase_minimal_subgraph (const walker &__position)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> erased_part erase_maximal_subgraph (const __SequenceCtr< walker, _Allocator > &__positions)

- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> erased_part erase_minimal_subgraph (const __SequenceCtr< walker, _Allocator > &__positions)
- erased_part erase_maximal_pregraph (const walker &__position)
- erased_part erase_minimal_pregraph (const walker &__position)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> erased_part erase_maximal_pregraph (const __SequenceCtr< walker, _Allocator > &__positions)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> erased_part erase_minimal_pregraph (const __SequenceCtr< walker, _Allocator > &__positions)
- bool erase_child (const walker &__position, const children_iterator &__It)
- bool erase_parent (const walker &__position, const parents_iterator &__It)
- void clear ()
- __DG (const _Self &__x)
- ~__DG ()
- _Self & operator= (const _Self &__x)
- _Self & operator= (const _RV_DG &__rl)
- _Self & operator= (const erased_part &__ep)
- void clear_children ()
- void clear_parents ()
- template<class _Output_Iterator> void add_all_children (_Output_Iterator fi, _DG_node< _Tp, _Ctr, _Iterator > *_parent)
- template<class _Output_Iterator> void add_all_parents (_Output_Iterator fi, _DG_node< _Tp, _Ctr, _Iterator > *_child)

Protected Types

- typedef std::pair< _RV_DG, std::vector< enhanced_edge > > erased_part

Protected Methods

- _Node * _C_create_node (const _Tp &__x)
- _Node * _C_create_node ()
- void clear_graph (_DG_node< _Tp, _Ctr, _Iterator > *_node)

Friends

- bool operator==__VGTL_NULL_TMPL_ARGS (const __DG &__x, const __DG &__y)

7.2.1 Detailed Description

template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> class __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >

This is the toplevel base class for all directed graphs independent of allocators

Definition at line 506 of file vgtl_dag.h.

7.2.2 Member Typedef Documentation

7.2.2.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _Base::allocator_type __DG::allocator_type

allocator type

Reimplemented from _DG_base.

Definition at line 535 of file vgtl_dag.h.

7.2.2.2 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _Iterator __DG::children_iterator

iterator for accessing the children

Reimplemented from _DG_base.

Reimplemented in dgraph.

Definition at line 510 of file vgtl_dag.h.

7.2.2.3 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _DG_iterator<_Tp,const _Tp&,const _Tp*,container_type,children_iterator> __DG::const_iterator

the const iterator

Definition at line 543 of file vgtl_dag.h.

7.2.2.4 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef const value_type* __DG::const_pointer

standard typedef

Definition at line 528 of file vgtl_dag.h.

7.2.2.5 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef const value_type& __DG::const_reference

standard typedef

Definition at line 530 of file vgtl_dag.h.

7.2.2.6 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef std::reverse_iterator<const_iterator> __DG::const_reverse_iterator

the const reverse iterator

Definition at line 547 of file vgtl_dag.h.

7.2.2.7 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _DG_walker<_Tp,const _Tp&,const _Tp*,container_type,children_iterator> __DG::const_walker

the (recursive) const walker

Reimplemented in dgraph.

Definition at line 564 of file vgtl_dag.h.

7.2.2.8 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _Ctr __DG::container_type

internal container used to store the children and parents

Reimplemented from _DG_base.

Definition at line 509 of file vgtl_dag.h.

7.2.2.9 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef ptrdiff_t __DG::difference_type

standard typedef

Definition at line 532 of file vgtl_dag.h.

7.2.2.10 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef std::pair<walker,walker> __DG::edge

an edge of the graph (parent, child)

Definition at line 567 of file vgtl_dag.h.

7.2.2.11 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef std::pair<edge,bool> __DG::enhanced_edge

an edge with additiona information about erased ground/sky edges

Definition at line 569 of file vgtl_dag.h.

7.2.2.12 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef std::pair<_RV_DG, std::vector<enhanced_edge> > __DG::erased_part [protected]

an erased subgraph which is not yet a new directed graph

Definition at line 573 of file vgtl_dag.h.

7.2.2.13 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _DG_iterator<_Tp,_Tp&,_Tp*,container_type,children_iterator> __DG::iterator

the iterator

Definition at line 541 of file vgtl_dag.h.

7.2.2.14 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _Node __DG::node_type

standard typedef

Definition at line 526 of file vgtl_dag.h.

7.2.2.15 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _Iterator __DG::parents_iterator

iterator for accessing the parents

Reimplemented from _DG_base.

Reimplemented in dgraph.

Definition at line 511 of file vgtl_dag.h.

7.2.2.16 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef value_type* __DG::pointer

standard typedef

Definition at line 527 of file vgtl_dag.h.

7.2.2.17 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef value_type& __DG::reference

standard typedef

Definition at line 529 of file vgtl_dag.h.

7.2.2.18 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef std::reverse_iterator<iterator> __DG::reverse_iterator

the reverse iterator

Definition at line 549 of file vgtl_dag.h.

7.2.2.19 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef size_t __DG::size_type

standard typedef

Definition at line 531 of file vgtl_dag.h.

7.2.2.20 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _Tp __DG::value_type

standard typedef

Definition at line 525 of file vgtl_dag.h.

7.2.2.21 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _DG_walker<_Tp,_Tp&,_Tp*,container_type,children_iterator> __DG::walker

the (recursive) walker

Reimplemented in dgraph.

Definition at line 562 of file vgtl_dag.h.

7.2.3 Constructor & Destructor Documentation

7.2.3.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::__DG (const allocator_type & __*a* = allocator_type()) `[inline, explicit]`

standard constructor

Definition at line 615 of file vgtl_dag.h.

7.2.3.2 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::__DG (const _Self & __x) [inline]

copy constructor

Definition at line 1773 of file vgtl_dag.h.

7.2.3.3 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::~__DG () [inline]

standard destructor

Definition at line 1790 of file vgtl_dag.h.

7.2.4 Member Function Documentation

7.2.4.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> _Node* __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::_C_create_node () [inline, protected]

construct a new tree node containing default data

Definition at line 600 of file vgtl_dag.h.

7.2.4.2 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> _Node* __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::_C_create_node (const _Tp & __x) [inline, protected]

construct a new tree node containing data __x

Definition at line 586 of file vgtl_dag.h.

7.2.4.3 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> template<class _Output_Iterator> void _DG_base< _Tp, _Ctr, _Iterator, _Alloc >::add_all_children (_Output_Iterator *fi*, _DG_node< _Tp, _Ctr, _Iterator > * *_parent*) [inline, inherited]

add all children to the parent _parent. fi is a iterator to the children container of the parent

Definition at line 459 of file vgtl_dagbase.h.

7.2.4.4 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> template<class _Output_Iterator> void _DG_base< _Tp, _Ctr, _Iterator, _Alloc >::add_all_parents (_Output_Iterator *fi*, _DG_node< _Tp, _Ctr, _Iterator > * *_child*) [inline, inherited]

add all parents to the child _child. fi is a iterator to the container of the child

Definition at line 466 of file vgtl_dagbase.h.

7.2.4.5 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::add_edge (const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inline]

add an edge between __parent and __child at positions __Itc and __Itp, respectively

Definition at line 980 of file vgtl_dag.h.

7.2.4.6 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::add_edge (const edge & __*edge*, const container_insert_arg & __*Itc*, const container_insert_arg & __*Itp*) `[inline]`

add one edge between two nodes at the positions described by `__Itc` **and** `__Itp`**.**

Definition at line 971 of file vgtl_dag.h.

7.2.4.7 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::clear () `[inline]`

erase all the nodes except sky and ground

Reimplemented from _DG_base.

Reimplemented in dgraph.

Definition at line 1733 of file vgtl_dag.h.

7.2.4.8 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> void _DG_base< _Tp, _Ctr, _Iterator, _Alloc >::clear_children () `[inline, inherited]`

clear all children of the ground node

Definition at line 314 of file vgtl_dagbase.h.

7.2.4.9 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::clear_erased_part (erased_part & _*ep*) `[inline]`

clear all nodes in an erased part

Definition at line 1533 of file vgtl_dag.h.

7.2.4.10 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> void _DG_base< _Tp, _Ctr, _Iterator, _Alloc >::clear_graph (_DG_node< _Tp, _Ctr, _Iterator > * _*node*) `[protected, inherited]`

removes all the nodes of the graph except the sky and ground nodes

Definition at line 430 of file vgtl_dagbase.h.

7.2.4.11 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> void _DG_base< _Tp, _Ctr, _Iterator, _Alloc >::clear_parents () `[inline, inherited]`

clear all parents of the sky node

Definition at line 317 of file vgtl_dagbase.h.

7.2.4.12 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> bool __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::empty () const `[inline]`

returns `true` **if the DG is empty**

Definition at line 668 of file vgtl_dag.h.

7.2.4.13 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::erase (const walker & __*position*) `[inline]`

erase a node from the DG except the sky and ground

Definition at line 1297 of file vgtl_dag.h.

7.2.4.14 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> bool __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::erase_child (const walker & __*position*, const children_iterator & __*It*) [inline]

Erase a child of `__position`. This works if and only if the child has only one child and no other parents.

Definition at line 1685 of file vgtl_dag.h.

7.2.4.15 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> erased_part __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::erase_maximal_pregraph (const __SequenceCtr< walker, _Allocator > & __*positions*) [inline]

here every child is removed till the sky included all nodes from `__positions`. The removed subgraph is returned. The subgraph is maximal, i.e. all nodes are removed, which are reachable from any node from `__positions` by walking up.

Definition at line 1649 of file vgtl_dag.h.

7.2.4.16 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> erased_part __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::erase_maximal_pregraph (const walker & __*position*) [inline]

here every child is removed till the sky node. included the node at `__position`. The removed subgraph is returned. The subgraph is maximal, i.e. all nodes are removed, which are reachable from `__position` by walking upwards.

Definition at line 1615 of file vgtl_dag.h.

7.2.4.17 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> erased_part __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::erase_maximal_subgraph (const __SequenceCtr< walker, _Allocator > & __*positions*) [inline]

here every child is removed till the last base node, included all nodes from `__positions`. The removed subgraph is returned. The subgraph is maximal, i.e. all nodes are removed, which are reachable from any node from `__positions` by walking down.

Definition at line 1578 of file vgtl_dag.h.

7.2.4.18 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> erased_part __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::erase_maximal_subgraph (const walker & __*position*) [inline]

here every child is removed till the last base node, included the node at `__position`. The removed subgraph is returned. The subgraph is maximal, i.e. all nodes are removed, which are reachable from `__position` by walking down.

Definition at line 1544 of file vgtl_dag.h.

7.2.4.19 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> erased_part __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::erase_minimal_pregraph (const __SequenceCtr< walker, _Allocator > & __*positions*) `[inline]`

here every child is removed till the sky. included all nodes from `__positions`. The removed subgraph is returned. The subgraph is minimal, i.e. only nodes are removed, which have no other ancestor than any node in `__positions`. I.e., when walking towards the ground, there is no way which bypasses all nodes in `__positions`.

Definition at line 1669 of file vgtl_dag.h.

7.2.4.20 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> erased_part __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::erase_minimal_pregraph (const walker & __*position*) `[inline]`

here every child is removed till the sky. included the node at `__position`. The removed subgraph is returned. The subgraph is minimal, i.e. only nodes are removed, which have no other descendant than `__position`. I.e., when walking towards the sky, there is no way which bypasses `__position`.

Definition at line 1631 of file vgtl_dag.h.

7.2.4.21 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> erased_part __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::erase_minimal_subgraph (const __SequenceCtr< walker, _Allocator > & __*positions*) `[inline]`

here every child is removed till the last base node, included all nodes from `__positions`. The removed subgraph is returned. The subgraph is minimal, i.e. only nodes are removed, which have no other ancestor than any node in `__positions`. I.e., when walking towards the ground, there is no way which bypasses all nodes in `__positions`.

Definition at line 1598 of file vgtl_dag.h.

7.2.4.22 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> erased_part __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::erase_minimal_subgraph (const walker & __*position*) `[inline]`

here every child is removed till the last base node, included the node at `__position`. The removed subgraph is returned. The subgraph is minimal, i.e. only nodes are removed, which have no other ancestor than `__position`. I.e., when walking towards the ground, there is no way which bypasses `__position`.

Definition at line 1560 of file vgtl_dag.h.

7.2.4.23 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> bool __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::erase_parent (const walker & __*position*, const parents_iterator & __*It*) `[inline]`

Erase a parent of `__position`. This works if and only if the parent has only one parent and no other children.

Definition at line 1711 of file vgtl_dag.h.

7.2.4.24 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> allocator_type __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::get_allocator () const `[inline]`

construct an allocator object

Reimplemented from _DG_alloc_base< _Tp, _Ctr, _Iterator, _Alloc, std::_Alloc_traits< _Tp, _Alloc >::_S_instanceless >.

Definition at line 537 of file vgtl_dag.h.

7.2.4.25 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::ground () const [inline]

return a const walker to the virtual ground node.

Definition at line 628 of file vgtl_dag.h.

7.2.4.26 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::ground () [inline]

return a walker to the virtual ground node.

Definition at line 618 of file vgtl_dag.h.

7.2.4.27 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_in_graph (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const container_insert_arg & *__cref*) [inline]

insert a node with default data into the graph between all parents from `__parents` and the child `__child`.

Definition at line 907 of file vgtl_dag.h.

7.2.4.28 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_in_graph (const _Tp & *__x*, const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const container_insert_arg & *__cref*) [inline]

insert a node with data `__x` into the graph between all parents from `__parents` and the child `__child`.

Definition at line 892 of file vgtl_dag.h.

7.2.4.29 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_in_graph (const walker & *__parent*, const container_insert_arg & *__pref*, const __SequenceCtr< walker, _Allocator > & *__children*) [inline]

insert a node with data `__x` into the graph between the parent `__parent` and all children from `__children`.

Definition at line 853 of file vgtl_dag.h.

7.2.4.30 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker

__DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_in_graph (const _Tp & __*x*, const walker & __*parent*, const container_insert_arg & __*pref*, const __SequenceCtr< walker, _Allocator > & __*children*) `[inline]`

insert a node with data `__x` **into the graph between the parent** `__parent` **and all children from** `__children`**.**

Definition at line 839 of file vgtl_dag.h.

7.2.4.31 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_in_graph (const __SequenceCtr1< walker, _Allocator1 > & __*parents*, const __SequenceCtr2< walker, _Allocator2 > & __*children*) `[inline]`

insert a node with default data into the graph between all parents from `__parents` **and all children from** `__children`**.**

Definition at line 801 of file vgtl_dag.h.

7.2.4.32 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_in_graph (const _Tp & __*x*, const __SequenceCtr1< walker, _Allocator1 > & __*parents*, const __SequenceCtr2< walker, _Allocator2 > & __*children*) `[inline]`

insert a node with data `__x` **into the graph between all parents from** `__parents` **and all children from** `__children`**.**

Definition at line 786 of file vgtl_dag.h.

7.2.4.33 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_in_graph (const walker & __*parent*, const walker & __*child*, const container_insert_arg & __*Itc*, const container_insert_arg & __*Itp*) `[inline]`

insert node with default data into the graph between `__parent` **and** `__child`**, the edge at the specific positions described by** `__Itc` **and** `__Itp`**.**

Definition at line 722 of file vgtl_dag.h.

7.2.4.34 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_in_graph (const _Tp & __*x*, const walker & __*parent*, const walker & __*child*, const container_insert_arg & __*Itc*, const container_insert_arg & __*Itp*) `[inline]`

insert node with data `__n` **into the graph between** `__parent` **and** `__child`**, the edge at the specific positions described by** `__Itc` **and** `__Itp`**.**

Definition at line 708 of file vgtl_dag.h.

7.2.4.35 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_node (const walker & __*position*, const container_insert_arg & __*It*) `[inline]`

insert a new node with default data as child of `__position`

Definition at line 1178 of file vgtl_dag.h.

7.2.4.36 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_node (const _Tp & __*x*, const walker & __*position*, const container_insert_arg & __*It*) `[inline]`

insert a new node with data `__x` **as child of** `__position`

Definition at line 1172 of file vgtl_dag.h.

7.2.4.37 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_node (_Node * _*node*, const walker & __*position*, const container_insert_arg & __*It*) `[inline]`

insert one node as child of `__position`

Definition at line 1158 of file vgtl_dag.h.

7.2.4.38 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_node_before (const walker & __*position*, const container_insert_arg & __*It*) `[inline]`

insert a new node with default data as parent of `__position`

Definition at line 1202 of file vgtl_dag.h.

7.2.4.39 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_node_before (const _Tp & __*x*, const walker & __*position*, const container_insert_arg & __*It*) `[inline]`

insert a new node with data `__x` **as parent of** `__position`

Definition at line 1197 of file vgtl_dag.h.

7.2.4.40 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_node_before (_Node * _*node*, const walker & __*position*, const container_insert_arg & __*It*) `[inline]`

insert a node as parent of `__position`

Definition at line 1183 of file vgtl_dag.h.

7.2.4.41 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_node_in_graph (_Node * __*node*, const __SequenceCtr< walker, _Allocator > & __*parents*, const walker & __*child*, const container_insert_arg & __*cref*) `[inline]`

insert node `__n` **into the graph between all parents from** `__parents` **and the child** `__child`**.**

Definition at line 867 of file vgtl_dag.h.

7.2.4.42 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_node_in_graph (_Node * __*node*, const walker

& __*parent*, const container_insert_arg & __*pref*, const __SequenceCtr< walker, _Allocator > & __*children*) `[inline]`

insert node __n into the graph between the parent __parent and all children from __children.

Definition at line 814 of file vgtl_dag.h.

7.2.4.43 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_node_in_graph (_Node * __*node*, const __SequenceCtr1< walker, _Allocator1 > & __*parents*, const __SequenceCtr2< walker, _Allocator2 > & __*children*) `[inline]`

insert node __n into the graph between all parents from __parents and all children from __children.

Definition at line 755 of file vgtl_dag.h.

7.2.4.44 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_node_in_graph (_Node * __*n*, const walker & __*parent*, const walker & __*child*, const container_insert_arg & __*Itc*, const container_insert_arg & __*Itp*) `[inline]`

insert node __n into the graph between __parent and __child, the edge at the specific positions described by __Itc and __Itp.

Definition at line 692 of file vgtl_dag.h.

7.2.4.45 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_subgraph (_Self & __*subgraph*, const __SequenceCtr1< walker, _Allocator1 > & __*parents*, const __SequenceCtr2< walker, _Allocator2 > & __*children*) `[inline]`

in this method one DG is inserted into another DG between the parents __parents and the children __children.

Definition at line 921 of file vgtl_dag.h.

7.2.4.46 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::insert_subgraph (_Self & __*subgraph*, const walker & __*parent*, const walker & __*child*, const container_insert_arg & __*Itc*, const container_insert_arg & __*Itp*) `[inline]`

insert a subgraph into the graph between __parent and __child, the edge at the specific positions described by __Itc and __Itp.

Definition at line 733 of file vgtl_dag.h.

7.2.4.47 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> parents_iterator __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::leaf_begin () `[inline]`

return the first leaf of the directed graph

Definition at line 645 of file vgtl_dag.h.

7.2.4.48 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> parents_iterator __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::leaf_end () [inline]

return beyond the last leaf of the directed graph

Definition at line 648 of file vgtl_dag.h.

7.2.4.49 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> size_type __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::max_size () const [inline]

the maximum size of a DG is virtually unlimited

Definition at line 679 of file vgtl_dag.h.

7.2.4.50 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::merge (const walker & *__position*, const walker & *__second*, bool *merge_parent_edges* = true, bool *merge_child_edges* = true) [inline]

merge two nodes, call also the merge method for the node data

Definition at line 1208 of file vgtl_dag.h.

7.2.4.51 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> _Self& __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::operator= (const erased_part & *__ep*) [inline]

assignment operator from an erased part

Reimplemented in dgraph.

Definition at line 1804 of file vgtl_dag.h.

7.2.4.52 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> _Self& __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::operator= (const _RV_DG & *__rl*) [inline]

assignment operator from a part of an erased part

Reimplemented in dgraph.

Definition at line 1796 of file vgtl_dag.h.

7.2.4.53 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> _Self& __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::operator= (const _Self & *__x*)

standard assignment operator

7.2.4.54 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::remove_edge (const walker & *__parent*, const walker & *__child*) [inline]

just remove one edge between `__parent` and `__child`

Definition at line 1111 of file vgtl_dag.h.

7.2.4.55 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::remove_edge (const edge & *__edge*) [inline]

remove an edge with a particular parent and child

Definition at line 1094 of file vgtl_dag.h.

7.2.4.56 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::remove_edge_and_deattach (const walker & __*parent*, const walker & __*child*) `[inline]`

remove one egde and don't reconnect the node to sky/ground

Definition at line 1098 of file vgtl_dag.h.

7.2.4.57 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::replace_edge_to_child (const walker & __*parent*, const walker & __*child_old*, const walker & __*child_new*) `[inline]`

change the edge from `__parent` **to** `__child_old` **to an edge from** `__parent` **to** `__child_new`**.**

Definition at line 1023 of file vgtl_dag.h.

7.2.4.58 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::replace_edge_to_parent (const walker & __*parent_old*, const walker & __*parent_new*, const walker & __*child*) `[inline]`

change the edge from `__parent_old` **to** `__child` **to an edge from** `__parent_new` **to** `__child`**.**

Definition at line 1060 of file vgtl_dag.h.

7.2.4.59 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> children_iterator __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::root_begin () `[inline]`

return the first root of the directed graph

Definition at line 638 of file vgtl_dag.h.

7.2.4.60 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> children_iterator __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::root_end () `[inline]`

return beyond the last root of the directed graph

Definition at line 641 of file vgtl_dag.h.

7.2.4.61 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> size_type __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::size () const `[inline]`

returns the size of the DG (number of nodes)

Definition at line 672 of file vgtl_dag.h.

7.2.4.62 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::sky () const `[inline]`

return a const walker to the virtual sky node.

Definition at line 633 of file vgtl_dag.h.

7.2.4.63 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> walker __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::sky () [inline]

return a walker to the virtual sky node.

Definition at line 623 of file vgtl_dag.h.

7.2.4.64 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<class Compare> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::sort_child_edges (walker __*position*, Compare *comp*) [inline]

sort all child edges according to `comp`

Definition at line 1147 of file vgtl_dag.h.

7.2.4.65 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<class Compare> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::sort_child_edges (walker __*position*, children_iterator *first*, children_iterator *last*, Compare *comp*) [inline]

sort the child edges in the range [first,last) according to `comp`

Definition at line 1135 of file vgtl_dag.h.

7.2.4.66 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<class Compare> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::sort_parent_edges (walker __*position*, Compare *comp*) [inline]

sort all parent edges according to `comp`

Definition at line 1153 of file vgtl_dag.h.

7.2.4.67 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> template<class Compare> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::sort_parent_edges (walker __*position*, parents_iterator *first*, parents_iterator *last*, Compare *comp*) [inline]

sort the parent edges in the range [first,last) according to `comp`

Definition at line 1141 of file vgtl_dag.h.

7.2.4.68 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> void __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::swap (_Self & __*x*) [inline]

swap two DGs

Definition at line 682 of file vgtl_dag.h.

7.2.5 Friends And Related Function Documentation

7.2.5.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> bool operator==__VGTL_NULL_TMPL_ARGS (const __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc > & __*x*, const __DG< _Tp, _Ctr, _Iterator, _Inserter, _Alloc > & __*y*) [friend]

standard comparison operator

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

- vgtl_dag.h

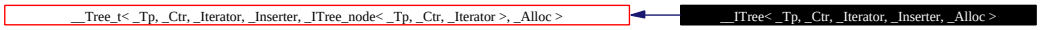
7.3 __ITree Class Template Reference

Tree base class with data hooks.

`#include <vgtl_tree.h>`

Inheritance diagram for __ITree:

Collaboration diagram for __ITree:



Public Types

- **typedef _Tree_iterator< _Tp, _Tp &, _Tp *, container_type, children_iterator, node_type > iterator**
- **typedef _Tree_iterator< _Tp, const _Tp &, const _Tp *, container_type, children_iterator, node_type > const_iterator**
- **typedef _Tree_walker< _Tp, _Tp &, _Tp *, container_type, children_iterator, _Node > iterative_walker**
- **typedef _Tree_walker< _Tp, const _Tp &, const _Tp *, container_type, children_iterator, _Node > const_iterative_walker**
- **typedef std::reverse_iterator< const_iterator > const_reverse_iterator**
- **typedef std::reverse_iterator< iterator > reverse_iterator**

Public Methods

- **__ITree (const allocator_type &__a=allocator_type())**
- **iterative_walker root (walker_type wt=cw_pre_post, bool front_to_back=true, bool depth_first=true)**
- **const_iterative_walker root (walker_type wt=cw_pre_post, bool front_to_back=true, bool depth_first=true) const**
- **iterative_walker through ()**
- **const_iterative_walker through () const**
- **iterative_walker begin (walker_type wt=cw_pre_post, bool front_to_back=true, bool depth_first=true)**
- **const_iterative_walker begin (walker_type wt=cw_pre_post, bool front_to_back=true, bool depth_first=true) const**
- **iterative_walker end (walker_type wt=cw_pre_post, bool front_to_back=true, bool depth_first=true)**
- **const_iterative_walker end (walker_type wt=cw_pre_post, bool front_to_back=true, bool depth_first=true) const**
- **reverse_iterator rbegin ()**
- **reverse_iterator rend ()**
- **const_reverse_iterator rbegin () const**
- **const_reverse_iterator rend () const**
- **size_type size () const**

- reference getroot ()
- const_reference getroot () const
- size_type depth (const iterative_walker &__position)
- __ITree (size_type __n, const _Tp &__value, const allocator_type &__a=allocator_type())
- __ITree (size_type __n)
- __ITree (const _Self &__x)
- virtual ~__ITree ()
- _Self & operator= (const _Self &__x)
- _Self & operator= (_Node *__x)

Friends

- bool operator==__VGTL_NULL_TMPL_ARGS (const __ITree &__x, const __ITree &__y)

7.3.1 Detailed Description

template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> class __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >

This is the base class for all trees with data hooks

Definition at line 2044 of file vgtl_tree.h.

7.3.2 Member Typedef Documentation

7.3.2.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _Tree_walker<_Tp,const _Tp&,const _Tp*,container_type,children_iterator,_Node> __ITree::const_iterative_walker

the const iterative walker

Definition at line 2064 of file vgtl_tree.h.

7.3.2.2 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _Tree_iterator<_Tp,const _Tp&,const _Tp*,container_type,children_iterator,node_type> __ITree::const_iterator

the const iterator

Reimplemented from __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _ITree_node< _Tp, _Ctr, _Iterator >, _Alloc >.

Definition at line 2059 of file vgtl_tree.h.

7.3.2.3 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef std::reverse_iterator<const_iterator> __ITree::const_reverse_iterator

the const reverse iterator

Reimplemented from __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _ITree_node< _Tp, _Ctr, _Iterator >, _Alloc >.

Definition at line 2068 of file vgtl_tree.h.

7.3.2.4 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _Tree_walker<_Tp,_Tp&,_Tp*,container_type,children_iterator,_Node> __ITree::iterative_walker

the iterative walker

Definition at line 2062 of file vgtl_tree.h.

7.3.2.5 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _Tree_iterator<_Tp,_Tp&,_Tp*,container_type,children_iterator,node_type> __ITree::iterator

the iterator

Reimplemented from __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _ITree_node< _Tp, _Ctr, _Iterator >, _Alloc >.

Definition at line 2057 of file vgtl_tree.h.

7.3.2.6 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef std::reverse_iterator<iterator> __ITree::reverse_iterator

the reverse iterator

Reimplemented from __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _ITree_node< _Tp, _Ctr, _Iterator >, _Alloc >.

Definition at line 2070 of file vgtl_tree.h.

7.3.3 Constructor & Destructor Documentation

7.3.3.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::__ITree (const allocator_type & __*a* = allocator_type()) `[inline, explicit]`

standard constructor

Definition at line 2091 of file vgtl_tree.h.

7.3.3.2 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::__ITree (size_type __*n*, const _Tp & __*value*, const allocator_type & __*a* = allocator_type()) `[inline]`

construct a tree containing `__n` nodes with value `__value` at the root spot.

Definition at line 2183 of file vgtl_tree.h.

7.3.3.3 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::__ITree (size_type __*n*) `[inline, explicit]`

construct a tree containing `__n` nodes with default value at the root spot.

Definition at line 2190 of file vgtl_tree.h.

7.3.3.4 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::__ITree (const _Self & __*x*) `[inline]`

copy constructor

Definition at line 2195 of file vgtl_tree.h.

7.3.3.5 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> virtual __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::~__ITree () [inline, virtual]

standard destructor

Definition at line 2198 of file vgtl_tree.h.

7.3.4 Member Function Documentation

7.3.4.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_iterative_walker __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::begin (walker_type *wt* = cw_pre_post, bool *front_to_back* = true, bool *depth_first* = true) const [inline]

the const walker to the first node of the complete walk

Definition at line 2128 of file vgtl_tree.h.

7.3.4.2 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> iterative_walker __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::begin (walker_type *wt* = cw_pre_post, bool *front_to_back* = true, bool *depth_first* = true) [inline]

the walker to the first node of the complete walk

Definition at line 2121 of file vgtl_tree.h.

7.3.4.3 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> size_type __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::depth (const iterative_walker & *__position*) [inline]

return the depth of this `__position` **in the tree**

Definition at line 2176 of file vgtl_tree.h.

7.3.4.4 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_iterative_walker __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::end (walker_type *wt* = cw_pre_post, bool *front_to_back* = true, bool *depth_first* = true) const [inline]

the const walker beyond the last node of the walk

Definition at line 2142 of file vgtl_tree.h.

7.3.4.5 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> iterative_walker __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::end (walker_type *wt* = cw_pre_post, bool *front_to_back* = true, bool *depth_first* = true) [inline]

the walker beyond the last node of the walk

Definition at line 2136 of file vgtl_tree.h.

7.3.4.6 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_reference __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::getroot () const [inline]

get a const reference to the virtual root node

Definition at line 2173 of file vgtl_tree.h.

7.3.4.7 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> reference __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::getroot () [inline]

get a reference to the virtual root node

Definition at line 2171 of file vgtl_tree.h.

7.3.4.8 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> _Self& __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::operator= (_Node * __x) [inline]

assign a tree from one node -> make this node the root node. This is useful for making trees out of erased subtrees.

Definition at line 2207 of file vgtl_tree.h.

7.3.4.9 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> _Self& __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::operator= (const _Self & __x)

standard assignment operator

Reimplemented from __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _ITree_node< _Tp, _Ctr, _Iterator >, _Alloc >.

7.3.4.10 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_reverse_iterator __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::rbegin () const [inline]

return a const reverse iterator to the first node in walk

Definition at line 2157 of file vgtl_tree.h.

7.3.4.11 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> reverse_iterator __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::rbegin () [inline]

return a reverse iterator to the first node in walk

Definition at line 2150 of file vgtl_tree.h.

7.3.4.12 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_reverse_iterator __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::rend () const [inline]

return a const reverse iterator beyond the last node in walk

Definition at line 2160 of file vgtl_tree.h.

7.3.4.13 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> reverse_iterator __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::rend () [inline]

return a reverse iterator beyond the last node in walk

Definition at line 2153 of file vgtl_tree.h.

7.3.4.14 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_iterative_walker __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::root (walker_type *wt* = cw_pre_post, bool *front_to_back* = true, bool *depth_first* = true) const [inline]

return a const iterative walker of type `wt` to the ground node

Definition at line 2105 of file vgtl_tree.h.

7.3.4.15 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> iterative_walker __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::root (walker_type *wt* = cw_pre_post, bool *front_to_back* = true, bool *depth_first* = true) `[inline]`

return an iterative walker of type `wt` to the ground node

Definition at line 2098 of file vgtl_tree.h.

7.3.4.16 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> size_type __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::size () const `[inline]`

return the size of the tree (# of nodes)

Definition at line 2164 of file vgtl_tree.h.

7.3.4.17 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_iterative_walker __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::through () const `[inline]`

the const walker beyond the complete walk

Definition at line 2116 of file vgtl_tree.h.

7.3.4.18 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> iterative_walker __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::through () `[inline]`

the walker beyond the complete walk

Definition at line 2112 of file vgtl_tree.h.

7.3.5 Friends And Related Function Documentation

7.3.5.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> bool operator==__VGTL_NULL_TMPL_ARGS (const __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc > & *__x*, const __ITree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc > & *__y*) `[friend]`

comparison operator

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

- **vgtl_tree.h**

7.4 __one_iterator Class Template Reference

make an iterator out of one pointer.

`#include <vgtl_intadapt.h>`

Public Types

- **typedef std::random_access_iterator_tag iterator_category**

  standard iterator definitions.

- **typedef ptrdiff_t difference_type**
  standard iterator definitions.
- **typedef _Tp value_type**
  standard iterator definitions.
- **typedef value_type * pointer**
  standard iterator definitions.
- **typedef value_type & reference**
  standard iterator definitions.

Public Methods

- **__one_iterator ()**
  standard constructor.
- **__one_iterator (const value_type *__x)**
  standard constructor setting the value.
- **__one_iterator (const _Self &__x)**
  copy constructor.
- **__one_iterator (const pointer &__v, bool __a)**
  constructor, explicitly setting value and iterator position.
- **reference operator * () const**
  dereference operator.

- **_Self & operator++ ()**
  standard increment, decrement, and access operators for random access.
- **_Self operator++ (int)**
  standard increment, decrement, and access operators for random access.
- **_Self & operator– ()**
  standard increment, decrement, and access operators for random access.
- **_Self operator– (int)**
  standard increment, decrement, and access operators for random access.
- **_Self operator+ (difference_type __n) const**
  standard increment, decrement, and access operators for random access.
- **_Self & operator+= (difference_type __n)**
  standard increment, decrement, and access operators for random access.

- _Self operator- (difference_type __n) const

  standard increment, decrement, and access operators for random access.

- _Self & operator-= (difference_type __n)

  standard increment, decrement, and access operators for random access.

- reference operator[ ] (difference_type __n) const

  standard increment, decrement, and access operators for random access.

- bool operator== (const _Self &__x)

  comparsion operator.

- bool operator!= (const _Self &__x)

  comparsion operator.

Protected Attributes

- pointer __value

  The single value of the 'sequence'.

- bool __at

  are we at begin()?

7.4.1 Detailed Description

template<class _Tp> class __one_iterator< _Tp >

This adaptor takes a pointer to a value of type `_Tp` and constructs an iterator, which only has two possibilities:

- **begin() points to the same place as the pointer**
- **end() is beyond the end. So a pointer is transformed to a sequence of length one, and this iterator iterates over it.**

Definition at line 209 of file vgtl_intadapt.h.

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

- vgtl_intadapt.h

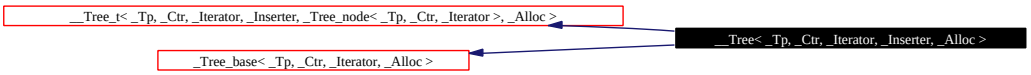
7.5 __Tree Class Template Reference

Tree base class without data hooks.

```
#include <vgtl_tree.h>
```

Inheritance diagram for __Tree:

Collaboration diagram for __Tree:



Public Types

- typedef _Tree_iterator< _Tp, _Tp &, _Tp *, container_type, children_iterator, node_type > iterator
- typedef _Tree_iterator< _Tp, const _Tp &, const _Tp *, container_type, children_iterator, node_type > const_iterator
- typedef std::reverse_iterator< const_iterator > const_reverse_iterator
- typedef std::reverse_iterator< iterator > reverse_iterator

Public Methods

- __Tree (const allocator_type &__a=allocator_type())
- walker ground ()
- const_walker ground () const
- walker root (children_iterator __it)
- const_walker root (children_iterator __it) const
- walker root ()
- const_walker root () const
- iterator begin ()
- iterator end ()
- const_iterator begin () const
- const_iterator end () const
- reverse_iterator rbegin ()
- reverse_iterator rend ()
- const_reverse_iterator rbegin () const
- const_reverse_iterator rend () const
- reference getroot ()
- const_reference getroot () const
- __Tree (size_type __n, const _Tp &__value, const allocator_type &__a=allocator_type())
- __Tree (size_type __n)
- __Tree (const _Self &__x)
- virtual ~__Tree ()
- _Self & operator= (const _Self &__x)
- _Self & operator= (_Node *__x)

Protected Methods

- void _C_put_node (_Node *__p)

Friends

- bool operator==__VGTL_NULL_TMPL_ARGS (const __Tree &__x, const __Tree &__y)

7.5.1 Detailed Description

template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> class __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >

This is the base class for all trees without data hooks

Definition at line 1233 of file vgtl_graph.h.

7.5.2 Member Typedef Documentation

7.5.2.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _Tree_iterator<_Tp,const _Tp&,const _Tp*,container_type,children_iterator,node_type> __Tree::const_iterator

the const iterator

Reimplemented from __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Tree_node< _Tp, _Ctr, _Iterator >, _Alloc >.

Definition at line 1900 of file vgtl_tree.h.

7.5.2.2 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef std::reverse_iterator<const_iterator> __Tree::const_reverse_iterator

the const reverse iterator

Reimplemented from __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Tree_node< _Tp, _Ctr, _Iterator >, _Alloc >.

Definition at line 1904 of file vgtl_tree.h.

7.5.2.3 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef _Tree_iterator<_Tp,_Tp&,_Tp*,container_type,children_iterator,node_type> __Tree::iterator

the iterator

Reimplemented from __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Tree_node< _Tp, _Ctr, _Iterator >, _Alloc >.

Definition at line 1898 of file vgtl_tree.h.

7.5.2.4 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> typedef std::reverse_iterator<iterator> __Tree::reverse_iterator

the reverse iterator

Reimplemented from __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Tree_node< _Tp, _Ctr, _Iterator >, _Alloc >.

Definition at line 1906 of file vgtl_tree.h.

7.5.3 Constructor & Destructor Documentation

7.5.3.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::__Tree (const allocator_type & __a = allocator_type()) [inline, explicit]

standard constructor

Definition at line 1931 of file vgtl_tree.h.

7.5.3.2 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::__Tree (size_type __n, const _Tp & __value, const allocator_type & __a = allocator_type()) [inline]

construct a tree containing `__n` **nodes with value** `__value` **at the root spot.**

Definition at line 2003 of file vgtl_tree.h.

7.5.3.3 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::__Tree (size_type __n) [inline, explicit]

construct a tree containing `__n` **nodes with default value at the root spot.**

Definition at line 2010 of file vgtl_tree.h.

7.5.3.4 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::__Tree (const _Self & __x) [inline]

copy constructor

Definition at line 2015 of file vgtl_tree.h.

7.5.3.5 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> virtual __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::~__Tree () [inline, virtual]

standard destructor

Definition at line 2018 of file vgtl_tree.h.

7.5.4 Member Function Documentation

7.5.4.1 template<class _Tp, class _Ctr, class _I, class _Allocator, bool _IsStatic> void _Tree_alloc_base< _Tp, _Ctr, _I, _Allocator, _IsStatic >::_C_put_node (_Node * __p) [inline, protected, inherited]

deallocate a node

Definition at line 1377 of file vgtl_tree.h.

7.5.4.2 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_iterator __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::begin () const [inline]

return a const iterator to the first node in walk

Definition at line 1972 of file vgtl_tree.h.

7.5.4.3 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> iterator __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::begin () [inline]

return an iterator to the first node in walk

Definition at line 1963 of file vgtl_tree.h.

7.5.4.4 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_iterator __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::end () const [inline]

return a const iterator beyond the last node in walk

Definition at line 1976 of file vgtl_tree.h.

7.5.4.5 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> iterator __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::end () [inline]

return an iterator beyond the last node in walk

Definition at line 1967 of file vgtl_tree.h.

7.5.4.6 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_reference __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::getroot () const [inline]

get a const reference to the virtual root node

Definition at line 1997 of file vgtl_tree.h.

7.5.4.7 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> reference __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::getroot () [inline]

get a reference to the virtual root node

Definition at line 1995 of file vgtl_tree.h.

7.5.4.8 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_walker __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::ground () const [inline]

return a const walker to the virtual root node.

Definition at line 1942 of file vgtl_tree.h.

7.5.4.9 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> walker __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::ground () [inline]

return a walker to the virtual root node.

Definition at line 1938 of file vgtl_tree.h.

7.5.4.10 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> _Self& __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::operator= (_Node * __x) [inline]

assign a tree from one node -> make this node the root node. This is useful for making trees out of erased subtrees.

Definition at line 2027 of file vgtl_tree.h.

7.5.4.11 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> _Self& __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::operator= (const _Self & __x)

standard assignment operator

Reimplemented from __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Tree_node< _Tp, _Ctr, _Iterator >, _Alloc >.

7.5.4.12 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_reverse_iterator __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::rbegin () const [inline]

return a const reverse iterator to the first node in walk

Definition at line 1988 of file vgtl_tree.h.

7.5.4.13 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> reverse_iterator __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::rbegin () [inline]

return a reverse iterator to the first node in walk

Definition at line 1981 of file vgtl_tree.h.

7.5.4.14 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_reverse_iterator __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::rend () const [inline]

return a const reverse iterator beyond the last node in walk

Definition at line 1991 of file vgtl_tree.h.

7.5.4.15 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> reverse_iterator __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::rend () [inline]

return a reverse iterator beyond the last node in walk

Definition at line 1984 of file vgtl_tree.h.

7.5.4.16 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_walker __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::root () const [inline]

return a const walker to the first non-virtual tree root

Definition at line 1959 of file vgtl_tree.h.

7.5.4.17 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> walker __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::root () [inline]

return a walker to the first non-virtual tree root

Definition at line 1956 of file vgtl_tree.h.

7.5.4.18 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> const_walker __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::root (children_iterator __it) const [inline]

return a const walker to a root node.

Definition at line 1951 of file vgtl_tree.h.

7.5.4.19 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> walker __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc >::root (children_iterator __it) [inline]

return a walker to a root node.

Definition at line 1946 of file vgtl_tree.h.

7.5.5 Friends And Related Function Documentation

7.5.5.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Alloc> bool operator==__VGTL_NULL_TMPL_ARGS (const __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc > & __x, const __Tree< _Tp, _Ctr, _Iterator, _Inserter, _Alloc > & __y) [friend]

comparison operator

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

- vgtl_graph.h
- vgtl_tree.h

7.6 __Tree_t Class Template Reference

Tree base class.

```
#include <vgtl_tree.h>
```

Inheritance diagram for __Tree_t:

Collaboration diagram for __Tree_t:

Public Types

- typedef _Tree_iterator< _Tp, _Tp &, _Tp *, container_type, children_iterator, node_type > iterator
- typedef _Tree_iterator< _Tp, const _Tp &, const _Tp *, container_type, children_iterator, node_type > const_iterator
- typedef std::reverse_iterator< const_iterator > const_reverse_iterator
- typedef std::reverse_iterator< iterator > reverse_iterator
- typedef _RTree_walker< _Tp, _Tp &, _Tp *, container_type, children_iterator, node_type > walker
- typedef _RTree_walker< _Tp, const _Tp &, const _Tp *, container_type, children_iterator, node_type > const_walker

- typedef _Tp value_type
- typedef _Node node_type
- typedef value_type * pointer

- typedef const value_type * const_pointer
- typedef value_type & reference
- typedef const value_type & const_reference
- typedef size_t size_type
- typedef ptrdiff_t difference_type

Public Methods

- allocator_type get_allocator () const
- __Tree_t (const allocator_type &__a=allocator_type())
- bool empty () const
- size_type max_size () const
- void swap (_Self &__x)
- void insert_child (const __walker_base &__position, const _Tp &__x, const container_insert_arg &__It)
- void insert_child (const __walker_base &__position, const container_insert_arg &__It)
- void insert_children (const __walker_base &__position, size_type __n, const _Tp &__x, const children_iterator &__It)
- void insert_subtree (const __walker_base &__position, _Self &__subtree, const children_iterator &__It)
- void erase (const __walker_base &__position)
- _Node * erase_tree (const __walker_base &__position)
- bool erase_child (const __walker_base &__position, const children_iterator &__It)
- _Node * erase_subtree (const __walker_base &__position, const children_iterator &__It)
- size_type depth (const walker &__position)
- void clear ()
- __Tree_t (size_type __n, const _Tp &__value, const allocator_type &__a=allocator_type())
- __Tree_t (size_type __n)
- __Tree_t (const _Self &__x)
- virtual ~__Tree_t ()
- _Self & operator= (const _Self &__x)
- _Self & operator= (_Node *__x)

Protected Methods

- _Node * _C_create_node (const _Tp &__x)
- _Node * _C_create_node ()
- void _C_put_node (_Node *__p)

7.6.1 Detailed Description

template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> class __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >

This is the toplevel base class for all trees independent of allocators

Definition at line 1558 of file vgtl_tree.h.

7.6.2 Member Typedef Documentation

7.6.2.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef _Tree_iterator<_Tp,const _Tp&,const _Tp*,container_type,children_iterator,node_type> __Tree_t::const_iterator

the const iterator

Definition at line 1592 of file vgtl_tree.h.

7.6.2.2 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef const value_type* __Tree_t::const_pointer

standard typedef

Definition at line 1577 of file vgtl_tree.h.

7.6.2.3 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef const value_type& __Tree_t::const_reference

standard typedef

Definition at line 1579 of file vgtl_tree.h.

7.6.2.4 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef std::reverse_iterator<const_iterator> __Tree_t::const_reverse_iterator

the const reverse iterator

Definition at line 1596 of file vgtl_tree.h.

7.6.2.5 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef _RTree_walker<_Tp,const _Tp&,const _Tp*,container_type,children_iterator,node_type> __Tree_t::const_walker

the (recursive) const walker

Definition at line 1613 of file vgtl_tree.h.

7.6.2.6 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef ptrdiff_t __Tree_t::difference_type

standard typedef

Definition at line 1581 of file vgtl_tree.h.

7.6.2.7 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef _Tree_iterator<_Tp,_Tp&,_Tp*,container_type,children_iterator,node_type> __Tree_t::iterator

the iterator

Definition at line 1590 of file vgtl_tree.h.

7.6.2.8 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef _Node __Tree_t::node_type

standard typedef

Definition at line 1575 of file vgtl_tree.h.

7.6.2.9 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef value_type* __Tree_t::pointer

standard typedef

Definition at line 1576 of file vgtl_tree.h.

7.6.2.10 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef value_type& __Tree_t::reference

standard typedef

Definition at line 1578 of file vgtl_tree.h.

7.6.2.11 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef std::reverse_iterator<iterator> __Tree_t::reverse_iterator

the reverse iterator

Definition at line 1598 of file vgtl_tree.h.

7.6.2.12 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef size_t __Tree_t::size_type

standard typedef

Definition at line 1580 of file vgtl_tree.h.

7.6.2.13 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef _Tp __Tree_t::value_type

standard typedef

Definition at line 1574 of file vgtl_tree.h.

7.6.2.14 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> typedef _RTree_walker<_Tp,_Tp&,_Tp*,container_type,children_iterator,node_type> __Tree_t::walker

the (recursive) walker

Definition at line 1611 of file vgtl_tree.h.

7.6.3 Constructor & Destructor Documentation

7.6.3.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::__Tree_t (const allocator_type & __a = allocator_type()) `[inline, explicit]`

standard constructor

Definition at line 1653 of file vgtl_tree.h.

7.6.3.2 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::__Tree_t (size_type __n, const _Tp & __value, const allocator_type & __a = allocator_type()) [inline]

construct a tree containing __n nodes with value __value at the root spot.

Definition at line 1822 of file vgtl_tree.h.

7.6.3.3 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::__Tree_t (size_type __n) [inline, explicit]

construct a tree containing __n nodes with default value at the root spot.

Definition at line 1829 of file vgtl_tree.h.

7.6.3.4 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::__Tree_t (const _Self & __x) [inline]

copy constructor

Definition at line 1848 of file vgtl_tree.h.

7.6.3.5 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> virtual __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::~__Tree_t () [inline, virtual]

standard destructor

Definition at line 1857 of file vgtl_tree.h.

7.6.4 Member Function Documentation

7.6.4.1 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> _Node* __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::_C_create_node () [inline, protected]

construct a new tree node containing default data

Definition at line 1640 of file vgtl_tree.h.

7.6.4.2 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> _Node* __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::_C_create_node (const _Tp & __x) [inline, protected]

construct a new tree node containing data __x

Definition at line 1628 of file vgtl_tree.h.

7.6.4.3 template<class _Tp, class _Ctr, class _I, class _Allocator, bool _IsStatic> void _Tree_alloc_base< _Tp, _Ctr, _I, _Allocator, _IsStatic >::_C_put_node (_Node * __p) [inline, protected, inherited]

deallocate a node

Definition at line 1377 of file vgtl_tree.h.

7.6.4.4 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> void __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::clear () [inline]

empty the tree

Reimplemented from _Tree_base< _Tp, _Ctr, _Iterator, _Node, _Alloc >.

Definition at line 1816 of file vgtl_tree.h.

7.6.4.5 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> size_type __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::depth (const walker & __*position*) [inline]

return the depth of node __position in the tree

Definition at line 1804 of file vgtl_tree.h.

7.6.4.6 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> bool __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::empty () const [inline]

is the tree empty?

Definition at line 1656 of file vgtl_tree.h.

7.6.4.7 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> void __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::erase (const __walker_base & __*position*) [inline]

erase the node at position __position.

Definition at line 1712 of file vgtl_tree.h.

7.6.4.8 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> bool __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::erase_child (const __walker_base & __*position*, const children_iterator & __*It*) [inline]

erase the (leaf) child __It of node __position. This works if and only if the child is a leaf.

Definition at line 1769 of file vgtl_tree.h.

7.6.4.9 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> _Node* __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::erase_subtree (const __walker_base & __*position*, const children_iterator & __*It*) [inline]

erase the subtree position __position, whose top node is the child at children_iterator position __It, and return its top node.

Definition at line 1789 of file vgtl_tree.h.

7.6.4.10 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> _Node* __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::erase_tree (const __walker_base & __*position*) [inline]

erase the subtree starting at position __position, and return its top node.

Definition at line 1742 of file vgtl_tree.h.

7.6.4.11 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> allocator_type __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::get_allocator () const `[inline]`

construct an allocator object

Reimplemented from _Tree_alloc_base.

Definition at line 1586 of file vgtl_tree.h.

7.6.4.12 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> void __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::insert_child (const __walker_base & *__position*, const container_insert_arg & *__It*) `[inline]`

add a child below `__position` **with default data, at the** `__It` **position in the** `__position` **- node's children container**

Definition at line 1675 of file vgtl_tree.h.

7.6.4.13 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> void __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::insert_child (const __walker_base & *__position*, const _Tp & *__x*, const container_insert_arg & *__It*) `[inline]`

add a child below `__position` **with data** `__x`**, at the** `__It` **position in the** `__position` **- node's children container**

Definition at line 1667 of file vgtl_tree.h.

7.6.4.14 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> void __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::insert_children (const __walker_base & *__position*, size_type *__n*, const _Tp & *__x*, const children_iterator & *__It*) `[inline]`

add `__n` **children below** `__position` **with data** `__x`**, after the** `__It` **position in the** `__position` **- node's children container**

Definition at line 1681 of file vgtl_tree.h.

7.6.4.15 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> void __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::insert_subtree (const __walker_base & *__position*, _Self & *__subtree*, const children_iterator & *__It*) `[inline]`

add a complete subtree `__subtree` **below position** `__position` **and children iterator position** `__It`**.**

Definition at line 1701 of file vgtl_tree.h.

7.6.4.16 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> size_type __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::max_size () const `[inline]`

return the maximum possible size of the tree (theor. infinity)

Definition at line 1659 of file vgtl_tree.h.

7.6.4.17 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> _Self& __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::operator= (_Node * *__x*) `[inline]`

assign a tree from one node -> make this node the root node. This is useful for making trees out of erased subtrees.

Definition at line 1866 of file vgtl_tree.h.

7.6.4.18 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> _Self& __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::operator= (const _Self & __x)

standard assignment operator

7.6.4.19 template<class _Tp, class _Ctr, class _Iterator, class _Inserter, class _Node, class _Alloc> void __Tree_t< _Tp, _Ctr, _Iterator, _Inserter, _Node, _Alloc >::swap (_Self & __x) [inline]

swap two trees

Definition at line 1662 of file vgtl_tree.h.

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

- **vgtl_tree.h**

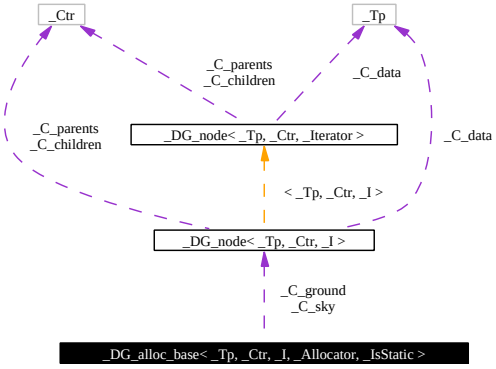
7.7 _DG_alloc_base Class Template Reference

Directed graph base class for general standard-conforming allocators.

`#include <vgtl_dagbase.h>`

Inheritance diagram for _DG_alloc_base:

Collaboration diagram for _DG_alloc_base:



Protected Methods

- **_DG_node< _Tp, _Ctr, _I > * _C_get_node ()**
- **void _C_put_node (_DG_node< _Tp, _Ctr, _I > *__p)**

Protected Attributes

- _DG_node< _Tp, _Ctr, _I > * _C_ground
- _DG_node< _Tp, _Ctr, _I > * _C_sky
- int _C_mark

7.7.1 Detailed Description

template<class _Tp, class _Ctr, class _I, class _Allocator, bool _IsStatic> class _DG_alloc_base< _Tp, _Ctr, _I, _Allocator, _IsStatic >

Base directed graph class that encapsulates details of allocators. Three cases: an ordinary standard-conforming allocator, a standard-conforming allocator with no non-static data, and an SGI-style allocator. This complexity is necessary only because we're worrying about STL compatibility and because we want to avoid wasting storage on an allocator instance if it isn't necessary. Base for general standard-conforming allocators.

Definition at line 184 of file vgtl_dagbase.h.

7.7.2 Member Function Documentation

7.7.2.1 template<class _Tp, class _Ctr, class _I, class _Allocator, bool _IsStatic> _DG_node<_Tp,_Ctr,_I>* _DG_alloc_base< _Tp, _Ctr, _I, _Allocator, _IsStatic >::_C_get_node () [inline, protected]

allocates the memory of one node

Definition at line 194 of file vgtl_dagbase.h.

7.7.2.2 template<class _Tp, class _Ctr, class _I, class _Allocator, bool _IsStatic> void _DG_alloc_base< _Tp, _Ctr, _I, _Allocator, _IsStatic >::_C_put_node (_DG_node< _Tp, _Ctr, _I > * *_p*) [inline, protected]

de-allocates the memory of one node

Definition at line 197 of file vgtl_dagbase.h.

7.7.3 Member Data Documentation

7.7.3.1 template<class _Tp, class _Ctr, class _I, class _Allocator, bool _IsStatic> _DG_node<_Tp,_Ctr,_I>* _DG_alloc_base::_C_ground [protected]

the virtual ground node (below all roots)

Definition at line 206 of file vgtl_dagbase.h.

7.7.3.2 template<class _Tp, class _Ctr, class _I, class _Allocator, bool _IsStatic> int _DG_alloc_base::_C_mark [protected]

internal counter for various algorithms

Definition at line 210 of file vgtl_dagbase.h.

7.7.3.3 template<class _Tp, class _Ctr, class _I, class _Allocator, bool _IsStatic> _DG_node<_Tp,_Ctr,_I>* _DG_alloc_base::_C_sky [protected]

the virtual sky node (above all leaves)

Definition at line 208 of file vgtl_dagbase.h.

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

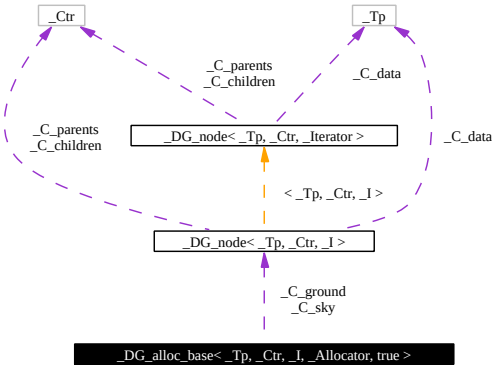
- **vgtl_dagbase.h**

7.8 _DG_alloc_base< _Tp, _Ctr, _I, _Allocator, true > Class Template Reference

Directed graph base class specialization for instanceless allocators.

`#include <vgtl_dagbase.h>`

Collaboration diagram for _DG_alloc_base< _Tp, _Ctr, _I, _Allocator, true >:



Protected Methods

- **_DG_node< _Tp, _Ctr, _I > * _C_get_node ()**
- **void _C_put_node (_DG_node< _Tp, _Ctr, _I > *__p)**

Protected Attributes

- **_DG_node< _Tp, _Ctr, _I > * _C_ground**
- **_DG_node< _Tp, _Ctr, _I > * _C_sky**
- **int _C_mark**

7.8.1 Detailed Description

template<class _Tp, class _Ctr, class _I, class _Allocator> class _DG_alloc_base< _Tp, _Ctr, _I, _Allocator, true >

Base directed graph class that encapsulates details of allocators. Three cases: an ordinary standard-conforming allocator, a standard-conforming allocator with no non-static data, and an SGI-style allocator. This complexity is necessary only because we're worrying about STL compatibility and

because we want to avoid wasting storage on an allocator instance if it isn't necessary. Base class specialization for instanceless allocators.

Definition at line 226 of file vgtl_dagbase.h.

7.8.2 Member Function Documentation

7.8.2.1 template<class _Tp, class _Ctr, class _I, class _Allocator> _DG_node<_Tp,_Ctr,_I>* _DG_alloc_base< _Tp, _Ctr, _I, _Allocator, true >::_C_get_node () [inline, protected]

allocate a new node

Definition at line 238 of file vgtl_dagbase.h.

7.8.2.2 template<class _Tp, class _Ctr, class _I, class _Allocator> void _DG_alloc_base< _Tp, _Ctr, _I, _Allocator, true >::_C_put_node (_DG_node< _Tp, _Ctr, _I > * *_p*) [inline, protected]

deallocate a node

Definition at line 241 of file vgtl_dagbase.h.

7.8.3 Member Data Documentation

7.8.3.1 template<class _Tp, class _Ctr, class _I, class _Allocator> _DG_node<_Tp,_Ctr,_I>* _DG_alloc_base< _Tp, _Ctr, _I, _Allocator, true >::_C_ground [protected]

This is the virtual node ground (below all roots)

Definition at line 246 of file vgtl_dagbase.h.

7.8.3.2 template<class _Tp, class _Ctr, class _I, class _Allocator> int _DG_alloc_base< _Tp, _Ctr, _I, _Allocator, true >::_C_mark [protected]

internal counter for various algorithms

Definition at line 250 of file vgtl_dagbase.h.

7.8.3.3 template<class _Tp, class _Ctr, class _I, class _Allocator> _DG_node<_Tp,_Ctr,_I>* _DG_alloc_base< _Tp, _Ctr, _I, _Allocator, true >::_C_sky [protected]

This is the virtual node sky (above all leafs)

Definition at line 248 of file vgtl_dagbase.h.

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

- **vgtl_dagbase.h**

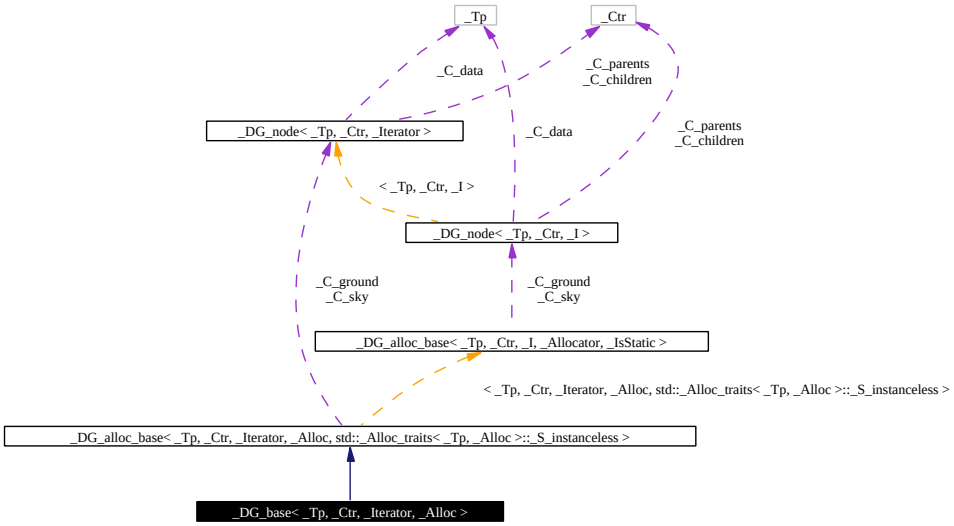
7.9 _DG_base Class Template Reference

Directed graph base class for allocator encapsulation.

```
#include <vgtl_dagbase.h>
```

Inheritance diagram for _DG_base:

Collaboration diagram for _DG_base:



Public Types

- **typedef _Base::allocator_type allocator_type**
- **typedef _Ctr container_type**
- **typedef _Iterator children_iterator**
- **typedef _Iterator parents_iterator**

Public Methods

- **_DG_base (const allocator_type &__a)**
- **~_DG_base ()**
- **void clear ()**
- **void clear_children ()**
- **void clear_parents ()**
- **template<class _Output_Iterator> void add_all_children (_Output_Iterator fi, _DG_node< _Tp, _Ctr, _Iterator > *_parent)**
- **template<class _Output_Iterator> void add_all_parents (_Output_Iterator fi, _DG_node< _Tp, _Ctr, _Iterator > *_child)**

Protected Methods

- **void clear_graph (_DG_node< _Tp, _Ctr, _Iterator > *_node)**

7.9.1 Detailed Description

template<class _Tp, class _Ctr, class _Iterator, class _Alloc> class _DG_base< _Tp, _Ctr, _Iterator, _Alloc >

Base directed graph class top level that encapsulates details of allocators.

Definition at line 260 of file vgtl_dagbase.h.

7.9.2 Member Typedef Documentation

7.9.2.1 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> typedef _Base::allocator_type _DG_base::allocator_type

allocator type

Reimplemented from _DG_alloc_base< _Tp, _Ctr, _Iterator, _Alloc, std::_Alloc_traits< _Tp, _Alloc >::_S_instanceless >.

Reimplemented in __DG.

Definition at line 269 of file vgtl_dagbase.h.

7.9.2.2 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> typedef _Iterator _DG_base::children_iterator

iterator for accessing the children

Reimplemented in __DG.

Definition at line 274 of file vgtl_dagbase.h.

7.9.2.3 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> typedef _Ctr _DG_base::container_type

internal container used to store the children and parents

Reimplemented in __DG.

Definition at line 272 of file vgtl_dagbase.h.

7.9.2.4 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> typedef _Iterator _DG_base::parents_iterator

iterator for accessing the parents

Reimplemented in __DG.

Definition at line 276 of file vgtl_dagbase.h.

7.9.3 Constructor & Destructor Documentation

7.9.3.1 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> _DG_base< _Tp, _Ctr, _Iterator, _Alloc >::_DG_base (const allocator_type & __*a*) [inline]

constructor initializing the allocator and the root

Definition at line 280 of file vgtl_dagbase.h.

7.9.3.2 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> _DG_base< _Tp, _Ctr, _Iterator, _Alloc >::~_DG_base () [inline]

standard destructor

Definition at line 300 of file vgtl_dagbase.h.

7.9.4 Member Function Documentation

7.9.4.1 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> template<class _Output_Iterator> void _DG_base< _Tp, _Ctr, _Iterator, _Alloc >::add_all_children (_Output_Iterator *fi*, _DG_node< _Tp, _Ctr, _Iterator > * *_parent*) [inline]

add all children to the parent `_parent`. `fi` **is a iterator to the children container of the parent**

Definition at line 459 of file vgtl_dagbase.h.

7.9.4.2 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> template<class _Output_Iterator> void _DG_base< _Tp, _Ctr, _Iterator, _Alloc >::add_all_parents (_Output_Iterator *fi*, _DG_node< _Tp, _Ctr, _Iterator > * *_child*) [inline]

add all parents to the child `_child`. `fi` **is a iterator to the container of the child**

Definition at line 466 of file vgtl_dagbase.h.

7.9.4.3 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> void _DG_base< _Tp, _Ctr, _Iterator, _Alloc >::clear ()

empty the tree

Reimplemented in _DG.

Definition at line 472 of file vgtl_dagbase.h.

7.9.4.4 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> void _DG_base< _Tp, _Ctr, _Iterator, _Alloc >::clear_children () [inline]

clear all children of the ground node

Definition at line 314 of file vgtl_dagbase.h.

7.9.4.5 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> void _DG_base< _Tp, _Ctr, _Iterator, _Alloc >::clear_graph (_DG_node< _Tp, _Ctr, _Iterator > * *_node*) [protected]

removes all the nodes of the graph except the sky and ground nodes

Definition at line 430 of file vgtl_dagbase.h.

7.9.4.6 template<class _Tp, class _Ctr, class _Iterator, class _Alloc> void _DG_base< _Tp, _Ctr, _Iterator, _Alloc >::clear_parents () [inline]

clear all parents of the sky node

Definition at line 317 of file vgtl_dagbase.h.

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

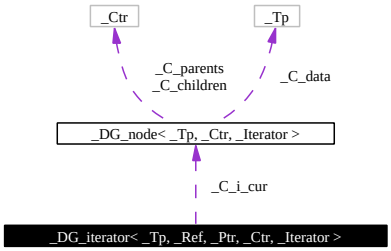
- **vgtl_dagbase.h**

7.10 _DG_iterator Class Template Reference

iterator through the directed graph.

`#include <vgtl_dag.h>`

Collaboration diagram for _DG_iterator:



Public Types

- typedef std::bidirectional_iterator_tag iterator_category
- typedef _Tp value_type
- typedef _Ptr pointer
- typedef _Ref reference
- typedef _DG_node< _Tp, _Ctr, _Iterator > _Node
- typedef size_t size_type
- typedef ptrdiff_t difference_type

Public Methods

- _DG_iterator ()
- _DG_iterator (const iterator &__x)
- reference operator * () const
- pointer operator → () const
- _Self & operator= (const _Walk &__x)

- bool operator== (const _Self &__x) const
- bool operator!= (const _Self &__x) const

- _Self & operator++ ()
- _Self operator++ (int)
- _Self & operator– ()
- _Self operator– (int)

Protected Attributes

- _Node * _C_i_cur
- std::vector< _Ctr_iterator > _C_i_cur_it

7.10.1 Detailed Description

template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> class _DG_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >

This is an iterator, which visits each node of a directed graph once. It is based on a preorder depth-first automatic walker which visits a child if and only if the parent is the first in the list.

Definition at line 235 of file vgtl_dag.h.

7.10.2 Member Typedef Documentation

7.10.2.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef _DG_node<_Tp,_Ctr,_Iterator> _DG_iterator::_Node

standard iterator definition

Definition at line 249 of file vgtl_dag.h.

7.10.2.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef ptrdiff_t _DG_iterator::difference_type

standard iterator definition

Definition at line 251 of file vgtl_dag.h.

7.10.2.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef std::bidirectional_iterator_tag _DG_iterator::iterator_category

standard iterator definition

Definition at line 245 of file vgtl_dag.h.

7.10.2.4 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef _Ptr _DG_iterator::pointer

standard iterator definition

Definition at line 247 of file vgtl_dag.h.

7.10.2.5 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef _Ref _DG_iterator::reference

standard iterator definition

Definition at line 248 of file vgtl_dag.h.

7.10.2.6 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef size_t _DG_iterator::size_type

standard iterator definition

Definition at line 250 of file vgtl_dag.h.

7.10.2.7 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef _Tp _DG_-iterator::value_type

standard iterator definition

Definition at line 246 of file vgtl_dag.h.

7.10.3 Constructor & Destructor Documentation

7.10.3.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _DG_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::_DG_iterator () [inline]

standard constructor

Definition at line 263 of file vgtl_dag.h.

7.10.3.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _DG_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::_DG_iterator (const iterator & __x) [inline]

copy constructor

Definition at line 265 of file vgtl_dag.h.

7.10.4 Member Function Documentation

7.10.4.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> reference _DG_-iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator * () const [inline]

dereference operator

Definition at line 288 of file vgtl_dag.h.

7.10.4.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> bool _DG_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator!= (const _Self & __x) const [inline]

comparison operator

Definition at line 278 of file vgtl_dag.h.

7.10.4.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self _DG_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator++ (int) [inline]

in(de)crement operator

Definition at line 320 of file vgtl_dag.h.

7.10.4.4 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _DG_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator++ () [inline]

in(de)crement operator

Definition at line 316 of file vgtl_dag.h.

7.10.4.5 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self _DG_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator– (int) [inline]

in(de)crement operator

Definition at line 330 of file vgtl_dag.h.

7.10.4.6 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _DG_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator– () [inline]

in(de)crement operator

Definition at line 326 of file vgtl_dag.h.

7.10.4.7 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> pointer _DG_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator → () const [inline]

pointer operator

Definition at line 292 of file vgtl_dag.h.

7.10.4.8 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _DG_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator= (const _Walk & __*x*) [inline]

assignment to iterator from walker

Definition at line 305 of file vgtl_dag.h.

7.10.4.9 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> bool _DG_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator== (const _Self & __*x*) const [inline]

comparison operator

Definition at line 270 of file vgtl_dag.h.

7.10.5 Member Data Documentation

7.10.5.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Node* _DG_iterator::_C_i_cur [protected]

The current node

Definition at line 257 of file vgtl_dag.h.

7.10.5.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> std::vector<_Ctr_iterator> _DG_iterator::_C_i_cur_it [protected]

The internal stack

Definition at line 259 of file vgtl_dag.h.

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

- **vgtl_dag.h**

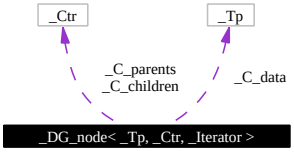
7.11 _DG_node Class Template Reference

directed graph node.

`#include <vgtl_dagbase.h>`

Inheritance diagram for _DG_node:

Collaboration diagram for _DG_node:



Public Methods

- **_DG_node ()**
- **~_DG_node ()**
- **void clear_children ()**
- **void clear_parents ()**
- **_Ctr_iterator get_childentry_iterator (const _Void_pointer __p)**
- **_Ctr_iterator get_parententry_iterator (const _Void_pointer __p)**
- **template<class _Output_Iterator> void add_all_children (_Output_Iterator fi, _Self *_parent)**
- **template<class _Output_Iterator> void add_all_parents (_Output_Iterator fi, _Self *_child)**
- **template<class Compare> void sort_child_edges (_Ctr_iterator first, _Ctr_iterator last, Compare comp)**
- **template<class Compare> void sort_parent_edges (_Ctr_iterator first, _Ctr_iterator last, Compare comp)**

Public Attributes

- **_Tp _C_data**
- **_Ctr _C_parents**
- **_Ctr _C_children**
- **int _C_visited**

7.11.1 Detailed Description

template<class _Tp, class _Ctr, class _Iterator> class _DG_node< _Tp, _Ctr, _Iterator >

This is the node for a directed graph

Definition at line 44 of file vgtl_dagbase.h.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 template<class _Tp, class _Ctr, class _Iterator> _DG_node< _Tp, _Ctr, _Iterator >::_DG_node () [inline]

standard constructor

Definition at line 62 of file vgtl_dagbase.h.

7.11.2.2 template<class _Tp, class _Ctr, class _Iterator> _DG_node< _Tp, _Ctr, _Iterator >::~_DG_node () [inline]

standard destructor

Definition at line 73 of file vgtl_dagbase.h.

7.11.3 Member Function Documentation

7.11.3.1 template<class _Tp, class _Ctr, class _Iterator> template<class _Output_Iterator> void _DG_node< _Tp, _Ctr, _Iterator >::add_all_children (_Output_Iterator *fi*, _Self * *_parent*) [inline]

add all children to parent `_parent`. `fi` **is an iterator to the children container of** `_parent`

Definition at line 142 of file vgtl_dagbase.h.

7.11.3.2 template<class _Tp, class _Ctr, class _Iterator> template<class _Output_Iterator> void _DG_node< _Tp, _Ctr, _Iterator >::add_all_parents (_Output_Iterator *fi*, _Self * *_child*) [inline]

add all parents to child `_child`. `fi` **is an iterator to the parents container of** `_child`

Definition at line 157 of file vgtl_dagbase.h.

7.11.3.3 template<class _Tp, class _Ctr, class _Iterator> void _DG_node< _Tp, _Ctr, _Iterator >::clear_children () [inline]

erase all children entries

Definition at line 80 of file vgtl_dagbase.h.

7.11.3.4 template<class _Tp, class _Ctr, class _Iterator> void _DG_node< _Tp, _Ctr, _Iterator >::clear_parents () [inline]

erase all parents entries

Definition at line 83 of file vgtl_dagbase.h.

7.11.3.5 template<class _Tp, class _Ctr, class _Iterator> _Ctr_iterator _DG_node< _Tp, _Ctr, _Iterator >::get_childentry_iterator (const _Void_pointer *__p*) [inline]

find the iterator into the children container for child `__p`

Definition at line 87 of file vgtl_dagbase.h.

7.11.3.6 template<class _Tp, class _Ctr, class _Iterator> _Ctr_iterator _DG_node< _Tp, _Ctr, _Iterator >::get_parententry_iterator (const _Void_pointer __p) [inline]

find the iterator into the parents container for parent __p

Definition at line 96 of file vgtl_dagbase.h.

7.11.3.7 template<class _Tp, class _Ctr, class _Iterator> template<class Compare> void _DG_node< _Tp, _Ctr, _Iterator >::sort_child_edges (_Ctr_iterator *first*, _Ctr_iterator *last*, Compare *comp*) [inline]

sort the children according to comp

Definition at line 123 of file vgtl_dagbase.h.

7.11.3.8 template<class _Tp, class _Ctr, class _Iterator> template<class Compare> void _DG_node< _Tp, _Ctr, _Iterator >::sort_parent_edges (_Ctr_iterator *first*, _Ctr_iterator *last*, Compare *comp*) [inline]

sort the parents according to comp

Definition at line 130 of file vgtl_dagbase.h.

7.11.4 Member Data Documentation

7.11.4.1 template<class _Tp, class _Ctr, class _Iterator> _Ctr _DG_node::_C_children

the edges to the children

Definition at line 57 of file vgtl_dagbase.h.

7.11.4.2 template<class _Tp, class _Ctr, class _Iterator> _Tp _DG_node::_C_data

the node data

Definition at line 53 of file vgtl_dagbase.h.

7.11.4.3 template<class _Tp, class _Ctr, class _Iterator> _Ctr _DG_node::_C_parents

the edges to the parents

Definition at line 55 of file vgtl_dagbase.h.

7.11.4.4 template<class _Tp, class _Ctr, class _Iterator> int _DG_node::_C_visited

internal counter for marks in algorithms

Definition at line 59 of file vgtl_dagbase.h.

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

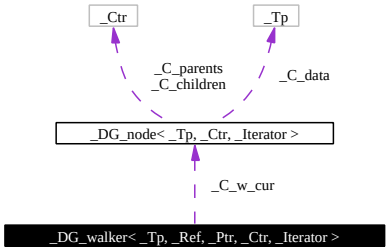
- vgtl_dagbase.h

7.12 _DG_walker Class Template Reference

recursive directed graph walkers.

`#include <vgtl_dag.h>`

Collaboration diagram for _DG_walker:



Public Types

- typedef _Tp value_type
- typedef _Ptr pointer
- typedef _Ref reference

- typedef _Ctr_iterator children_iterator
- typedef _Ctr_iterator parents_iterator
- typedef _Node node_type
- typedef size_t size_type
- typedef ptrdiff_t difference_type

Public Methods

- _DG_walker ()
- _DG_walker (_Node *__x)
- _DG_walker (const walker &__x)
- reference operator * () const
- pointer operator → () const
- const _Node * node ()
- size_type n_children () const
- size_type n_parents () const
- bool is_root () const
- bool is_leaf () const
- bool is_ground () const
- bool is_sky () const
- children_iterator child_begin ()
- children_iterator child_end ()
- parents_iterator parent_begin ()
- parents_iterator parent_end ()
- template<class _Function> _Function for_each_child (_Function __f)
- template<class _Function> _Function for_each_parent (_Function __f)
- _Self operator<< (parents_iterator __i)
- _Self operator>> (children_iterator __i)
- _Self & operator<<= (parents_iterator __i)
- _Self & operator>>= (children_iterator __i)

- _Self & operator= (const _Itr &__x)
- _Self & operator= (const _Self &__x)
- _Self & operator= (const _Node &__n)

- bool operator== (const _Self &__x) const
- bool operator!= (const _Self &__x) const

Public Attributes

- _Node * _C_w_cur

7.12.1 Detailed Description

template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> class _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >

This is the class defining reursive directed graph walkers, which walk directed graphs under guidance.

Definition at line 59 of file vgtl_dag.h.

7.12.2 Member Typedef Documentation

7.12.2.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef _Ctr_iterator _DG_walker::children_iterator

standard walker definition

Definition at line 84 of file vgtl_dag.h.

7.12.2.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef ptrdiff_t _DG_walker::difference_type

standard walker definition

Definition at line 89 of file vgtl_dag.h.

7.12.2.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef _Node _DG_walker::node_type

standard walker definition

Definition at line 86 of file vgtl_dag.h.

7.12.2.4 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef _Ctr_iterator _DG_walker::parents_iterator

standard walker definition

Definition at line 85 of file vgtl_dag.h.

7.12.2.5 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef _Ptr _DG_walker::pointer

standard walker definition

Definition at line 73 of file vgtl_dag.h.

7.12.2.6 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef _Ref _DG_walker::reference

standard walker definition

Definition at line 74 of file vgtl_dag.h.

7.12.2.7 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef size_t _DG_walker::size_type

standard walker definition

Definition at line 88 of file vgtl_dag.h.

7.12.2.8 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef _Tp _DG_walker::value_type

standard walker definition

Definition at line 72 of file vgtl_dag.h.

7.12.3 Constructor & Destructor Documentation

7.12.3.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::_DG_walker () [inline]

standard constructor

Definition at line 98 of file vgtl_dag.h.

7.12.3.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::_DG_walker (_Node * __x) [inline]

constructor setting the position

Definition at line 102 of file vgtl_dag.h.

7.12.3.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::_DG_walker (const walker & __x) [inline]

copy constructor

Definition at line 105 of file vgtl_dag.h.

7.12.4 Member Function Documentation

7.12.4.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> children_iterator _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::child_begin () [inline]

return children_iterator to first child

Definition at line 151 of file vgtl_dag.h.

7.12.4.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> children_iterator _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::child_end () [inline]

return children_iterator beyond last child

Definition at line 153 of file vgtl_dag.h.

7.12.4.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> template<class _Function> _Function _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::for_each_child (_Function __f) [inline]

apply the function __f to all children

Definition at line 162 of file vgtl_dag.h.

7.12.4.4 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> template<class _Function> _Function _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::for_each_parent (_Function __f) [inline]

apply the function __f to all parents

Definition at line 168 of file vgtl_dag.h.

7.12.4.5 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> bool _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::is_ground () const [inline]

is this node a virtual node - the ground (below all roots)?

Definition at line 146 of file vgtl_dag.h.

7.12.4.6 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> bool _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::is_leaf () const [inline]

is this node a leaf?

Definition at line 135 of file vgtl_dag.h.

7.12.4.7 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> bool _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::is_root () const [inline]

is this node a root?

Definition at line 125 of file vgtl_dag.h.

7.12.4.8 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> bool _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::is_sky () const [inline]

is this node a virtual node - the sky (above all leafs)?

Definition at line 148 of file vgtl_dag.h.

7.12.4.9 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> size_type _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::n_children () const [inline]

return the number of children

Definition at line 120 of file vgtl_dag.h.

7.12.4.10 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> size_type _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::n_parents () const [inline]

return the number of parents

Definition at line 122 of file vgtl_dag.h.

7.12.4.11 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> const _Node* _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::node () [inline]

retrieve the full node

Definition at line 117 of file vgtl_dag.h.

7.12.4.12 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> reference _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator * () const [inline]

dereference operator

Definition at line 108 of file vgtl_dag.h.

7.12.4.13 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> bool _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator!= (const _Self & __x) const [inline]

comparison operator

Definition at line 178 of file vgtl_dag.h.

7.12.4.14 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> pointer _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator → () const [inline]

pointer operator

Definition at line 112 of file vgtl_dag.h.

7.12.4.15 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator<< (parents_iterator __i) [inline]

this function returns the walker pointing to the required parent

Definition at line 183 of file vgtl_dag.h.

7.12.4.16 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator<<= (parents_iterator __i) [inline]

here the original walker goes to the required parent

Definition at line 197 of file vgtl_dag.h.

7.12.4.17 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator= (const _Node & __n) `[inline]`

a walker is assigned to any pointer to a graph node

Definition at line 221 of file vgtl_dag.h.

7.12.4.18 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator= (const _Self & __x) `[inline]`

standard assignment operator

Definition at line 215 of file vgtl_dag.h.

7.12.4.19 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator= (const _Itr & __x) `[inline]`

new walker is assigned from that particular iterator

Definition at line 209 of file vgtl_dag.h.

7.12.4.20 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> bool _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator== (const _Self & __x) const `[inline]`

comparison operator

Definition at line 176 of file vgtl_dag.h.

7.12.4.21 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator>> (children_iterator __i) `[inline]`

this functions returns the walker pointing to the required child

Definition at line 190 of file vgtl_dag.h.

7.12.4.22 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator>>= (children_iterator __i) `[inline]`

here the original walker goes to the required child

Definition at line 203 of file vgtl_dag.h.

7.12.4.23 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> parents_iterator _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::parent_begin () `[inline]`

return parents_iterator to first parent

Definition at line 156 of file vgtl_dag.h.

7.12.4.24 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> parents_iterator _DG_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::parent_end () `[inline]`

return parents_iterator beyond last parent

Definition at line 158 of file vgtl_dag.h.

7.12.5 Member Data Documentation

7.12.5.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Node* _DG_walker::_C_w_cur

pointer to the current node

Definition at line 94 of file vgtl_dag.h.

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

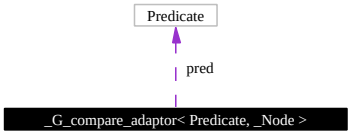
- vgtl_dag.h

7.13 _G_compare_adaptor Class Template Reference

Adaptor for data comparison in graph nodes.

```
#include <vgtl_intadapt.h>
```

Collaboration diagram for _G_compare_adaptor:



Public Methods

- _G_compare_adaptor (const Predicate &__p)
  constructor.
- bool operator() (const void *r, const void *l) const
  make it a function object on the nodes.

7.13.1 Detailed Description

template<class Predicate, class _Node> class _G_compare_adaptor< Predicate, _Node >

This adaptor takes a binary predicate for node data and transforms it to a binary predicate on the nodes.

Definition at line 316 of file vgtl_intadapt.h.

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

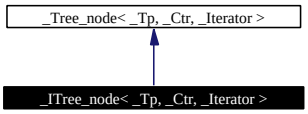
- vgtl_intadapt.h

7.14 _ITree_node Class Template Reference

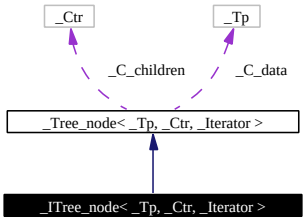
tree node for trees with data hooks.

`#include <vgtl_tree.h>`

Inheritance diagram for _ITree_node:



Collaboration diagram for _ITree_node:



Public Methods

- **_ITree_node ()**
- **void initialize ()**
- **void get_rid_of ()**
- **ctree_data_hook & data_hook ()**
- **void clear_tree ()**
- **void clear_children ()**
- **_Ctr_iterator get_childentry_iterator (_Void_pointer __p)**
- **template<class _Output_Iterator> void add_all_children (_Output_Iterator fi, _Self *_parent)**
- **template<class Compare> void sort_children (_Ctr_iterator first, _Ctr_iterator last, Compare comp)**
- **template<class Compare> void sort_parents (_Ctr_iterator first, _Ctr_iterator last, Compare comp)**

Public Attributes

- **ctree_data_hook _C_data_hook**
- **_Tp _C_data**
- **_Void_pointer _C_parent**
- **_Ctr _C_children**

7.14.1 Detailed Description

template<class _Tp, class _Ctr, class _Iterator> class _ITree_node< _Tp, _Ctr, _Iterator >

This is the tree node for a tree with data hooks

Definition at line 138 of file vgtl_tree.h.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 template<class _Tp, class _Ctr, class _Iterator> _ITree_node< _Tp, _Ctr, _Iterator >::_ITree_node () [inline]

standard constructor

Definition at line 150 of file vgtl_tree.h.

7.14.3 Member Function Documentation

7.14.3.1 template<class _Tp, class _Ctr, class _Iterator> template<class _Output_Iterator> void _Tree_node< _Tp, _Ctr, _Iterator >::add_all_children (_Output_Iterator *fi*, _Self * *_parent*) [inherited]

add all children to parent `_parent`. `fi` **is an iterator to the children container of** `_parent`

Definition at line 180 of file vgtl_tree.h.

7.14.3.2 template<class _Tp, class _Ctr, class _Iterator> void _Tree_node< _Tp, _Ctr, _Iterator >::clear_children () [inline, inherited]

erase all children entries

Definition at line 100 of file vgtl_tree.h.

7.14.3.3 template<class _Tp, class _Ctr, class _Iterator> void _Tree_node< _Tp, _Ctr, _Iterator >::clear_tree () [inherited]

remove the whole subtree below this node

Definition at line 195 of file vgtl_tree.h.

7.14.3.4 template<class _Tp, class _Ctr, class _Iterator> ctree_data_hook& _ITree_node< _Tp, _Ctr, _Iterator >::data_hook () [inline]

return the data of the data hook

Definition at line 171 of file vgtl_tree.h.

7.14.3.5 template<class _Tp, class _Ctr, class _Iterator> _Ctr_iterator _Tree_node< _Tp, _Ctr, _Iterator >::get_childentry_iterator (_Void_pointer __*p*) [inline, inherited]

find the iterator into the children container for child `__p`

Definition at line 104 of file vgtl_tree.h.

7.14.3.6 template<class _Tp, class _Ctr, class _Iterator> void _ITree_node< _Tp, _Ctr, _Iterator >::get_rid_of () [inline]

remove the children container

Reimplemented from _Tree_node.

Definition at line 165 of file vgtl_tree.h.

7.14.3.7 template<class _Tp, class _Ctr, class _Iterator> void _ITree_node< _Tp, _Ctr, _Iterator >::initialize () [inline]

initialize the data structure

Reimplemented from _Tree_node.

Definition at line 158 of file vgtl_tree.h.

7.14.3.8 template<class _Tp, class _Ctr, class _Iterator> template<class Compare> void _Tree_node< _Tp, _Ctr, _Iterator >::sort_children (_Ctr_iterator *first*, _Ctr_iterator *last*, Compare *comp*) [inline, inherited]

sort the children according to `comp`

Definition at line 121 of file vgtl_tree.h.

7.14.3.9 template<class _Tp, class _Ctr, class _Iterator> template<class Compare> void _Tree_node< _Tp, _Ctr, _Iterator >::sort_parents (_Ctr_iterator *first*, _Ctr_iterator *last*, Compare *comp*) [inline, inherited]

sort the children according to `comp`**, i.e. do nothing here**

Definition at line 128 of file vgtl_tree.h.

7.14.4 Member Data Documentation

7.14.4.1 template<class _Tp, class _Ctr, class _Iterator> _Ctr _Tree_node::_C_children [inherited]

the edges to the children

Definition at line 76 of file vgtl_tree.h.

7.14.4.2 template<class _Tp, class _Ctr, class _Iterator> _Tp _Tree_node::_C_data [inherited]

the node data

Definition at line 72 of file vgtl_tree.h.

7.14.4.3 template<class _Tp, class _Ctr, class _Iterator> ctree_data_hook _ITree_node::_C_data_hook

the data hook for trees with data hook

Definition at line 147 of file vgtl_tree.h.

7.14.4.4 template<class _Tp, class _Ctr, class _Iterator> _Void_pointer _Tree_node::_C_parent [inherited]

the edge to the parent

Definition at line 74 of file vgtl_tree.h.

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

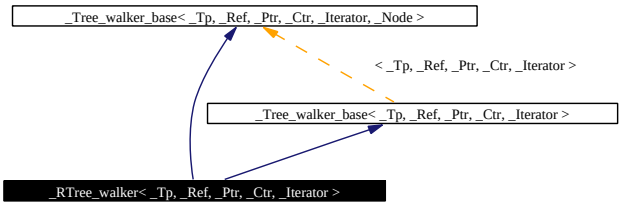
- **vgtl_tree.h**

7.15 _RTree_walker Class Template Reference

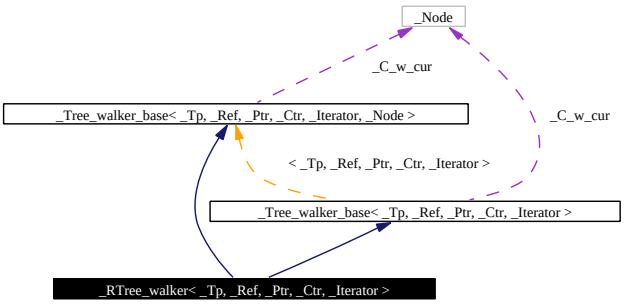
recursive tree walkers.

`#include <vgtl_tree.h>`

Inheritance diagram for _RTree_walker:



Collaboration diagram for _RTree_walker:



Public Types

- **typedef _Tp value_type**
- **typedef _Ptr pointer**
- **typedef _Ref reference**

- **typedef __one_iterator< void * > parents_iterator**
- **typedef _Ctr_iterator children_iterator**
- **typedef _Node node_type**
- **typedef size_t size_type**
- **typedef ptrdiff_t difference_type**

Public Methods

- **_RTree_walker ()**
- **_RTree_walker (_Node *__x)**
- **_RTree_walker (const walker &__x)**
- **_Self operator<< (const parents_iterator &__dummy)**

  go to parent operator.

- **_Self operator>> (const children_iterator &__i)**

go to child operator.

- _Self & operator<<= (const parents_iterator &__dummy)
- _Self & operator>>= (const children_iterator &__i)
- _Self & operator= (const _Itr &__x)
- _Self & operator= (const _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node > &__x)
- reference operator * () const
- pointer operator → () const
- ctree_data_hook & data_hook ()
- ctree_data_hook & parent_data_hook ()
- const _Node * parent ()
- const _Node * node ()
- size_type n_children ()
- size_type n_parents ()
- bool is_leaf ()
- bool is_root ()
- bool is_ground ()
- bool is_sky ()
- children_iterator child_begin ()
- children_iterator child_end ()
- parents_iterator parent_begin ()
- parents_iterator parent_end ()
- template<class _Function> _Function for_each_child (_Function __f)
- template<class _Function> _Function for_each_parent (_Function __f)
- template<class Compare> void sort_children (children_iterator first, children_iterator last, Compare comp)
- template<class Compare> void sort_children (Compare comp)
- template<class Compare> void sort_parents (parents_iterator first, parents_iterator last, Compare comp)
- template<class Compare> void sort_parents (Compare comp)

- bool operator== (const _Self &__x) const
- bool operator!= (const _Self &__x) const

Public Attributes

- _Node * _C_w_cur

7.15.1 Detailed Description

template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> class _RTree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >

This is the class defining reursive tree walkers, which walk trees under guidance.

Definition at line 837 of file vgtl_graph.h.

7.15.2 Member Typedef Documentation

7.15.2.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Ctr_iterator _Tree_walker_base::children_iterator [inherited]

standard walker definition

Definition at line 242 of file vgtl_tree.h.

7.15.2.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef ptrdiff_t _Tree_walker_base::difference_type [inherited]

standard walker definition

Definition at line 246 of file vgtl_tree.h.

7.15.2.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Node _Tree_walker_base::node_type [inherited]

standard walker definition

Definition at line 243 of file vgtl_tree.h.

7.15.2.4 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef __one_iterator<void *> _Tree_walker_base::parents_iterator [inherited]

standard walker definition

Definition at line 241 of file vgtl_tree.h.

7.15.2.5 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Ptr _Tree_walker_base::pointer [inherited]

standard walker definition

Definition at line 232 of file vgtl_tree.h.

7.15.2.6 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Ref _Tree_walker_base::reference [inherited]

standard walker definition

Definition at line 233 of file vgtl_tree.h.

7.15.2.7 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef size_t _Tree_walker_base::size_type [inherited]

standard walker definition

Definition at line 245 of file vgtl_tree.h.

7.15.2.8 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Tp _Tree_walker_base::value_type [inherited]

standard walker definition

Definition at line 231 of file vgtl_tree.h.

7.15.3 Constructor & Destructor Documentation

7.15.3.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _RTree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::_RTree_walker () [inline]

standard constructor

Definition at line 1069 of file vgtl_tree.h.

7.15.3.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _RTree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::_RTree_walker (_Node * __x) [inline]

constructor setting the position

Definition at line 1072 of file vgtl_tree.h.

7.15.3.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _RTree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::_RTree_walker (const walker & __x) [inline]

copy constructor

Definition at line 1075 of file vgtl_tree.h.

7.15.4 Member Function Documentation

7.15.4.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> children_iterator _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::child_begin () [inline, inherited]

return children_iterator to first child

Definition at line 306 of file vgtl_tree.h.

7.15.4.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> children_iterator _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::child_end () [inline, inherited]

return children_iterator beyond last child

Definition at line 308 of file vgtl_tree.h.

7.15.4.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> ctree_data_hook& _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::data_hook () [inline, inherited]

retrieve the data hook

Definition at line 279 of file vgtl_tree.h.

7.15.4.4 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class _Function> _Function _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::for_each_child (_Function __f) [inline, inherited]

apply the function __f to all children

Definition at line 319 of file vgtl_tree.h.

7.15.4.5 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class _Function> _Function _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::for_each_parent (_Function __f) `[inline, inherited]`

apply the function __f to all parents

Definition at line 325 of file vgtl_tree.h.

7.15.4.6 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::is_ground () `[inline, inherited]`

is this node a virtual node - the ground (below all roots)?

Definition at line 301 of file vgtl_tree.h.

7.15.4.7 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::is_leaf () `[inline, inherited]`

is this node a leaf?

Definition at line 295 of file vgtl_tree.h.

7.15.4.8 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::is_root () `[inline, inherited]`

is this node a root?

Definition at line 297 of file vgtl_tree.h.

7.15.4.9 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::is_sky () `[inline, inherited]`

is this node a virtual node - the sky (above all leafs)?

Definition at line 303 of file vgtl_tree.h.

7.15.4.10 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> size_type _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::n_children () `[inline, inherited]`

return the number of children

Definition at line 290 of file vgtl_tree.h.

7.15.4.11 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> size_type _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::n_parents () `[inline, inherited]`

return the number of parents (0 or 1)

Definition at line 292 of file vgtl_tree.h.

7.15.4.12 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> const _Node* _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::node () [inline, inherited]

retrieve the full node

Definition at line 287 of file vgtl_tree.h.

7.15.4.13 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> reference _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator * () const [inline, inherited]

dereference operator

Definition at line 264 of file vgtl_tree.h.

7.15.4.14 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> bool _RTree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator!= (const _Self & __x) const [inline]

comparison operator

Definition at line 1082 of file vgtl_tree.h.

7.15.4.15 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> pointer _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator → () const [inline, inherited]

pointer operator

Definition at line 268 of file vgtl_tree.h.

7.15.4.16 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self _RTree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator<< (const parents_iterator & __dummy) [inline]

This operator moves the walker to the parent

Definition at line 1088 of file vgtl_tree.h.

7.15.4.17 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _RTree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator<<= (const parents_iterator & __dummy) [inline]

go to parent assignment operator

Definition at line 1105 of file vgtl_tree.h.

7.15.4.18 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _RTree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator= (const _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node > & __x) [inline]

assignment from automatic iterator

Definition at line 1125 of file vgtl_tree.h.

7.15.4.19 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _RTree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator= (const _Itr & __x) `[inline]`

assignment from iterator

Reimplemented from _Tree_walker_base.

Definition at line 1119 of file vgtl_tree.h.

7.15.4.20 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> bool _RTree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator== (const _Self & __x) const `[inline]`

comparison operator

Definition at line 1080 of file vgtl_tree.h.

7.15.4.21 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self _RTree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator>> (const children_iterator & __i) `[inline]`

This operator moves the walker to the child pointed to by `__i`

Definition at line 1098 of file vgtl_tree.h.

7.15.4.22 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _RTree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator>>= (const children_iterator & __i) `[inline]`

go to child assignment operator

Definition at line 1113 of file vgtl_tree.h.

7.15.4.23 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> const _Node* _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::parent () `[inline, inherited]`

retrieve the parent node

Definition at line 285 of file vgtl_tree.h.

7.15.4.24 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> parents_iterator _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::parent_begin () `[inline, inherited]`

return parents_iterator to first parent (the parent)

Definition at line 311 of file vgtl_tree.h.

7.15.4.25 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> ctree_data_hook& _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::parent_data_hook () `[inline, inherited]`

retrieve the parent's data hook

Definition at line 281 of file vgtl_tree.h.

7.15.4.26 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> parents_iterator _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::parent_end () [inline, inherited]

return parents_iterator beyond last parent

Definition at line 314 of file vgtl_tree.h.

7.15.4.27 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class Compare> void _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::sort_children (Compare *comp*) [inline, inherited]

sort all children according to `comp`

Definition at line 343 of file vgtl_tree.h.

7.15.4.28 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class Compare> void _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::sort_children (children_iterator *first*, children_iterator *last*, Compare *comp*) [inline, inherited]

sort the children in the range [first,last) according to `comp`

Definition at line 332 of file vgtl_tree.h.

7.15.4.29 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class Compare> void _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::sort_parents (Compare *comp*) [inline, inherited]

sort all parents according to `comp` **(NOP = do nothing)**

Definition at line 348 of file vgtl_tree.h.

7.15.4.30 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class Compare> void _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::sort_parents (parents_iterator *first*, parents_iterator *last*, Compare *comp*) [inline, inherited]

sort the parents in the range [first,last) according to `comp` **(NOP)**

Definition at line 338 of file vgtl_tree.h.

7.15.5 Member Data Documentation

7.15.5.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Node* _Tree_walker_base::_C_w_cur [inherited]

pointer to the current node

Definition at line 251 of file vgtl_tree.h.

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

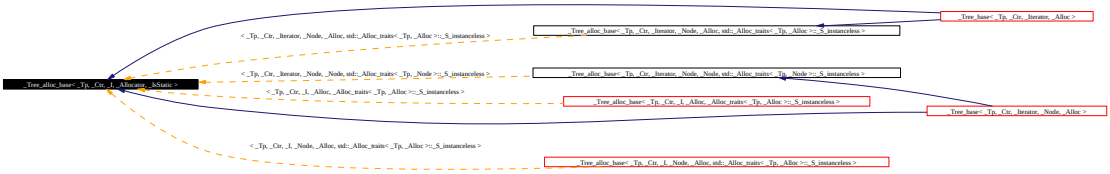
- **vgtl_graph.h**
- **vgtl_tree.h**

7.16 _Tree_alloc_base Class Template Reference

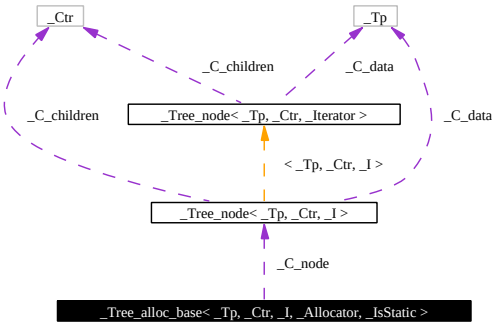
Tree base class for general standard-conforming allocators.

`#include <vgtl_tree.h>`

Inheritance diagram for _Tree_alloc_base:



Collaboration diagram for _Tree_alloc_base:



Protected Methods

- **_Node * _C_get_node ()**
- **void _C_put_node (_Node *__p)**

Protected Attributes

- **_Node * _C_node**

7.16.1 Detailed Description

template<class _Tp, class _Ctr, class _I, class _Allocator, bool _IsStatic> class _Tree_alloc_base< _Tp, _Ctr, _I, _Allocator, _IsStatic >

Base tree class that encapsulates details of allocators. Three cases: an ordinary standard-conforming allocator, a standard-conforming allocator with no non-static data, and an SGI-style allocator. This complexity is necessary only because we're worrying about STL compatibility and because we want to avoid wasting storage on an allocator instance if it isn't necessary. Base for general standard-conforming allocators.

Definition at line 1092 of file vgtl_graph.h.

7.16.2 Member Function Documentation

7.16.2.1 template<class _Tp, class _Ctr, class _I, class _Allocator, bool _IsStatic> _Node* _Tree_alloc_base< _Tp, _Ctr, _I, _Allocator, _IsStatic >::_C_get_node () [inline, protected]

allocate a new node

Definition at line 1374 of file vgtl_tree.h.

7.16.2.2 template<class _Tp, class _Ctr, class _I, class _Allocator, bool _IsStatic> void _Tree_alloc_base< _Tp, _Ctr, _I, _Allocator, _IsStatic >::_C_put_node (_Node * __p) [inline, protected]

deallocate a node

Definition at line 1377 of file vgtl_tree.h.

7.16.3 Member Data Documentation

7.16.3.1 template<class _Tp, class _Ctr, class _I, class _Allocator, bool _IsStatic> _Node* _Tree_alloc_base::_C_node [protected]

This is the node

Definition at line 1386 of file vgtl_tree.h.

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

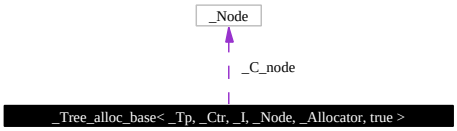
- vgtl_graph.h
- vgtl_tree.h

7.17 _Tree_alloc_base< _Tp, _Ctr, _I, _Node, _Allocator, true > Class Template Reference

Tree base class specialization for instanceless allocators.

`#include <vgtl_tree.h>`

Collaboration diagram for _Tree_alloc_base< _Tp, _Ctr, _I, _Node, _Allocator, true >:



Protected Methods

- _Node * _C_get_node ()
- void _C_put_node (_Node *__p)

Protected Attributes

- _Node * _C_node

7.17.1 Detailed Description

template<class _Tp, class _Ctr, class _I, class _Node, class _Allocator> class _Tree_alloc_base< _Tp, _Ctr, _I, _Node, _Allocator, true >

Base tree class that encapsulates details of allocators. Three cases: an ordinary standard-conforming allocator, a standard-conforming allocator with no non-static data, and an SGI-style allocator. This complexity is necessary only because we're worrying about STL compatibility and because we want to avoid wasting storage on an allocator instance if it isn't necessary. Base class specialization for instanceless allocators.

Definition at line 1401 of file vgtl_tree.h.

7.17.2 Member Function Documentation

7.17.2.1 template<class _Tp, class _Ctr, class _I, class _Node, class _Allocator> _Node* _Tree_alloc_base< _Tp, _Ctr, _I, _Node, _Allocator, true >::_C_get_node () [inline, protected]

allocate a new node

Definition at line 1413 of file vgtl_tree.h.

7.17.2.2 template<class _Tp, class _Ctr, class _I, class _Node, class _Allocator> void _Tree_alloc_base< _Tp, _Ctr, _I, _Node, _Allocator, true >::_C_put_node (_Node * __p) [inline, protected]

deallocate a node

Definition at line 1416 of file vgtl_tree.h.

7.17.3 Member Data Documentation

7.17.3.1 template<class _Tp, class _Ctr, class _I, class _Node, class _Allocator> _Node* _Tree_alloc_base< _Tp, _Ctr, _I, _Node, _Allocator, true >::_C_node [protected]

This is the root node

Definition at line 1421 of file vgtl_tree.h.

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

- **vgtl_tree.h**

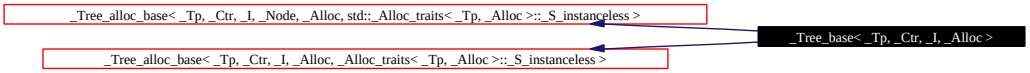
7.18 _Tree_base Class Template Reference

Tree base class for allocator encapsulation.

`#include <vgtl_tree.h>`

Inheritance diagram for _Tree_base:

Collaboration diagram for _Tree_base:



Public Types

- typedef _Base::allocator_type allocator_type
- typedef _Ctr container_type
- typedef _I children_iterator
- typedef __one_iterator< void * > parents_iterator

Public Methods

- _Tree_base (const allocator_type &__a)
- virtual ~_Tree_base ()
- void clear ()
- void clear_children ()
- template<class _Output_Iterator> void add_all_children (_Output_Iterator fi, _Node *_parent)

7.18.1 Detailed Description

template<class _Tp, class _Ctr, class _I, class _Alloc> class _Tree_base< _Tp, _Ctr, _I, _Alloc >

Base tree class top level that encapsulates details of allocators.

Definition at line 1138 of file vgtl_graph.h.

7.18.2 Member Typedef Documentation

7.18.2.1 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef _Base::allocator_type _Tree_base::allocator_type

allocator type

Reimplemented from _Tree_alloc_base< _Tp, _Ctr, _I, _Alloc, _Alloc_traits< _Tp, _Alloc >::_S_instanceless >.

Definition at line 1439 of file vgtl_tree.h.

7.18.2.2 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef _I _Tree_base::children_iterator

iterator for accessing the children

Definition at line 1444 of file vgtl_tree.h.

7.18.2.3 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef _Ctr _Tree_base::container_type

internal container used to store the children

Definition at line 1442 of file vgtl_tree.h.

7.18.2.4 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef __one_iterator<void *> _Tree_base::parents_iterator

iterator for accessing the parents

Definition at line 1446 of file vgtl_tree.h.

7.18.3 Constructor & Destructor Documentation

7.18.3.1 template<class _Tp, class _Ctr, class _I, class _Alloc> _Tree_base< _Tp, _Ctr, _I, _Alloc >::_Tree_base (const allocator_type & __a) [inline]

constructor initializing the allocator and the root

Definition at line 1449 of file vgtl_tree.h.

7.18.3.2 template<class _Tp, class _Ctr, class _I, class _Alloc> virtual _Tree_base< _Tp, _Ctr, _I, _Alloc >::~_Tree_base () [inline, virtual]

standard destructor

Definition at line 1457 of file vgtl_tree.h.

7.18.4 Member Function Documentation

7.18.4.1 template<class _Tp, class _Ctr, class _I, class _Alloc> template<class _Output_Iterator> void _Tree_base< _Tp, _Ctr, _I, _Alloc >::add_all_children (_Output_Iterator *fi*, _Node * *_parent*)

add all children to the parent `_parent`. `fi` is a iterator to the children container of the parent

7.18.4.2 template<class _Tp, class _Ctr, class _I, class _Alloc> void _Tree_base< _Tp, _Ctr, _I, _Alloc >::clear ()

empty the tree

7.18.4.3 template<class _Tp, class _Ctr, class _I, class _Alloc> void _Tree_base< _Tp, _Ctr, _I, _Alloc >::clear_children () [inline]

clear all children of the root node

Definition at line 1465 of file vgtl_tree.h.

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

- vgtl_graph.h
- vgtl_tree.h

7.19 _Tree_data_hook Union Reference

`#include <vgtl_gdata.h>`

7.19.1 Detailed Description

This is a mixed-type union for data hooks on trees. A data hook can be used for non-recursive walks.

Definition at line 39 of file vgtl_gdata.h.

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

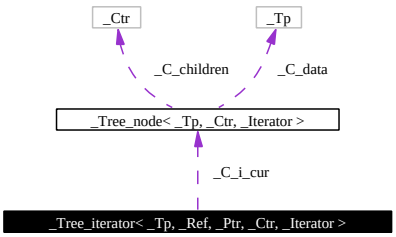
- **vgtl_gdata.h**

7.20 _Tree_iterator Class Template Reference

iterator through the tree.

`#include <vgtl_tree.h>`

Collaboration diagram for _Tree_iterator:



Public Types

- **typedef std::bidirectional_iterator_tag iterator_category**
- **typedef _Tp value_type**
- **typedef _Ptr pointer**
- **typedef _Ref reference**
- **typedef size_t size_type**
- **typedef ptrdiff_t difference_type**

Public Methods

- **_Tree_iterator ()**
- **_Tree_iterator (const iterator &__x)**
- **_Tree_iterator (const _Node *__n, bool st=false)**
- **reference operator * () const**
- **pointer operator → () const**
- **ctree_data_hook & data_hook ()**
- **_Self & operator= (const _Walk &__x)**

- **bool operator== (const _Self &__x) const**

- bool operator!= (const _Self &__x) const

- _Self & operator++ ()
- _Self operator++ (int)
- _Self & operator– ()
- _Self operator– (int)

Protected Attributes

- _Node * _C_i_cur
- std::vector< _Ctr_iterator > _C_i_cur_it

7.20.1 Detailed Description

template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> class _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >

This is an iterator, which visits each node of a tree once. It is based on a preorder depth-first automatic walker.

Definition at line 896 of file vgtl_graph.h.

7.20.2 Member Typedef Documentation

7.20.2.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef ptrdiff_t _Tree_iterator::difference_type

standard iterator definition

Definition at line 1155 of file vgtl_tree.h.

7.20.2.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef std::bidirectional_iterator_tag _Tree_iterator::iterator_category

standard iterator definition

Definition at line 1150 of file vgtl_tree.h.

7.20.2.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef _Ptr _Tree_iterator::pointer

standard iterator definition

Definition at line 1152 of file vgtl_tree.h.

7.20.2.4 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef _Ref _Tree_iterator::reference

standard iterator definition

Definition at line 1153 of file vgtl_tree.h.

7.20.2.5 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef size_t _Tree_iterator::size_type

standard iterator definition

Definition at line 1154 of file vgtl_tree.h.

7.20.2.6 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> typedef _Tp _Tree_iterator::value_type

standard iterator definition

Definition at line 1151 of file vgtl_tree.h.

7.20.3 Constructor & Destructor Documentation

7.20.3.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::_Tree_iterator () [inline]

standard constructor

Definition at line 1167 of file vgtl_tree.h.

7.20.3.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::_Tree_iterator (const iterator & __x) [inline]

copy constructor

Definition at line 1169 of file vgtl_tree.h.

7.20.3.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::_Tree_iterator (const _Node * __n, bool st = false) [inline]

constructor setting a specific position

Definition at line 1172 of file vgtl_tree.h.

7.20.4 Member Function Documentation

7.20.4.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> ctree_data_hook& _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::data_hook () [inline]

access to the data hook of the node

Definition at line 1198 of file vgtl_tree.h.

7.20.4.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> reference _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator * () const [inline]

dereference operator

Definition at line 1191 of file vgtl_tree.h.

7.20.4.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> bool _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator!= (const _Self & __x) const [inline]

comparison operator

Definition at line 1183 of file vgtl_tree.h.

7.20.4.4 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator++ (int) [inline]

in(de)crement operator

Definition at line 1225 of file vgtl_tree.h.

7.20.4.5 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator++ () [inline]

in(de)crement operator

Definition at line 1221 of file vgtl_tree.h.

7.20.4.6 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator– (int) [inline]

in(de)crement operator

Definition at line 1235 of file vgtl_tree.h.

7.20.4.7 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator– () [inline]

in(de)crement operator

Definition at line 1231 of file vgtl_tree.h.

7.20.4.8 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> pointer _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator → () const [inline]

pointer operator

Definition at line 1195 of file vgtl_tree.h.

7.20.4.9 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Self& _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator= (const _Walk & __*x*) [inline]

assignment to iterator from walker

Definition at line 1210 of file vgtl_tree.h.

7.20.4.10 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> bool _Tree_iterator< _Tp, _Ref, _Ptr, _Ctr, _Iterator >::operator== (const _Self & __*x*) const [inline]

comparison operator

Definition at line 1177 of file vgtl_tree.h.

7.20.5 Member Data Documentation

7.20.5.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> _Node* _Tree_iterator::_C_i_cur [protected]

current position

Definition at line 1161 of file vgtl_tree.h.

7.20.5.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator> std::vector<_Ctr_iterator> _Tree_iterator::_C_i_cur_it [protected]

internal stack

Definition at line 1163 of file vgtl_tree.h.

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

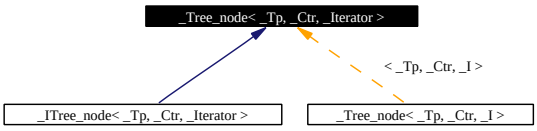
- vgtl_graph.h
- vgtl_tree.h

7.21 _Tree_node Class Template Reference

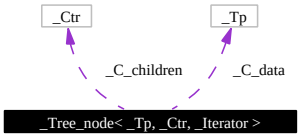
tree node for trees w/o data hooks.

`#include <vgtl_tree.h>`

Inheritance diagram for _Tree_node:



Collaboration diagram for _Tree_node:



Public Methods

- _Tree_node ()
- void initialize ()
- void get_rid_of ()
- void clear_tree ()
- void clear_children ()
- _Ctr_iterator get_childentry_iterator (_Void_pointer __p)
- template<class _Output_Iterator> void add_all_children (_Output_Iterator fi, _Self *_parent)

- template<class Compare> void sort_children (_Ctr_iterator first, _Ctr_iterator last, Compare comp)
- template<class Compare> void sort_parents (_Ctr_iterator first, _Ctr_iterator last, Compare comp)

Public Attributes

- _Tp _C_data
- _Void_pointer _C_parent
- _Ctr _C_children

7.21.1 Detailed Description

template<class _Tp, class _Ctr, class _Iterator> class _Tree_node< _Tp, _Ctr, _Iterator >

This is the tree node for a tree without data hooks

Definition at line 63 of file vgtl_tree.h.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 template<class _Tp, class _Ctr, class _Iterator> _Tree_node< _Tp, _Ctr, _Iterator >::_Tree_node () [inline]

standard constructor

Definition at line 79 of file vgtl_tree.h.

7.21.3 Member Function Documentation

7.21.3.1 template<class _Tp, class _Ctr, class _Iterator> template<class _Output_Iterator> void _Tree_node< _Tp, _Ctr, _Iterator >::add_all_children (_Output_Iterator *fi*, _Self * *_parent*)

add all children to parent `_parent`. `fi` is an iterator to the children container of `_parent`

Definition at line 180 of file vgtl_tree.h.

7.21.3.2 template<class _Tp, class _Ctr, class _Iterator> void _Tree_node< _Tp, _Ctr, _Iterator >::clear_children () [inline]

erase all children entries

Definition at line 100 of file vgtl_tree.h.

7.21.3.3 template<class _Tp, class _Ctr, class _Iterator> void _Tree_node< _Tp, _Ctr, _Iterator >::clear_tree ()

remove the whole subtree below this node

Definition at line 195 of file vgtl_tree.h.

7.21.3.4 template<class _Tp, class _Ctr, class _Iterator> _Ctr_iterator _Tree_node< _Tp, _Ctr, _Iterator >::get_childentry_iterator (_Void_pointer __p) [inline]

find the iterator into the children container for child __p

Definition at line 104 of file vgtl_tree.h.

7.21.3.5 template<class _Tp, class _Ctr, class _Iterator> void _Tree_node< _Tp, _Ctr, _Iterator >::get_rid_of () [inline]

remove the children container

Reimplemented in _ITree_node.

Definition at line 93 of file vgtl_tree.h.

7.21.3.6 template<class _Tp, class _Ctr, class _Iterator> void _Tree_node< _Tp, _Ctr, _Iterator >::initialize () [inline]

initialize the data structure

Reimplemented in _ITree_node.

Definition at line 87 of file vgtl_tree.h.

7.21.3.7 template<class _Tp, class _Ctr, class _Iterator> template<class Compare> void _Tree_node< _Tp, _Ctr, _Iterator >::sort_children (_Ctr_iterator *first*, _Ctr_iterator *last*, Compare *comp*) [inline]

sort the children according to comp

Definition at line 121 of file vgtl_tree.h.

7.21.3.8 template<class _Tp, class _Ctr, class _Iterator> template<class Compare> void _Tree_node< _Tp, _Ctr, _Iterator >::sort_parents (_Ctr_iterator *first*, _Ctr_iterator *last*, Compare *comp*) [inline]

sort the children according to comp, i.e. do nothing here

Definition at line 128 of file vgtl_tree.h.

7.21.4 Member Data Documentation

7.21.4.1 template<class _Tp, class _Ctr, class _Iterator> _Ctr _Tree_node::_C_children

the edges to the children

Definition at line 76 of file vgtl_tree.h.

7.21.4.2 template<class _Tp, class _Ctr, class _Iterator> _Tp _Tree_node::_C_data

the node data

Definition at line 72 of file vgtl_tree.h.

7.21.4.3 template<class _Tp, class _Ctr, class _Iterator> _Void_pointer _Tree_node::_C_parent

the edge to the parent

Definition at line 74 of file vgtl_tree.h.

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

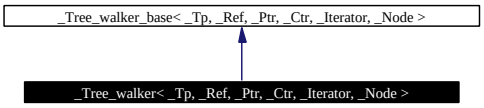
- **vgtl_tree.h**

7.22 _Tree_walker Class Template Reference

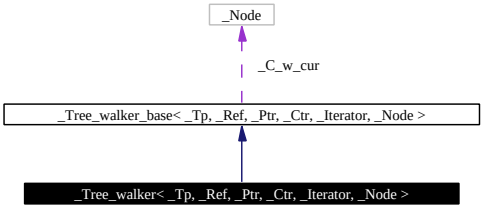
automatic tree walkers.

`#include <vgtl_tree.h>`

Inheritance diagram for _Tree_walker:



Collaboration diagram for _Tree_walker:



Public Types

- **typedef _Tp value_type**
- **typedef _Ptr pointer**
- **typedef _Ref reference**

- **typedef __one_iterator< void * > parents_iterator**
- **typedef _Ctr_iterator children_iterator**
- **typedef _Node node_type**
- **typedef size_t size_type**
- **typedef ptrdiff_t difference_type**

Public Methods

- **_Tree_walker ()**
- **_Tree_walker (_Node *__x, int order=(_C_W_preorder|_C_W_postorder), bool front_to_-back=true, bool depth_first=true, bool find_start=true)**
- **_Tree_walker (const walker &__x)**
- **_Self operator<< (const parents_iterator &__dummy)**

  go to parent operator.

- _Self operator>> (const children_iterator &__i)

  go to child operator.

- _Self & operator<<= (const parents_iterator &__dummy)
- _Self & operator>>= (const children_iterator &__i)
- _Self & operator~ ()
- _Self & operator= (const _Itr &__x)
- bool in_preorder ()
- reference operator * () const
- pointer operator → () const
- ctree_data_hook & data_hook ()
- ctree_data_hook & parent_data_hook ()
- const _Node * parent ()
- const _Node * node ()
- size_type n_children ()
- size_type n_parents ()
- bool is_leaf ()
- bool is_root ()
- bool is_ground ()
- bool is_sky ()
- children_iterator child_begin ()
- children_iterator child_end ()
- parents_iterator parent_begin ()
- parents_iterator parent_end ()
- template<class _Function> _Function for_each_child (_Function __f)
- template<class _Function> _Function for_each_parent (_Function __f)
- template<class Compare> void sort_children (children_iterator first, children_iterator last, Compare comp)
- template<class Compare> void sort_children (Compare comp)
- template<class Compare> void sort_parents (parents_iterator first, parents_iterator last, Compare comp)
- template<class Compare> void sort_parents (Compare comp)

- bool operator== (const _Self &__x) const
- bool operator!= (const _Self &__x) const

- _Self & operator++ ()
- _Self operator++ (int)
- _Self & operator– ()
- _Self operator– (int)

Public Attributes

- struct {
  } _C_w_t
- bool _C_w_in_preorder
- std::vector< _Iterator > _C_w_cur_it
- _Node * _C_w_cur

7.22.1 Detailed Description

template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> class _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >

This is the class defining automatic (iterative) tree walkers, which walk trees without guidance.

Definition at line 359 of file vgtl_tree.h.

7.22.2 Member Typedef Documentation

7.22.2.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Ctr_iterator _Tree_walker_base::children_iterator [inherited]

standard walker definition

Definition at line 242 of file vgtl_tree.h.

7.22.2.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef ptrdiff_t _Tree_walker_base::difference_type [inherited]

standard walker definition

Definition at line 246 of file vgtl_tree.h.

7.22.2.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Node _Tree_walker_base::node_type [inherited]

standard walker definition

Definition at line 243 of file vgtl_tree.h.

7.22.2.4 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef __one_iterator<void *> _Tree_walker_base::parents_iterator [inherited]

standard walker definition

Definition at line 241 of file vgtl_tree.h.

7.22.2.5 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Ptr _Tree_walker_base::pointer [inherited]

standard walker definition

Definition at line 232 of file vgtl_tree.h.

7.22.2.6 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Ref _Tree_walker_base::reference [inherited]

standard walker definition

Definition at line 233 of file vgtl_tree.h.

7.22.2.7 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef size_t _Tree_walker_base::size_type [inherited]

standard walker definition

Definition at line 245 of file vgtl_tree.h.

7.22.2.8 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Tp _Tree_walker_base::value_type [inherited]

standard walker definition

Definition at line 231 of file vgtl_tree.h.

7.22.3 Constructor & Destructor Documentation

7.22.3.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::_Tree_walker () [inline]

standard constructor

Definition at line 380 of file vgtl_tree.h.

7.22.3.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::_Tree_walker (_Node * __x, int *order* = (_C_W_preorder|_C_W_postorder), bool *front_to_back* = true, bool *depth_first* = true, bool *find_start* = true) [inline]

This is the main constructor for an automatic walker. It sets the starting position and, optionally, the walker type.

Definition at line 405 of file vgtl_tree.h.

7.22.3.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::_Tree_walker (const walker & __x) [inline]

copy constructor

Definition at line 422 of file vgtl_tree.h.

7.22.4 Member Function Documentation

7.22.4.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> children_iterator _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::child_begin () [inline, inherited]

return children_iterator to first child

Definition at line 306 of file vgtl_tree.h.

7.22.4.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> children_iterator _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::child_end () [inline, inherited]

return children_iterator beyond last child

Definition at line 308 of file vgtl_tree.h.

7.22.4.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> ctree_data_hook& _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::data_hook () [inline, inherited]

retrieve the data hook

Definition at line 279 of file vgtl_tree.h.

7.22.4.4 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class _Function> _Function _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::for_each_child (_Function __f) [inline, inherited]

apply the function __f to all children

Definition at line 319 of file vgtl_tree.h.

7.22.4.5 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class _Function> _Function _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::for_each_parent (_Function __f) [inline, inherited]

apply the function __f to all parents

Definition at line 325 of file vgtl_tree.h.

7.22.4.6 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::in_preorder () [inline]

are we in the preorder phase of a pre+post walk?

Definition at line 586 of file vgtl_tree.h.

7.22.4.7 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::is_ground () [inline, inherited]

is this node a virtual node - the ground (below all roots)?

Definition at line 301 of file vgtl_tree.h.

7.22.4.8 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::is_leaf () [inline, inherited]

is this node a leaf?

Definition at line 295 of file vgtl_tree.h.

7.22.4.9 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::is_root () [inline, inherited]

is this node a root?

Definition at line 297 of file vgtl_tree.h.

7.22.4.10 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::is_sky () [inline, inherited]

is this node a virtual node - the sky (above all leafs)?

Definition at line 303 of file vgtl_tree.h.

7.22.4.11 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> size_type _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::n_children () [inline, inherited]

return the number of children

Definition at line 290 of file vgtl_tree.h.

7.22.4.12 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> size_type _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::n_parents () [inline, inherited]

return the number of parents (0 or 1)

Definition at line 292 of file vgtl_tree.h.

7.22.4.13 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> const _Node* _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::node () [inline, inherited]

retrieve the full node

Definition at line 287 of file vgtl_tree.h.

7.22.4.14 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> reference _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator * () const [inline, inherited]

dereference operator

Definition at line 264 of file vgtl_tree.h.

7.22.4.15 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator!= (const _Self & __x) const [inline]

comparison operator

Definition at line 438 of file vgtl_tree.h.

7.22.4.16 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Self _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator++ (int) [inline]

in(de)crement operator

Definition at line 473 of file vgtl_tree.h.

7.22.4.17 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Self& _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator++ () `[inline]`

in(de)crement operator

Definition at line 451 of file vgtl_tree.h.

7.22.4.18 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Self _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator– (int) `[inline]`

in(de)crement operator

Definition at line 501 of file vgtl_tree.h.

7.22.4.19 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Self& _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator– () `[inline]`

in(de)crement operator

Definition at line 479 of file vgtl_tree.h.

7.22.4.20 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> pointer _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator → () const `[inline, inherited]`

pointer operator

Definition at line 268 of file vgtl_tree.h.

7.22.4.21 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Self _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator<< (const parents_iterator & __dummy) `[inline]`

This operator moves the walker to the parent

Definition at line 510 of file vgtl_tree.h.

7.22.4.22 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Self& _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator<<= (const parents_iterator & __dummy) `[inline]`

go to parent assignment operator

Definition at line 541 of file vgtl_tree.h.

7.22.4.23 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Self& _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator= (const _Itr & __x) `[inline]`

assignment from iterator

Reimplemented from _Tree_walker_base.

Definition at line 576 of file vgtl_tree.h.

7.22.4.24 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator== (const _Self & __x) const `[inline]`

comparison operator

Definition at line 430 of file vgtl_tree.h.

7.22.4.25 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Self _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator>> (const children_iterator & __i) `[inline]`

This operator moves the walker to the child pointed to by __i

Definition at line 530 of file vgtl_tree.h.

7.22.4.26 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Self& _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator>>= (const children_iterator & __i) `[inline]`

go to child assignment operator

Definition at line 559 of file vgtl_tree.h.

7.22.4.27 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Self& _Tree_walker< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator~ () `[inline]`

switch from preorder to postorder phase

Definition at line 569 of file vgtl_tree.h.

7.22.4.28 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> const _Node* _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::parent () `[inline, inherited]`

retrieve the parent node

Definition at line 285 of file vgtl_tree.h.

7.22.4.29 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> parents_iterator _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::parent_begin () `[inline, inherited]`

return parents_iterator to first parent (the parent)

Definition at line 311 of file vgtl_tree.h.

7.22.4.30 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> ctree_data_hook& _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::parent_data_hook () `[inline, inherited]`

retrieve the parent's data hook

Definition at line 281 of file vgtl_tree.h.

7.22.4.31 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> parents_iterator _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::parent_end () `[inline, inherited]`

return parents_iterator beyond last parent

Definition at line 314 of file vgtl_tree.h.

7.22.4.32 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class Compare> void _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::sort_children (Compare *comp*) `[inline, inherited]`

sort all children according to `comp`

Definition at line 343 of file vgtl_tree.h.

7.22.4.33 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class Compare> void _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::sort_children (children_iterator *first*, children_iterator *last*, Compare *comp*) `[inline, inherited]`

sort the children in the range [first,last) according to `comp`

Definition at line 332 of file vgtl_tree.h.

7.22.4.34 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class Compare> void _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::sort_parents (Compare *comp*) `[inline, inherited]`

sort all parents according to `comp` **(NOP = do nothing)**

Definition at line 348 of file vgtl_tree.h.

7.22.4.35 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class Compare> void _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::sort_parents (parents_iterator *first*, parents_iterator *last*, Compare *comp*) `[inline, inherited]`

sort the parents in the range [first,last) according to `comp` **(NOP)**

Definition at line 338 of file vgtl_tree.h.

7.22.5 Member Data Documentation

7.22.5.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Node* _Tree_walker_base::_C_w_cur `[inherited]`

pointer to the current node

Definition at line 251 of file vgtl_tree.h.

7.22.5.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> std::vector<_Iterator> _Tree_walker::_C_w_cur_it

internal stack

Definition at line 376 of file vgtl_tree.h.

7.22.5.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker::_C_w_in_preorder

walker is in preorder mode?

Definition at line 374 of file vgtl_tree.h.

7.22.5.4 struct { ... } _Tree_walker::_C_w_t

walker type (order, front to back/back to front, depth/breath first

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

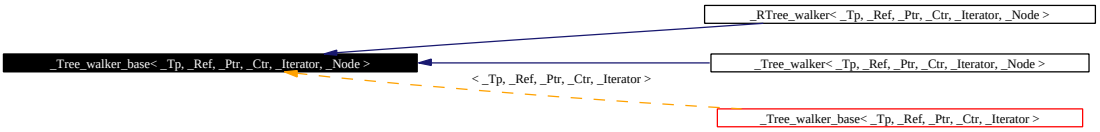
- **vgtl_tree.h**
- **vgtl_graph.h**

7.23 _Tree_walker_base Class Template Reference

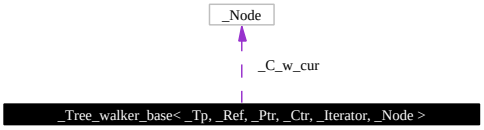
base class for all tree walkers.

`#include <vgtl_tree.h>`

Inheritance diagram for _Tree_walker_base:



Collaboration diagram for _Tree_walker_base:



Public Types

- **typedef _Tp value_type**
- **typedef _Ptr pointer**
- **typedef _Ref reference**

- **typedef __one_iterator< void * > parents_iterator**
- **typedef _Ctr_iterator children_iterator**
- **typedef _Node node_type**
- **typedef size_t size_type**
- **typedef ptrdiff_t difference_type**

Public Methods

- **_Tree_walker_base ()**
- **_Tree_walker_base (_Node *__x)**
- **_Tree_walker_base (const walker &__x)**

- reference operator * () const
- pointer operator → () const
- _Self & operator= (const _Itr &__x)
- ctree_data_hook & data_hook ()
- ctree_data_hook & parent_data_hook ()
- const _Node * parent ()
- const _Node * node ()
- size_type n_children ()
- size_type n_parents ()
- bool is_leaf ()
- bool is_root ()
- bool is_ground ()
- bool is_sky ()
- children_iterator child_begin ()
- children_iterator child_end ()
- parents_iterator parent_begin ()
- parents_iterator parent_end ()
- template<class _Function> _Function for_each_child (_Function __f)
- template<class _Function> _Function for_each_parent (_Function __f)
- template<class Compare> void sort_children (children_iterator first, children_iterator last, Compare comp)
- template<class Compare> void sort_parents (parents_iterator first, parents_iterator last, Compare comp)
- template<class Compare> void sort_children (Compare comp)
- template<class Compare> void sort_parents (Compare comp)

Public Attributes

- _Node * _C_w_cur

7.23.1 Detailed Description

template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> class _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >

This is the base class for all tree walkers.

Definition at line 221 of file vgtl_tree.h.

7.23.2 Member Typedef Documentation

7.23.2.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Ctr_iterator _Tree_walker_base::children_iterator

standard walker definition

Definition at line 242 of file vgtl_tree.h.

7.23.2.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef ptrdiff_t _Tree_walker_base::difference_type

standard walker definition

Definition at line 246 of file vgtl_tree.h.

7.23.2.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Node _Tree_walker_base::node_type

standard walker definition

Definition at line 243 of file vgtl_tree.h.

7.23.2.4 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef __one_iterator<void *> _Tree_walker_base::parents_iterator

standard walker definition

Definition at line 241 of file vgtl_tree.h.

7.23.2.5 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Ptr _Tree_walker_base::pointer

standard walker definition

Definition at line 232 of file vgtl_tree.h.

7.23.2.6 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Ref _Tree_walker_base::reference

standard walker definition

Definition at line 233 of file vgtl_tree.h.

7.23.2.7 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef size_t _Tree_walker_base::size_type

standard walker definition

Definition at line 245 of file vgtl_tree.h.

7.23.2.8 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> typedef _Tp _Tree_walker_base::value_type

standard walker definition

Definition at line 231 of file vgtl_tree.h.

7.23.3 Constructor & Destructor Documentation

7.23.3.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::_Tree_walker_base () [inline]

standard constructor

Definition at line 255 of file vgtl_tree.h.

7.23.3.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::_Tree_walker_base (_Node * __x) [inline]

constructor setting the position

Definition at line 258 of file vgtl_tree.h.

7.23.3.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::_Tree_walker_base (const walker & __x) [inline]

copy constructor

Definition at line 261 of file vgtl_tree.h.

7.23.4 Member Function Documentation

7.23.4.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> children_iterator _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::child_begin () [inline]

return children_iterator to first child

Definition at line 306 of file vgtl_tree.h.

7.23.4.2 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> children_iterator _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::child_end () [inline]

return children_iterator beyond last child

Definition at line 308 of file vgtl_tree.h.

7.23.4.3 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> ctree_data_hook& _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::data_hook () [inline]

retrieve the data hook

Definition at line 279 of file vgtl_tree.h.

7.23.4.4 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class _Function> _Function _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::for_each_child (_Function __f) [inline]

apply the function __f to all children

Definition at line 319 of file vgtl_tree.h.

7.23.4.5 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class _Function> _Function _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::for_each_parent (_Function __f) [inline]

apply the function __f to all parents

Definition at line 325 of file vgtl_tree.h.

7.23.4.6 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::is_ground () [inline]

is this node a virtual node - the ground (below all roots)?

Definition at line 301 of file vgtl_tree.h.

7.23.4.7 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::is_leaf () [inline]

is this node a leaf?

Definition at line 295 of file vgtl_tree.h.

7.23.4.8 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::is_root () [inline]

is this node a root?

Definition at line 297 of file vgtl_tree.h.

7.23.4.9 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> bool _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::is_sky () [inline]

is this node a virtual node - the sky (above all leafs)?

Definition at line 303 of file vgtl_tree.h.

7.23.4.10 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> size_type _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::n_children () [inline]

return the number of children

Definition at line 290 of file vgtl_tree.h.

7.23.4.11 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> size_type _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::n_parents () [inline]

return the number of parents (0 or 1)

Definition at line 292 of file vgtl_tree.h.

7.23.4.12 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> const _Node* _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::node () [inline]

retrieve the full node

Definition at line 287 of file vgtl_tree.h.

7.23.4.13 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> reference _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator * () const [inline]

dereference operator

Definition at line 264 of file vgtl_tree.h.

7.23.4.14 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> pointer _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator → () const `[inline]`

pointer operator

Definition at line 268 of file vgtl_tree.h.

7.23.4.15 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Self& _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::operator= (const _Itr & __x) `[inline]`

assignment operator from iterator to walker

Reimplemented in _Tree_walker.

Definition at line 273 of file vgtl_tree.h.

7.23.4.16 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> const _Node* _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::parent () `[inline]`

retrieve the parent node

Definition at line 285 of file vgtl_tree.h.

7.23.4.17 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> parents_iterator _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::parent_begin () `[inline]`

return parents_iterator to first parent (the parent)

Definition at line 311 of file vgtl_tree.h.

7.23.4.18 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> ctree_data_hook& _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::parent_data_hook () `[inline]`

retrieve the parent's data hook

Definition at line 281 of file vgtl_tree.h.

7.23.4.19 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> parents_iterator _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::parent_end () `[inline]`

return parents_iterator beyond last parent

Definition at line 314 of file vgtl_tree.h.

7.23.4.20 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class Compare> void _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::sort_children (Compare *comp*) `[inline]`

sort all children according to `comp`

Definition at line 343 of file vgtl_tree.h.

7.23.4.21 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class Compare> void _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::sort_children (children_iterator *first*, children_iterator *last*, Compare *comp*) `[inline]`

sort the children in the range [first,last) according to `comp`

Definition at line 332 of file vgtl_tree.h.

7.23.4.22 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class Compare> void _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::sort_parents (Compare *comp*) `[inline]`

sort all parents according to `comp` **(NOP = do nothing)**

Definition at line 348 of file vgtl_tree.h.

7.23.4.23 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> template<class Compare> void _Tree_walker_base< _Tp, _Ref, _Ptr, _Ctr, _Iterator, _Node >::sort_parents (parents_iterator *first*, parents_iterator *last*, Compare *comp*) `[inline]`

sort the parents in the range [first,last) according to `comp` **(NOP)**

Definition at line 338 of file vgtl_tree.h.

7.23.5 Member Data Documentation

7.23.5.1 template<class _Tp, class _Ref, class _Ptr, class _Ctr, class _Iterator, class _Node> _Node* _Tree_walker_base::_C_w_cur

pointer to the current node

Definition at line 251 of file vgtl_tree.h.

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

- **vgtl_tree.h**

7.24 array_vector Class Template Reference

STL vector wrapper for `C` **array.**

`#include <array_vector.h>`

Public Methods

- **array_vector ()**
- **array_vector (_T *__a, int n)**
- **~array_vector ()**
- **void assignvector (_T *__a, int n)**

7.24.1 Detailed Description

template<class _T> class array_vector< _T >

This class is a wrapper class, which builds a STL vector around a `C` array. Afterwards, this array_-vector can be used lika a const std::vector of the same type.

Definition at line 37 of file array_vector.h.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 template<class _T> array_vector< _T >::array_vector () `[inline]`

standard constructor

Definition at line 43 of file array_vector.h.

7.24.2.2 template<class _T> array_vector< _T >::array_vector (_T * __a, int n) `[inline]`

constructor building an array_vector from pointer `__a` with size `n`

Definition at line 48 of file array_vector.h.

7.24.2.3 template<class _T> array_vector< _T >::~array_vector () `[inline]`

standard destructor

Definition at line 57 of file array_vector.h.

7.24.3 Member Function Documentation

7.24.3.1 template<class _T> void array_vector< _T >::assignvector (_T * __a, int n) `[inline]`

assign an array (`__a`) of length `n` to this `array_vector`.

Definition at line 62 of file array_vector.h.

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

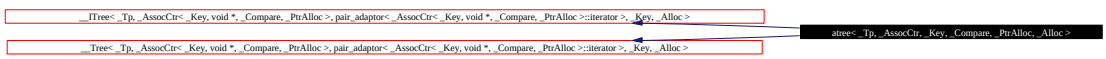
- **array_vector.h**

7.25 atree Class Template Reference

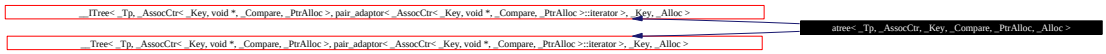
n-ary forest with labelled edges.

`#include <vgtl_tree.h>`

Inheritance diagram for atree:



Collaboration diagram for atree:



Public Types

- typedef _I children_iterator
- typedef __one_iterator< void * > parents_iterator

Public Methods

- _Self & operator= (_Node *__x)
- void insert (const __walker_base &__position, const _Tp &__x, const _Key &__k)
- void insert (const __walker_base &__position, const _Key &__k)
- void clear_children ()
- template<class _Output_Iterator> void add_all_children (_Output_Iterator fi, _Node *_parent)

7.25.1 Detailed Description

template<class _Tp, template< class __Key, class __Ty, class __Compare, class __AllocT > class _AssocCtr = multimap, class _Key = string, class _Compare = less<_Key>, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> class atree< _Tp, _AssocCtr, _Key, _Compare, _PtrAlloc, _Alloc >

This class constructs an n-ary forest with data hooks and labelled edges. By default, the children are collected in a STL multimap, but the container can be replaced by any other associative map container.

Definition at line 1769 of file vgtl_graph.h.

7.25.2 Member Typedef Documentation

7.25.2.1 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef _I _Tree_base::children_iterator [inherited]

iterator for accessing the children

Definition at line 1444 of file vgtl_tree.h.

7.25.2.2 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef __one_iterator<void *> _Tree_base::parents_iterator [inherited]

iterator for accessing the parents

Definition at line 1446 of file vgtl_tree.h.

7.25.3 Member Function Documentation

7.25.3.1 template<class _Tp, class _Ctr, class _I, class _Alloc> template<class _Output_Iterator> void _Tree_base< _Tp, _Ctr, _I, _Alloc >::add_all_children (_Output_Iterator *fi*, _Node * *_parent*) [inherited]

add all children to the parent `_parent`. `fi` is a iterator to the children container of the parent

7.25.3.2 template<class _Tp, class _Ctr, class _I, class _Alloc> void _Tree_base< _Tp, _Ctr, _I, _Alloc >::clear_children () [inline, inherited]

clear all children of the root node

Definition at line 1465 of file vgtl_tree.h.

7.25.3.3 template<class _Tp, template< class __Key, class __Ty, class __Compare, class __AllocT > class _AssocCtr = multimap, class _Key = string, class _Compare = less<_Key>, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void atree< _Tp, _AssocCtr, _Key, _Compare, _PtrAlloc, _Alloc >::insert (const __walker_base & *__position*, const _Key & *__k*) [inline]

Insert a node with default data and key __k at position __position.

Definition at line 2747 of file vgtl_tree.h.

7.25.3.4 template<class _Tp, template< class __Key, class __Ty, class __Compare, class __AllocT > class _AssocCtr = multimap, class _Key = string, class _Compare = less<_Key>, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void atree< _Tp, _AssocCtr, _Key, _Compare, _PtrAlloc, _Alloc >::insert (const __walker_base & *__position*, const _Tp & *__x*, const _Key & *__k*) [inline]

Insert a node with data __x and key _k at position __position.

Definition at line 2721 of file vgtl_tree.h.

7.25.3.5 template<class _Tp, template< class __Key, class __Ty, class __Compare, class __AllocT > class _AssocCtr = multimap, class _Key = string, class _Compare = less<_Key>, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> _Self& atree< _Tp, _AssocCtr, _Key, _Compare, _PtrAlloc, _Alloc >::operator= (_Node * *__x*) [inline]

assign a tree from one node -> make this node the root node. This is useful for making trees out of erased subtrees.

Reimplemented from __Tree< _Tp, _AssocCtr< _Key, void *, _Compare, _PtrAlloc >, pair_adaptor< _AssocCtr< _Key, void *, _Compare, _PtrAlloc >::iterator >, _Key, _Alloc >.

Definition at line 2712 of file vgtl_tree.h.

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

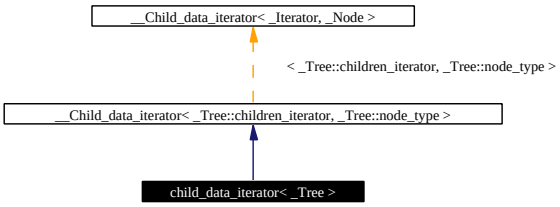
- vgtl_graph.h
- vgtl_tree.h

7.26 child_data_iterator Class Template Reference

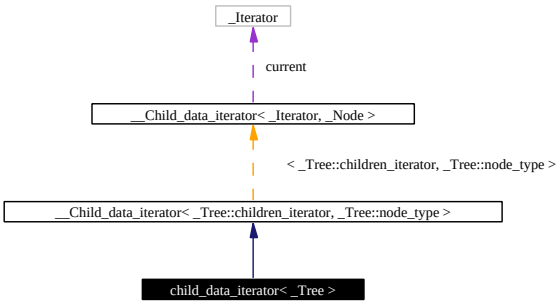
Iterator which iterates through the data hooks of all children.

```
#include <vgtl_algo.h>
```

Inheritance diagram for child_data_iterator:



Collaboration diagram for child_data_iterator:



Public Methods

- **child_data_iterator ()**

  standard constructor.

- **child_data_iterator (iterator_type __x)**

  constructor presetting the position.

- **child_data_iterator (const _Self &__x)**

  copy constructor.

- **_Self & operator= (const iterator_type &it)**

  assignment operator for setting the position.

7.26.1 Detailed Description

template<class _Tree> class child_data_iterator< _Tree >

This class defines an iterator for iterating through all data hooks of a node's children.

Definition at line 155 of file vgtl_algo.h.

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

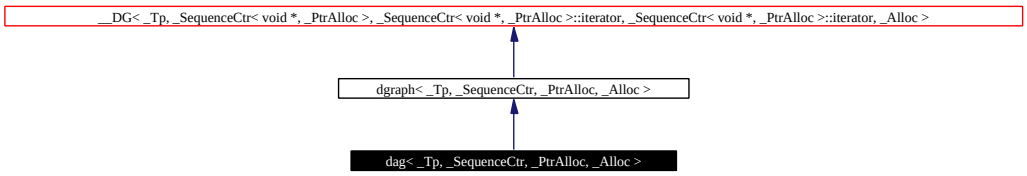
- **vgtl_algo.h**

7.27 dag Class Template Reference

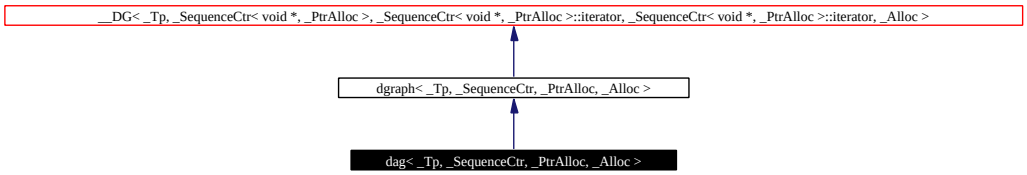
unlabeled directed acyclic graph (DAG).

`#include <vgtl_dag.h>`

Inheritance diagram for dag:



Collaboration diagram for dag:



Public Types

- **typedef _Base::walker walker**
- **typedef _Base::const_walker const_walker**
- **typedef _Base::children_iterator children_iterator**
- **typedef _Base::parents_iterator parents_iterator**
- **typedef _Base::erased_part erased_part**

Public Methods

- **dag (const allocator_type &__a=allocator_type())**
- **dag (const _Self &__dag)**
- **dag (const _Base &__dag)**
- **dag (const erased_part &__ep)**
- **bool check_acyclicity (const walker &__parent, const walker &__child)**
- **_Self & operator= (const _RV_DG &__rl)**
- **_Self & operator= (const erased_part &__ep)**
- **void clear ()**
- **walker between (const walker &__parent, const children_iterator &__cit, const walker &__child, const parents_iterator &__pit, const _Tp &__x)**
- **template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> walker between (const __SequenceCtr1< walker, _Allocator1 > &__parents, const __SequenceCtr2< walker, _Allocator2 > &__children, const _Tp &__x)**

- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker between (const walker &__parent, const children_iterator &__cit, const __SequenceCtr< walker, _Allocator > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker between (const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const parents_iterator &__pit, const _Tp &__x)
- walker split (const walker &__parent, const children_iterator &__ch_it, const walker &__child, const parents_iterator &__pa_it, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> void split (const __SequenceCtr1< walker, _Allocator1 > &__parents, const __SequenceCtr2< walker, _Allocator2 > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker split (const walker &__parent, const children_iterator &__ch_it, const __SequenceCtr< walker, _Allocator > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker split (const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const parents_iterator &__pr_it, const _Tp &__x)
- walker between_back (const walker &__parent, const walker &__child, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker between_back (const walker &__parent, const __SequenceCtr< walker, _Allocator > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker between_back (const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const _Tp &__x)
- walker split_back (const walker &__parent, const walker &__child, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker split_back (const walker &__parent, const __SequenceCtr< walker, _Allocator > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker split_back (const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const _Tp &__x)
- walker between_front (const walker &__parent, const walker &__child, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker between_front (const walker &__parent, const __SequenceCtr< walker, _Allocator > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker between_front (const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const _Tp &__x)
- walker split_front (const walker &__parent, const walker &__child, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker split_front (const walker &__parent, const __SequenceCtr< walker, _Allocator > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker split_front (const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const _Tp &__x)
- void insert_subgraph (_Self &__subgraph, const walker &__parent, const children_iterator &__ch_it, const walker &__child, const parents_iterator &__pa_it)
- void insert_back_subgraph (_Self &__subgraph, const walker &__parent, const walker &__child)
- void insert_front_subgraph (_Self &__subgraph, const walker &__parent, const walker &__child)

- void add_edge (const walker &__parent, const children_iterator &__ch_it, const walker &__-child, const parents_iterator &__pa_it)
- void add_edge_back (const walker &__parent, const walker &__child)
- void add_edge_front (const walker &__parent, const walker &__child)

7.27.1 Detailed Description

template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_-ALLOCATOR(_Tp)> class dag< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >

This class constructs an unlabeled directed acyclic graph (DAG). By default, the children and the parents are collected in an STL vector, but the container can be replaced by any other sequential container.

Definition at line 2399 of file vgtl_dag.h.

7.27.2 Member Typedef Documentation

7.27.2.1 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_-DEFAULT_ALLOCATOR(_Tp)> typedef _Base::children_iterator dag::children_iterator

the children iterator

Reimplemented from dgraph.

Definition at line 2417 of file vgtl_dag.h.

7.27.2.2 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_-DEFAULT_ALLOCATOR(_Tp)> typedef _Base::const_walker dag::const_walker

the const walker

Reimplemented from dgraph.

Definition at line 2415 of file vgtl_dag.h.

7.27.2.3 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_-DEFAULT_ALLOCATOR(_Tp)> typedef _Base::erased_part dag::erased_part

the erased part constructed in erasing subgraphs

Reimplemented from dgraph.

Definition at line 2422 of file vgtl_dag.h.

7.27.2.4 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_-DEFAULT_ALLOCATOR(_Tp)> typedef _Base::parents_iterator dag::parents_iterator

the parents iterator

Reimplemented from dgraph.

Definition at line 2419 of file vgtl_dag.h.

7.27.2.5 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> typedef _Base::walker dag::walker

the walker

Reimplemented from dgraph.

Definition at line 2413 of file vgtl_dag.h.

7.27.3 Constructor & Destructor Documentation

7.27.3.1 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> dag< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::dag (const allocator_type & __a = allocator_type()) [inline, explicit]

standard constructor

Definition at line 2426 of file vgtl_dag.h.

7.27.3.2 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> dag< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::dag (const _Self & __dag) [inline]

copy constructor

Definition at line 2429 of file vgtl_dag.h.

7.27.3.3 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> dag< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::dag (const _Base & __dag) [inline]

construct `dag` from directed graph

Definition at line 2435 of file vgtl_dag.h.

7.27.3.4 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> dag< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::dag (const erased_part & __ep) [inline]

construct `dag` from erased part

Definition at line 2443 of file vgtl_dag.h.

7.27.4 Member Function Documentation

7.27.4.1 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::add_edge

(const walker & __parent, const children_iterator & __ch_it, const walker & __child, const parents_iterator & __pa_it) `[inline, inherited]`

add an edge between `__parent` and `__child` at specific positions `__ch_it` and `__pa_it`.

Definition at line 2137 of file vgtl_dag.h.

7.27.4.2 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::add_edge_back (const walker & __parent, const walker & __child) `[inline, inherited]`

add an edge between `__parent` and `__child` at the end of the children and parents containers.

Definition at line 2147 of file vgtl_dag.h.

7.27.4.3 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::add_edge_front (const walker & __parent, const walker & __child) `[inline, inherited]`

add an edge between `__parent` and `__child` at the beginning of the children and parents containers.

Definition at line 2157 of file vgtl_dag.h.

7.27.4.4 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between (const __SequenceCtr< walker, _Allocator > & __parents, const walker & __child, const parents_iterator & __pit, const _Tp & __x) `[inline, inherited]`

here a new node is inserted between many parents and one child but the previous bonds are not broken, the node is always new

Definition at line 2273 of file vgtl_dag.h.

7.27.4.5 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between (const walker & __parent, const children_iterator & __cit, const __SequenceCtr< walker, _Allocator > & __children, const _Tp & __x) `[inline, inherited]`

here a new node is inserted between one parent and many children but the previous bonds are not broken, the node is always new

Definition at line 2173 of file vgtl_dag.h.

7.27.4.6 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between (const __SequenceCtr1< walker, _Allocator1 > & __parents, const __SequenceCtr2< walker, _Allocator2 > & __children, const _Tp & __x) `[inline, inherited]`

here a new node is inserted between many parents and many children but the previous bonds are not broken, the node is always new

Definition at line 2027 of file vgtl_dag.h.

7.27.4.7 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between (const walker & __parent, const children_iterator & __cit, const walker & __child, const parents_iterator & __pit, const _Tp & __x) `[inline, inherited]`

here a new node is inserted between a parent node and a child node but the previous bonds between the two are not broken, the node is always new with data __x.

Definition at line 1925 of file vgtl_dag.h.

7.27.4.8 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between_back (const __SequenceCtr< walker, _Allocator > & __parents, const walker & __child, const _Tp & __x) `[inline, inherited]`

here a new node is inserted between many parents and one child but the previous bonds are not broken, the node is always new. At the child the new parent is put last.

Definition at line 2327 of file vgtl_dag.h.

7.27.4.9 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between_back (const walker & __parent, const __SequenceCtr< walker, _Allocator > & __children, const _Tp & __x) `[inline, inherited]`

here a new node is inserted between one parent and many children but the previous bonds are not broken, the node is always new. At the parent the new child is put last.

Definition at line 2228 of file vgtl_dag.h.

7.27.4.10 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between_back (const walker & __parent, const walker & __child, const _Tp & __x) `[inline, inherited]`

insert the node as the last child between parent and child, without breaking old bonds.

Definition at line 1960 of file vgtl_dag.h.

7.27.4.11 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between_-

front (const __SequenceCtr< walker, _Allocator > & __parents, const walker & __child, const _Tp & __x) `[inline, inherited]`

here a new node is inserted between many parents and one child but the previous bonds are not broken, the node is always new. At the child the new parent is put first.

Definition at line 2355 of file vgtl_dag.h.

7.27.4.12 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between_front (const walker & __parent, const __SequenceCtr< walker, _Allocator > & __children, const _Tp & __x) `[inline, inherited]`

here a new node is inserted between one parent and many children but the previous bonds are not broken, the node is always new. At the parent the new child is put first.

Definition at line 2258 of file vgtl_dag.h.

7.27.4.13 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between_front (const walker & __parent, const walker & __child, const _Tp & __x) `[inline, inherited]`

Here the inserted node is the first child of its parent and first parent of its child. Insert the node without breaking old bonds.

Definition at line 1991 of file vgtl_dag.h.

7.27.4.14 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> bool dag< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::check_acyclicity (const walker & __parent, const walker & __child) `[inline]`

This method checks, whether the dag is indeed acyclic. This is NYI!

Definition at line 2466 of file vgtl_dag.h.

7.27.4.15 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::clear () `[inline, inherited]`

empty the graph

Reimplemented from __DG< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator, _SequenceCtr< void *, _PtrAlloc >::iterator, _Alloc >.

Definition at line 1918 of file vgtl_dag.h.

7.27.4.16 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::insert_-

back_subgraph (_Self & __subgraph, const walker & __parent, const walker & __child) `[inline, inherited]`

here a subgraph is inserted between a parent and a child, at the end of the children resp. parents lists.

Definition at line 2096 of file vgtl_dag.h.

7.27.4.17 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::insert_front_subgraph (_Self & __subgraph, const walker & __parent, const walker & __child) `[inline, inherited]`

here a subgraph is inserted between a parent and a child, at the front of the children resp. parents lists.

Definition at line 2109 of file vgtl_dag.h.

7.27.4.18 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::insert_subgraph (_Self & __subgraph, const walker & __parent, const children_iterator & __ch_it, const walker & __child, const parents_iterator & __pa_it) `[inline, inherited]`

here a subgraph is inserted between a parent and a child, at specific positions `__ch_it` **and** `__pa_it`**.**

Definition at line 2085 of file vgtl_dag.h.

7.27.4.19 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> _Self& dag< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::operator= (const erased_part & __ep) `[inline]`

assignment from erased part

Reimplemented from dgraph.

Definition at line 2490 of file vgtl_dag.h.

7.27.4.20 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> _Self& dag< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::operator= (const _RV_DG & __rl) `[inline]`

assignment from part of an erased part

Reimplemented from dgraph.

Definition at line 2482 of file vgtl_dag.h.

7.27.4.21 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split (const __SequenceCtr< walker, _Allocator > & __parents, const walker & __child, const parents_iterator & __pr_it, const _Tp & __x) `[inline, inherited]`

here a new node is inserted between many parents and one child, and the previous bonds are broken, the node is always new.

Definition at line 2286 of file vgtl_dag.h.

7.27.4.22 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split (const walker & __*parent*, const children_iterator & __*ch_it*, const __SequenceCtr< walker, _Allocator > & __*children*, const _Tp & __*x*) `[inline, inherited]`

here a new node is inserted between one parent and many children, and the previous bonds are broken, the node is always new.

Definition at line 2186 of file vgtl_dag.h.

7.27.4.23 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split (const __SequenceCtr1< walker, _Allocator1 > & __*parents*, const __SequenceCtr2< walker, _Allocator2 > & __*children*, const _Tp & __*x*) `[inline, inherited]`

here a new node is inserted between many parents and many children, and the previous bonds are broken, the node is always new.

Definition at line 2059 of file vgtl_dag.h.

7.27.4.24 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split (const walker & __*parent*, const children_iterator & __*ch_it*, const walker & __*child*, const parents_iterator & __*pa_it*, const _Tp & __*x*) `[inline, inherited]`

here a new node is inserted between a parent node and a child node and the previous bonds between them are broken, the node is always new with data `__x`.

Definition at line 1938 of file vgtl_dag.h.

7.27.4.25 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split_back (const __SequenceCtr< walker, _Allocator > & __*parents*, const walker & __*child*, const _Tp & __*x*) `[inline, inherited]`

here a new node is inserted between many parents and one child, and the previous bonds are broken, the node is always new. At the child the new parent is put last.

Definition at line 2313 of file vgtl_dag.h.

7.27.4.26 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_-

DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split_back (const walker & __*parent*, const __SequenceCtr< walker, _Allocator > & __*children*, const _Tp & __*x*) `[inline, inherited]`

here a new node is inserted between one parent and many children, and the previous bonds are broken, the node is always new. At the parent the new child is put last.

Definition at line 2213 of file vgtl_dag.h.

7.27.4.27 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split_back (const walker & __*parent*, const walker & __*child*, const _Tp & __*x*) `[inline, inherited]`

insert the node as the last child between parent and child, with breaking old bonds.

Definition at line 1973 of file vgtl_dag.h.

7.27.4.28 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split_front (const __SequenceCtr< walker, _Allocator > & __*parents*, const walker & __*child*, const _Tp & __*x*) `[inline, inherited]`

here a new node is inserted between many parents and one child, and the previous bonds are broken, the node is always new. At the child the new parent is put first.

Definition at line 2341 of file vgtl_dag.h.

7.27.4.29 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split_front (const walker & __*parent*, const __SequenceCtr< walker, _Allocator > & __*children*, const _Tp & __*x*) `[inline, inherited]`

here a new node is inserted between one parent and many children, and the previous bonds are broken, the node is always new. At the parent the new child is put first.

Definition at line 2243 of file vgtl_dag.h.

7.27.4.30 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split_front (const walker & __*parent*, const walker & __*child*, const _Tp & __*x*) `[inline, inherited]`

Here the inserted node is the first child of its parent and first parent of its child. Insert the node and break old bonds.

Definition at line 2004 of file vgtl_dag.h.

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

- vgtl_dag.h

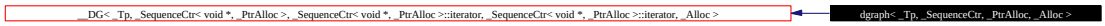
7.28 dgraph Class Template Reference

unlabeled directed graph.

`#include <vgtl_dag.h>`

Inheritance diagram for dgraph:

Collaboration diagram for dgraph:



Public Types

- typedef _Base::walker walker
- typedef _Base::const_walker const_walker
- typedef _Base::children_iterator children_iterator
- typedef _Base::parents_iterator parents_iterator

Public Methods

- dgraph (const allocator_type &__a=allocator_type())
- dgraph (const _Self &__dg)
- dgraph (const erased_part &__ep, const allocator_type &__a=allocator_type())
- void clear ()
- walker between (const walker &__parent, const children_iterator &__cit, const walker &__child, const parents_iterator &__pit, const _Tp &__x)
- walker split (const walker &__parent, const children_iterator &__ch_it, const walker &__child, const parents_iterator &__pa_it, const _Tp &__x)
- walker between_back (const walker &__parent, const walker &__child, const _Tp &__x)
- walker split_back (const walker &__parent, const walker &__child, const _Tp &__x)
- walker between_front (const walker &__parent, const walker &__child, const _Tp &__x)
- walker split_front (const walker &__parent, const walker &__child, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> walker between (const __SequenceCtr1< walker, _Allocator1 > &__parents, const __SequenceCtr2< walker, _Allocator2 > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> void split (const __SequenceCtr1< walker, _Allocator1 > &__parents, const __SequenceCtr2< walker, _Allocator2 > &__children, const _Tp &__x)
- void insert_subgraph (_Self &__subgraph, const walker &__parent, const children_iterator &__ch_it, const walker &__child, const parents_iterator &__pa_it)
- void insert_back_subgraph (_Self &__subgraph, const walker &__parent, const walker &__child)

- void insert_front_subgraph (_Self &__subgraph, const walker &__parent, const walker &__child)
- void add_edge (const walker &__parent, const children_iterator &__ch_it, const walker &__child, const parents_iterator &__pa_it)
- void add_edge_back (const walker &__parent, const walker &__child)
- void add_edge_front (const walker &__parent, const walker &__child)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker between (const walker &__parent, const children_iterator &__cit, const __SequenceCtr< walker, _Allocator > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker split (const walker &__parent, const children_iterator &__ch_it, const __SequenceCtr< walker, _Allocator > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker split_back (const walker &__parent, const __SequenceCtr< walker, _Allocator > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker between_back (const walker &__parent, const __SequenceCtr< walker, _Allocator > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker split_front (const walker &__parent, const __SequenceCtr< walker, _Allocator > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker between_front (const walker &__parent, const __SequenceCtr< walker, _Allocator > &__children, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker between (const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const parents_iterator &__pit, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker split (const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const parents_iterator &__pr_it, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker split_back (const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker between_back (const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker split_front (const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const _Tp &__x)
- template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker between_front (const __SequenceCtr< walker, _Allocator > &__parents, const walker &__child, const _Tp &__x)
- _Self & operator= (const _RV_DG &__rl)
- _Self & operator= (const erased_part &__ep)

7.28.1 Detailed Description

template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> class dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >

This class constructs an unlabeled directed graph. By default, the children and the parents are collected in an STL vector, but the container can be replaced by any other sequential container.

Definition at line 1869 of file vgtl_dag.h.

7.28.2 Member Typedef Documentation

7.28.2.1 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> typedef _Base::children_iterator dgraph::children_iterator

the children iterator

Reimplemented from __DG< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator, _SequenceCtr< void *, _PtrAlloc >::iterator, _Alloc >.

Reimplemented in dag.

Definition at line 1896 of file vgtl_dag.h.

7.28.2.2 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> typedef _Base::const_walker dgraph::const_walker

the const walker

Reimplemented from __DG< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator, _SequenceCtr< void *, _PtrAlloc >::iterator, _Alloc >.

Reimplemented in dag.

Definition at line 1894 of file vgtl_dag.h.

7.28.2.3 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> typedef _Base::parents_iterator dgraph::parents_iterator

the parents iterator

Reimplemented from __DG< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator, _SequenceCtr< void *, _PtrAlloc >::iterator, _Alloc >.

Reimplemented in dag.

Definition at line 1898 of file vgtl_dag.h.

7.28.2.4 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> typedef _Base::walker dgraph::walker

the walker

Reimplemented from __DG< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator, _SequenceCtr< void *, _PtrAlloc >::iterator, _Alloc >.

Reimplemented in dag.

Definition at line 1892 of file vgtl_dag.h.

7.28.3 Constructor & Destructor Documentation

7.28.3.1 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::dgraph (const allocator_type & __a = allocator_type()) [inline, explicit]

standard constructor

Definition at line 1902 of file vgtl_dag.h.

7.28.3.2 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::dgraph (const _Self & __dg) [inline]

copy constructor

Definition at line 1905 of file vgtl_dag.h.

7.28.3.3 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::dgraph (const erased_part & __ep, const allocator_type & __a = allocator_type()) [inline]

constructor from an erased_part

Definition at line 1908 of file vgtl_dag.h.

7.28.4 Member Function Documentation

7.28.4.1 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::add_edge (const walker & __parent, const children_iterator & __ch_it, const walker & __child, const parents_iterator & __pa_it) [inline]

add an edge between `__parent` and `__child` at specific positions `__ch_it` and `__pa_it`.

Definition at line 2137 of file vgtl_dag.h.

7.28.4.2 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::add_edge_back (const walker & __parent, const walker & __child) [inline]

add an edge between `__parent` and `__child` at the end of the children and parents containers.

Definition at line 2147 of file vgtl_dag.h.

7.28.4.3 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::add_edge_front (const walker & __parent, const walker & __child) [inline]

add an edge between `__parent` and `__child` at the beginning of the children and parents containers.

Definition at line 2157 of file vgtl_dag.h.

7.28.4.4 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between (const __SequenceCtr< walker, _Allocator > & __*parents*, const walker & __*child*, const parents_iterator & __*pit*, const _Tp & __*x*) `[inline]`

here a new node is inserted between many parents and one child but the previous bonds are not broken, the node is always new

Definition at line 2273 of file vgtl_dag.h.

7.28.4.5 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between (const walker & __*parent*, const children_iterator & __*cit*, const __SequenceCtr< walker, _Allocator > & __*children*, const _Tp & __*x*) `[inline]`

here a new node is inserted between one parent and many children but the previous bonds are not broken, the node is always new

Definition at line 2173 of file vgtl_dag.h.

7.28.4.6 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between (const __SequenceCtr1< walker, _Allocator1 > & __*parents*, const __SequenceCtr2< walker, _Allocator2 > & __*children*, const _Tp & __*x*) `[inline]`

here a new node is inserted between many parents and many children but the previous bonds are not broken, the node is always new

Definition at line 2027 of file vgtl_dag.h.

7.28.4.7 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between (const walker & __*parent*, const children_iterator & __*cit*, const walker & __*child*, const parents_iterator & __*pit*, const _Tp & __*x*) `[inline]`

here a new node is inserted between a parent node and a child node but the previous bonds between the two are not broken, the node is always new with data __x.

Definition at line 1925 of file vgtl_dag.h.

7.28.4.8 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between_-

back (const __SequenceCtr< walker, _Allocator > & __parents, const walker & __child, const _Tp & __x) `[inline]`

here a new node is inserted between many parents and one child but the previous bonds are not broken, the node is always new. At the child the new parent is put last.

Definition at line 2327 of file vgtl_dag.h.

7.28.4.9 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between_back (const walker & __parent, const __SequenceCtr< walker, _Allocator > & __children, const _Tp & __x) `[inline]`

here a new node is inserted between one parent and many children but the previous bonds are not broken, the node is always new. At the parent the new child is put last.

Definition at line 2228 of file vgtl_dag.h.

7.28.4.10 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between_back (const walker & __parent, const walker & __child, const _Tp & __x) `[inline]`

insert the node as the last child between parent and child, without breaking old bonds.

Definition at line 1960 of file vgtl_dag.h.

7.28.4.11 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between_front (const __SequenceCtr< walker, _Allocator > & __parents, const walker & __child, const _Tp & __x) `[inline]`

here a new node is inserted between many parents and one child but the previous bonds are not broken, the node is always new. At the child the new parent is put first.

Definition at line 2355 of file vgtl_dag.h.

7.28.4.12 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between_front (const walker & __parent, const __SequenceCtr< walker, _Allocator > & __children, const _Tp & __x) `[inline]`

here a new node is inserted between one parent and many children but the previous bonds are not broken, the node is always new. At the parent the new child is put first.

Definition at line 2258 of file vgtl_dag.h.

7.28.4.13 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __-

VGTL_DEFAULT_ALLOCATOR(_Tp)> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::between_front (const walker & __*parent*, const walker & __*child*, const _Tp & __*x*) `[inline]`

Here the inserted node is the first child of its parent and first parent of its child. Insert the node without breaking old bonds.

Definition at line 1991 of file vgtl_dag.h.

7.28.4.14 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::clear () `[inline]`

empty the graph

Reimplemented from __DG< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator, _SequenceCtr< void *, _PtrAlloc >::iterator, _Alloc >.

Definition at line 1918 of file vgtl_dag.h.

7.28.4.15 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::insert_back_subgraph (_Self & __*subgraph*, const walker & __*parent*, const walker & __*child*) `[inline]`

here a subgraph is inserted between a parent and a child, at the end of the children resp. parents lists.

Definition at line 2096 of file vgtl_dag.h.

7.28.4.16 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::insert_front_subgraph (_Self & __*subgraph*, const walker & __*parent*, const walker & __*child*) `[inline]`

here a subgraph is inserted between a parent and a child, at the front of the children resp. parents lists.

Definition at line 2109 of file vgtl_dag.h.

7.28.4.17 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::insert_subgraph (_Self & __*subgraph*, const walker & __*parent*, const children_iterator & __*ch_it*, const walker & __*child*, const parents_iterator & __*pa_it*) `[inline]`

here a subgraph is inserted between a parent and a child, at specific positions `__ch_it` and `__pa_it`.

Definition at line 2085 of file vgtl_dag.h.

7.28.4.18 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __-

VGTL_DEFAULT_ALLOCATOR(_Tp)> _Self& dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::operator= (const erased_part & __ep) `[inline]`

assignment operator from an erased part

Reimplemented from __DG< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator, _SequenceCtr< void *, _PtrAlloc >::iterator, _Alloc >.

Reimplemented in dag.

Definition at line 2372 of file vgtl_dag.h.

7.28.4.19 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> _Self& dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::operator= (const _RV_DG & __rl) `[inline]`

assignment operator from a part of an erased part

Reimplemented from __DG< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator, _SequenceCtr< void *, _PtrAlloc >::iterator, _Alloc >.

Reimplemented in dag.

Definition at line 2364 of file vgtl_dag.h.

7.28.4.20 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split (const __SequenceCtr< walker, _Allocator > & __parents, const walker & __child, const parents_iterator & __pr_it, const _Tp & __x) `[inline]`

here a new node is inserted between many parents and one child, and the previous bonds are broken, the node is always new.

Definition at line 2286 of file vgtl_dag.h.

7.28.4.21 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split (const walker & __parent, const children_iterator & __ch_it, const __SequenceCtr< walker, _Allocator > & __children, const _Tp & __x) `[inline]`

here a new node is inserted between one parent and many children, and the previous bonds are broken, the node is always new.

Definition at line 2186 of file vgtl_dag.h.

7.28.4.22 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr1, template< class __Tp, class __AllocTp > class __SequenceCtr2, class _Allocator1, class _Allocator2> void dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split (const __SequenceCtr1< walker, _Allocator1 > & __parents, const __SequenceCtr2< walker, _Allocator2 > & __children, const _Tp & __x) `[inline]`

here a new node is inserted between many parents and many children, and the previous bonds are broken, the node is always new.

Definition at line 2059 of file vgtl_dag.h.

7.28.4.23 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split (const walker & __parent, const children_iterator & __ch_it, const walker & __child, const parents_iterator & __pa_it, const _Tp & __x) `[inline]`

here a new node is inserted between a parent node and a child node and the previous bonds between them are broken, the node is always new with data `__x`**.**

Definition at line 1938 of file vgtl_dag.h.

7.28.4.24 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split_back (const __SequenceCtr< walker, _Allocator > & __parents, const walker & __child, const _Tp & __x) `[inline]`

here a new node is inserted between many parents and one child, and the previous bonds are broken, the node is always new. At the child the new parent is put last.

Definition at line 2313 of file vgtl_dag.h.

7.28.4.25 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split_back (const walker & __parent, const __SequenceCtr< walker, _Allocator > & __children, const _Tp & __x) `[inline]`

here a new node is inserted between one parent and many children, and the previous bonds are broken, the node is always new. At the parent the new child is put last.

Definition at line 2213 of file vgtl_dag.h.

7.28.4.26 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split_back (const walker & __parent, const walker & __child, const _Tp & __x) `[inline]`

insert the node as the last child between parent and child, with breaking old bonds.

Definition at line 1973 of file vgtl_dag.h.

7.28.4.27 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split_front (const __SequenceCtr< walker, _Allocator > & __parents, const walker & __child, const _Tp & __x) `[inline]`

here a new node is inserted between many parents and one child, and the previous bonds are broken, the node is always new. At the child the new parent is put first.

Definition at line 2341 of file vgtl_dag.h.

7.28.4.28 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> template<template< class __Tp, class __AllocTp > class __SequenceCtr, class _Allocator> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split_front (const walker & __*parent*, const __SequenceCtr< walker, _Allocator > & __*children*, const _Tp & __*x*) `[inline]`

here a new node is inserted between one parent and many children, and the previous bonds are broken, the node is always new. At the parent the new child is put first.

Definition at line 2243 of file vgtl_dag.h.

7.28.4.29 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> walker dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::split_front (const walker & __*parent*, const walker & __*child*, const _Tp & __*x*) `[inline]`

Here the inserted node is the first child of its parent and first parent of its child. Insert the node and break old bonds.

Definition at line 2004 of file vgtl_dag.h.

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

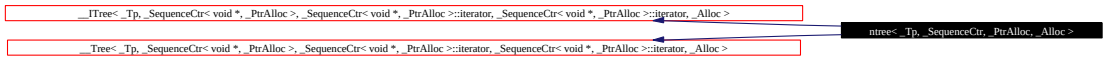
- **vgtl_dag.h**

7.29 ntree Class Template Reference

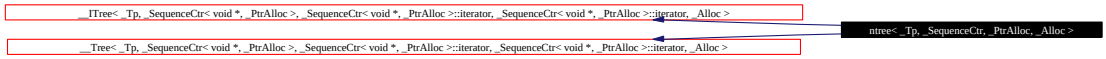
n-ary forest.

`#include <vgtl_tree.h>`

Inheritance diagram for ntree:



Collaboration diagram for ntree:



Public Types

- **typedef _I children_iterator**
- **typedef __one_iterator< void * > parents_iterator**

Public Methods

- void insert (const __walker_base &__position, const _Tp &__x)
- void insert (const __walker_base &__position)
- void push_child (const __walker_base &__position, const _Tp &__x)
- void push_child (const __walker_base &__position)
- void push_children (const __walker_base &__position, size_type __n, const _Tp &__x)
- void push_children (const __walker_base &__position, size_type __n)
- void unshift_child (const __walker_base &__position, const _Tp &__x)
- void unshift_child (const __walker_base &__position)
- void unshift_children (const __walker_base &__position, size_type __n, const _Tp &__x)
- void unshift_children (const __walker_base &__position, size_type __n)
- void push_subtree (const __walker_base &__position, _Self &__subtree)
- void unshift_subtree (const __walker_base &__position, _Self &__subtree)
- bool pop_child (const __walker_base &__position)
- bool shift_child (const __walker_base &__position)
- _Node * pop_subtree (const __walker_base &__position)
- _Node * shift_subtree (const __walker_base &__position)
- _Self & operator= (_Node *__x)
- template<class _Output_Iterator> void add_all_children (_Output_Iterator fi, _Node *_parent)

7.29.1 Detailed Description

template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_-ALLOCATOR(_Tp)> class ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >

This class constructs an n-ary forest with data hooks. By default, the children are collected in a STL vector, but the container can be replaced by any other sequential container.

Definition at line 1632 of file vgtl_graph.h.

7.29.2 Member Typedef Documentation

7.29.2.1 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef _I _Tree_base::children_-iterator [inherited]

iterator for accessing the children

Definition at line 1444 of file vgtl_tree.h.

7.29.2.2 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef __one_iterator<void *> _-Tree_base::parents_iterator [inherited]

iterator for accessing the parents

Definition at line 1446 of file vgtl_tree.h.

7.29.3 Member Function Documentation

7.29.3.1 template<class _Tp, class _Ctr, class _I, class _Alloc> template<class _Output_Iterator> void _Tree_base< _Tp, _Ctr, _I, _Alloc >::add_all_children (_Output_Iterator *fi*, _Node * *_parent*) [inherited]

add all children to the parent `_parent`. `fi` **is a iterator to the children container of the parent**

7.29.3.2 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::insert (const __walker_base & *__position*) [inline]

Insert a node with default data at position `__position`.

Definition at line 2363 of file vgtl_tree.h.

7.29.3.3 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::insert (const __walker_base & *__position*, const _Tp & *__x*) [inline]

Insert a node with data `__x` **at position** `__position`.

Definition at line 2335 of file vgtl_tree.h.

7.29.3.4 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> _Self& ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::operator= (_Node * *__x*) [inline]

assign a tree from one node -> make this node the root node. This is useful for making trees out of erased subtrees.

Reimplemented from __Tree< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator, _SequenceCtr< void *, _PtrAlloc >::iterator, _Alloc >.

Definition at line 2490 of file vgtl_tree.h.

7.29.3.5 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> bool ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::pop_child (const __walker_base & *__position*) [inline]

erase the last (leaf) child of node `__position`. **This works if and only if the child is a leaf.**

Definition at line 2432 of file vgtl_tree.h.

7.29.3.6 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> _Node* ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::pop_subtree (const __walker_base & *__position*) [inline]

erase the subtree position `__position`, **whose top node is the last child of the node, and return its top node.**

Definition at line 2460 of file vgtl_tree.h.

7.29.3.7 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::push_child (const __walker_base & __*position*) `[inline]`

add a child below `__position` with default data, at the last position in the `__position` - node's children container

Definition at line 2373 of file vgtl_tree.h.

7.29.3.8 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::push_child (const __walker_base & __*position*, const _Tp & __*x*) `[inline]`

add a child below `__position` with data `__x`, at the last position in the `__position` - node's children container

Definition at line 2368 of file vgtl_tree.h.

7.29.3.9 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::push_children (const __walker_base & __*position*, size_type __*n*) `[inline]`

add `__n` children below `__position` with default data, after the last position in the `__position` - node's children container

Definition at line 2384 of file vgtl_tree.h.

7.29.3.10 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::push_children (const __walker_base & __*position*, size_type __*n*, const _Tp & __*x*) `[inline]`

add `__n` children below `__position` with data `__x`, after the last position in the `__position` - node's children container

Definition at line 2378 of file vgtl_tree.h.

7.29.3.11 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::push_subtree (const __walker_base & __*position*, _Self & __*subtree*) `[inline]`

add a complete subtree `__subtree` below position `__position` and last children iterator position.

Definition at line 2412 of file vgtl_tree.h.

7.29.3.12 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> bool ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::shift_child (const __walker_base & __*position*) `[inline]`

erase the first (leaf) child of node `__position`. This works if and only if the child is a leaf.

Definition at line 2446 of file vgtl_tree.h.

7.29.3.13 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_-ALLOCATOR(_Tp)> _Node* ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::shift_subtree (const __walker_base & __*position*) `[inline]`

erase the subtree position `__position`**, whose top node is the last child of the node, and return its top node.**

Definition at line 2475 of file vgtl_tree.h.

7.29.3.14 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_-ALLOCATOR(_Tp)> void ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::unshift_child (const __-walker_base & __*position*) `[inline]`

add a child below `__position` **with default data, at the first position in the** `__position` **- node's children container**

Definition at line 2394 of file vgtl_tree.h.

7.29.3.15 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_-ALLOCATOR(_Tp)> void ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::unshift_child (const __-walker_base & __*position*, const _Tp & __*x*) `[inline]`

add a child below `__position` **with data** `__x`**, at the first position in the** `__position` **- node's children container**

Definition at line 2389 of file vgtl_tree.h.

7.29.3.16 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_-ALLOCATOR(_Tp)> void ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::unshift_children (const __walker_base & __*position*, size_type __*n*) `[inline]`

add `__n` **children below** `__position` **with default data, after the first position in the** `__position` **- node's children container**

Definition at line 2405 of file vgtl_tree.h.

7.29.3.17 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_-ALLOCATOR(_Tp)> void ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::unshift_children (const __walker_base & __*position*, size_type __*n*, const _Tp & __*x*) `[inline]`

add `__n` **children below** `__position` **with data** `__x`**, after the first position in the** `__position` **- node's children container**

Definition at line 2399 of file vgtl_tree.h.

7.29.3.18 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_-ALLOCATOR(_Tp)> void ntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::unshift_subtree (const __walker_base & __*position*, _Self & __*subtree*) `[inline]`

add a complete subtree `__subtree` **below position** `__position` **and first children iterator position.**

Definition at line 2422 of file vgtl_tree.h.

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

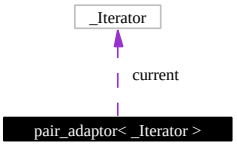
- **vgtl_graph.h**
- **vgtl_tree.h**

7.30 pair_adaptor Class Template Reference

adaptor for an iterator over a pair to an iterator returning the second element.

`#include <vgtl_intadapt.h>`

Collaboration diagram for pair_adaptor:



Public Types

- **typedef std::iterator_traits< _Iterator >::iterator_category iterator_category**
  standard iterator definitions.
- **typedef std::iterator_traits< _Iterator >::difference_type difference_type**
  standard iterator definitions.
- **typedef std::iterator_traits< _Iterator >::value_type p_value_type**
  standard iterator definitions.
- **typedef std::iterator_traits< _Iterator >::pointer p_pointer**
  standard iterator definitions.
- **typedef std::iterator_traits< _Iterator >::reference p_reference**
  standard iterator definitions.
- **typedef p_value_type::second_type value_type**
  standard iterator definitions.
- **typedef value_type & reference**
  standard iterator definitions.
- **typedef value_type * pointer**
  standard iterator definitions.
- **typedef p_value_type::first_type key_type**
  additional definitions for the key type.

- **typedef key_type & key_reference**
  additional definitions for the key type.
- **typedef key_type * key_pointer**
  additional definitions for the key type.

Public Methods

- **pair_adaptor ()**
  standard constructor.
- **pair_adaptor (iterator_type __x)**
  constructor setting the position.
- **pair_adaptor (const _Self &__x)**
  copy constructor.
- **template<class _Iter> pair_adaptor (const pair_adaptor< _Iter > &__x)**
  a copy constructor setting the position from another pair adaptor.
- **iterator_type base () const**
  return the base iterator.
- **reference operator * () const**
  dereference operator.
- **pointer operator → () const**
  pointer operator.
- **key_reference operator~ () const**
  dereference to the key value.
- **_Self & operator= (const iterator_type &__x)**
  assignment operator setting the position from base iterator.
- **_Self & operator++ ()**
  standard increment, decrement operators.
- **_Self operator++ (int)**
  standard increment, decrement operators.
- **_Self & operator– ()**
  standard increment, decrement operators.
- **_Self operator– (int)**
  standard increment, decrement operators.

- _Self operator+ (difference_type __n) const
  standard random access operators.
- _Self & operator+= (difference_type __n)
  standard random access operators.
- _Self operator- (difference_type __n) const
  standard random access operators.
- _Self & operator-= (difference_type __n)
  standard random access operators.
- reference operator[ ] (difference_type __n) const
  standard random access operators.

- bool operator== (const iterator_type &__x)
  standard comparison operator.
- bool operator!= (const iterator_type &__x)
  standard comparison operator.

Protected Attributes

- _Iterator current
  the original iterator.

7.30.1 Detailed Description

template<class _Iterator> class pair_adaptor< _Iterator >

This adaptor transforms an iterator returning a pair (e.g. a `map` or `multimap` iterator) to an iterator returning only the value part. There is another operator (~), which returns the key value for a given position.

Definition at line 77 of file vgtl_intadapt.h.

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

- vgtl_intadapt.h

7.31 pointer_adaptor Class Template Reference

adaptor transforming a comparison predicate to pointers.

```
#include <vgtl_intadapt.h>
```

Public Types

- **typedef __a1 * first_argument_type**
  standard binary predicate definitions.
- **typedef __a2 * second_argument_type**
  standard binary predicate definitions.
- **typedef _Compare::result_type result_type**
  standard binary predicate definitions.

Public Methods

- **result_type operator() (__a1 *arg1, __a2 *arg2) const**
  the real adaptor.

7.31.1 Detailed Description

template<class _Compare> class pointer_adaptor< _Compare >

This adaptor transforms a binary comparison predicate for two data types `__a1` and `__a2` to a comparison predicate on the pointers to `__a1` and `__a2`, respectively.

Definition at line 46 of file vgtl_intadapt.h.

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

- **vgtl_intadapt.h**

7.32 postorder_visitor Class Template Reference

postorder visitor base class.

```
#include <vgtl_visitor.h>
```

Public Methods

- **virtual void vinit ()**
- **virtual return_value vvalue ()**
- **virtual void vcollect (const return_value &__r)**

- **virtual void init ()**
- **virtual bool postorder (const _Node &__n)**
- **virtual void collect (const _Node &__n, const return_value &__r)**
- **virtual return_value value ()**

7.32.1 Detailed Description

template<class _Node, class _Ret> class postorder_visitor< _Node, _Ret >

This is the base class of all postorder visitors. They can be used in all recursive postorder walks.

Definition at line 83 of file vgtl_visitor.h.

7.32.2 Member Function Documentation

7.32.2.1 template<class _Node, class _Ret> virtual void postorder_visitor< _Node, _Ret >::collect (const _Node & __n, const return_value & __r) [inline, virtual]

virtual functions for ordinary nodes

Definition at line 101 of file vgtl_visitor.h.

7.32.2.2 template<class _Node, class _Ret> virtual void postorder_visitor< _Node, _Ret >::init () [inline, virtual]

virtual functions for ordinary nodes

Definition at line 99 of file vgtl_visitor.h.

7.32.2.3 template<class _Node, class _Ret> virtual bool postorder_visitor< _Node, _Ret >::postorder (const _Node & __n) [inline, virtual]

virtual functions for ordinary nodes

Definition at line 100 of file vgtl_visitor.h.

7.32.2.4 template<class _Node, class _Ret> virtual return_value postorder_visitor< _Node, _Ret >::value () [inline, virtual]

virtual functions for ordinary nodes

Definition at line 102 of file vgtl_visitor.h.

7.32.2.5 template<class _Node, class _Ret> virtual void postorder_visitor< _Node, _Ret >::vcollect (const return_value & __r) [inline, virtual]

virtual functions for virtual nodes

Definition at line 94 of file vgtl_visitor.h.

7.32.2.6 template<class _Node, class _Ret> virtual void postorder_visitor< _Node, _Ret >::vinit () [inline, virtual]

virtual functions for virtual nodes

Definition at line 92 of file vgtl_visitor.h.

7.32.2.7 template<class _Node, class _Ret> virtual return_value postorder_visitor< _Node, _Ret >::vvalue () [inline, virtual]

virtual functions for virtual nodes

Definition at line 93 of file vgtl_visitor.h.

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

- **vgtl_visitor.h**

7.33 preorder_visitor Class Template Reference

preorder visitor base class.

```
#include <vgtl_visitor.h>
```

Public Types

- **typedef _Ret return_value**

Public Methods

- **preorder_visitor ()**

- **virtual void vinit ()**
- **virtual return_value vvalue ()**
- **virtual void vcollect (const return_value &__r)**

- **virtual bool preorder (const _Node &__n)**
- **virtual void collect (const _Node &__n, const return_value &__r)**
- **virtual return_value value ()**

7.33.1 Detailed Description

template<class _Node, class _Ret> class preorder_visitor< _Node, _Ret >

This is the base class of all preorder visitors. They can be used in all recursive preorder walks.

Definition at line 52 of file vgtl_visitor.h.

7.33.2 Member Typedef Documentation

7.33.2.1 template<class _Node, class _Ret> typedef _Ret preorder_visitor::return_value

the return value type

Definition at line 56 of file vgtl_visitor.h.

7.33.3 Constructor & Destructor Documentation

7.33.3.1 template<class _Node, class _Ret> preorder_visitor< _Node, _Ret >::preorder_visitor () [inline]

standard constructor

Definition at line 59 of file vgtl_visitor.h.

7.33.4 Member Function Documentation

7.33.4.1 template<class _Node, class _Ret> virtual void preorder_visitor< _Node, _Ret >::collect (const _Node & __n, const return_value & __r) [inline, virtual]

virtual functions for ordinary nodes

Definition at line 71 of file vgtl_visitor.h.

7.33.4.2 template<class _Node, class _Ret> virtual bool preorder_visitor< _Node, _Ret >::preorder (const _Node & __n) [inline, virtual]

virtual functions for ordinary nodes

Definition at line 70 of file vgtl_visitor.h.

7.33.4.3 template<class _Node, class _Ret> virtual return_value preorder_visitor< _Node, _Ret >::value () [inline, virtual]

virtual functions for ordinary nodes

Definition at line 72 of file vgtl_visitor.h.

7.33.4.4 template<class _Node, class _Ret> virtual void preorder_visitor< _Node, _Ret >::vcollect (const return_value & __r) [inline, virtual]

virtual functions for virtual nodes

Definition at line 65 of file vgtl_visitor.h.

7.33.4.5 template<class _Node, class _Ret> virtual void preorder_visitor< _Node, _Ret >::vinit () [inline, virtual]

virtual functions for virtual nodes

Definition at line 63 of file vgtl_visitor.h.

7.33.4.6 template<class _Node, class _Ret> virtual return_value preorder_visitor< _Node, _Ret >::vvalue () [inline, virtual]

virtual functions for virtual nodes

Definition at line 64 of file vgtl_visitor.h.

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

- **vgtl_visitor.h**

7.34 prepost_visitor Class Template Reference

pre+postorder visitor base class.

```
#include <vgtl_visitor.h>
```

Public Methods

- **virtual void vinit ()**
- **virtual return_value vvalue ()**
- **virtual void vcollect (const return_value &__r)**

- **virtual bool preorder (const _Node &__n)**
- **virtual bool postorder (const _Node &__n)**
- **virtual void collect (const _Node &__n, const return_value &__r)**
- **virtual return_value value ()**

7.34.1 Detailed Description

template<class _Node, class _Ret> class prepost_visitor< _Node, _Ret >

This is the base class of all pre+postorder visitors. They can be used in all recursive walks.

Definition at line 113 of file vgtl_visitor.h.

7.34.2 Member Function Documentation

7.34.2.1 template<class _Node, class _Ret> virtual void prepost_visitor< _Node, _Ret >::collect (const _Node & __n, const return_value & __r) [inline, virtual]

virtual functions for ordinary nodes

Definition at line 131 of file vgtl_visitor.h.

7.34.2.2 template<class _Node, class _Ret> virtual bool prepost_visitor< _Node, _Ret >::postorder (const _Node & __n) [inline, virtual]

virtual functions for ordinary nodes

Definition at line 130 of file vgtl_visitor.h.

7.34.2.3 template<class _Node, class _Ret> virtual bool prepost_visitor< _Node, _Ret >::preorder (const _Node & __n) [inline, virtual]

virtual functions for ordinary nodes

Definition at line 129 of file vgtl_visitor.h.

7.34.2.4 template<class _Node, class _Ret> virtual return_value prepost_visitor< _Node, _Ret >::value () [inline, virtual]

virtual functions for ordinary nodes

Definition at line 132 of file vgtl_visitor.h.

7.34.2.5 template<class _Node, class _Ret> virtual void prepost_visitor< _Node, _Ret >::vcollect (const return_value & __r) [inline, virtual]

virtual functions for virtual nodes

Definition at line 124 of file vgtl_visitor.h.

7.34.2.6 template<class _Node, class _Ret> virtual void prepost_visitor< _Node, _Ret >::vinit () [inline, virtual]

virtual functions for virtual nodes

Definition at line 122 of file vgtl_visitor.h.

7.34.2.7 template<class _Node, class _Ret> virtual return_value prepost_visitor< _Node, _Ret >::vvalue () [inline, virtual]

virtual functions for virtual nodes

Definition at line 123 of file vgtl_visitor.h.

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

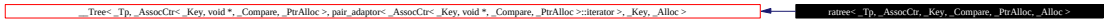
- **vgtl_visitor.h**

7.35 ratree Class Template Reference

n-ary forest with labelled edges.

`#include <vgtl_tree.h>`

Inheritance diagram for ratree:



Collaboration diagram for ratree:

Public Types

- **typedef _I children_iterator**
- **typedef __one_iterator< void * > parents_iterator**

Public Methods

- **_Self & operator= (_Node *__x)**
- **void insert (const __walker_base &__position, const _Tp &__x, const _Key &__k)**
- **void insert (const __walker_base &__position, const _Key &__k)**
- **void clear_children ()**
- **template<class _Output_Iterator> void add_all_children (_Output_Iterator fi, _Node *_parent)**

7.35.1 Detailed Description

template<class _Tp, template< class __Key, class __Ty, class __Compare, class __AllocT > class _AssocCtr = std::multimap, class _Key = string, class _Compare = less<_Key>, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> class ratree< _Tp, _AssocCtr, _Key, _Compare, _PtrAlloc, _Alloc >

This class constructs an n-ary forest without data hooks and labelled edges. By default, the children are collected in a STL multimap, but the container can be replaced by any other associative map container.

Definition at line 2800 of file vgtl_tree.h.

7.35.2 Member Typedef Documentation

7.35.2.1 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef _I _Tree_base::children_iterator [inherited]

iterator for accessing the children

Definition at line 1444 of file vgtl_tree.h.

7.35.2.2 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef __one_iterator<void *> _Tree_base::parents_iterator [inherited]

iterator for accessing the parents

Definition at line 1446 of file vgtl_tree.h.

7.35.3 Member Function Documentation

7.35.3.1 template<class _Tp, class _Ctr, class _I, class _Alloc> template<class _Output_Iterator> void _Tree_base< _Tp, _Ctr, _I, _Alloc >::add_all_children (_Output_Iterator *fi*, _Node * *_parent*) [inherited]

add all children to the parent `_parent`**.** `fi` **is a iterator to the children container of the parent**

7.35.3.2 template<class _Tp, class _Ctr, class _I, class _Alloc> void _Tree_base< _Tp, _Ctr, _I, _Alloc >::clear_children () [inline, inherited]

clear all children of the root node

Definition at line 1465 of file vgtl_tree.h.

7.35.3.3 template<class _Tp, template< class __Key, class __Ty, class __Compare, class __AllocT > class _AssocCtr = std::multimap, class _Key = string, class _Compare = less<_Key>, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void ratree< _Tp, _AssocCtr, _Key, _Compare, _PtrAlloc, _Alloc >::insert (const __walker_base & *__position*, const _Key & *__k*) [inline]

Insert a node with default data and key `__k` **at position** `__position`**.**

Definition at line 2848 of file vgtl_tree.h.

7.35.3.4 template<class _Tp, template< class __Key, class __Ty, class __Compare, class __AllocT > class _AssocCtr = std::multimap, class _Key = string, class _Compare = less<_Key>, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void ratree< _Tp, _AssocCtr, _Key, _Compare, _PtrAlloc, _Alloc >::insert (const __walker_base & *__position*, const _Tp & *__x*, const _Key & *__k*) [inline]

Insert a node with data __x and key _k at position __position.

Definition at line 2822 of file vgtl_tree.h.

7.35.3.5 template<class _Tp, template< class __Key, class __Ty, class __Compare, class __AllocT > class _AssocCtr = std::multimap, class _Key = string, class _Compare = less<_Key>, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> _Self& ratree< _Tp, _AssocCtr, _Key, _Compare, _PtrAlloc, _Alloc >::operator= (_Node * *__x*) [inline]

assign a tree from one node -> make this node the root node. This is useful for making trees out of erased subtrees.

Reimplemented from __Tree< _Tp, _AssocCtr< _Key, void *, _Compare, _PtrAlloc >, pair_adaptor< _AssocCtr< _Key, void *, _Compare, _PtrAlloc >::iterator >, _Key, _Alloc >.

Definition at line 2813 of file vgtl_tree.h.

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

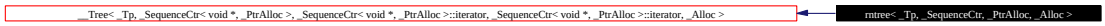
- vgtl_tree.h

7.36 rntree Class Template Reference

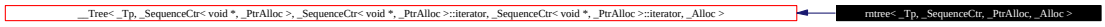
n-ary forest.

`#include <vgtl_tree.h>`

Inheritance diagram for rntree:



Collaboration diagram for rntree:



Public Types

- typedef _I children_iterator
- typedef __one_iterator< void * > parents_iterator

Public Methods

- void insert (const __walker_base &__position, const _Tp &__x)
- void insert (const __walker_base &__position)
- void push_child (const __walker_base &__position, const _Tp &__x)
- void push_child (const __walker_base &__position)

- void push_children (const __walker_base &__position, size_type __n, const _Tp &__x)
- void push_children (const __walker_base &__position, size_type __n)
- void unshift_child (const __walker_base &__position, const _Tp &__x)
- void unshift_child (const __walker_base &__position)
- void unshift_children (const __walker_base &__position, size_type __n, const _Tp &__x)
- void unshift_children (const __walker_base &__position, size_type __n)
- void push_subtree (const __walker_base &__position, _Self &__subtree)
- void unshift_subtree (const __walker_base &__position, _Self &__subtree)
- bool pop_child (const __walker_base &__position)
- bool shift_child (const __walker_base &__position)
- _Node * pop_subtree (const __walker_base &__position)
- _Node * shift_subtree (const __walker_base &__position)
- _Self & operator= (_Node *__x)
- template<class _Output_Iterator> void add_all_children (_Output_Iterator fi, _Node *_parent)

7.36.1 Detailed Description

template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_-ALLOCATOR(_Tp)> class rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >

This class constructs an n-ary forest without data hooks. By default, the children are collected in a STL vector, but the container can be replaced by any other sequential container.

Definition at line 2508 of file vgtl_tree.h.

7.36.2 Member Typedef Documentation

7.36.2.1 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef _I _Tree_base::children_-iterator [inherited]

iterator for accessing the children

Definition at line 1444 of file vgtl_tree.h.

7.36.2.2 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef __one_iterator<void *> _-Tree_base::parents_iterator [inherited]

iterator for accessing the parents

Definition at line 1446 of file vgtl_tree.h.

7.36.3 Member Function Documentation

7.36.3.1 template<class _Tp, class _Ctr, class _I, class _Alloc> template<class _Output_Iterator> void _Tree_base< _Tp, _Ctr, _I, _Alloc >::add_all_children (_Output_Iterator *fi*, _Node * *_parent*) [inherited]

add all children to the parent `_parent`. `fi` is a iterator to the children container of the parent

7.36.3.2 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::insert (const __walker_base & __*position*) `[inline]`

Insert a node with default data at position `__position`**.**

Definition at line 2550 of file vgtl_tree.h.

7.36.3.3 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::insert (const __walker_base & __*position*, const _Tp & __*x*) `[inline]`

Insert a node with data `__x` **at position** `__position`**.**

Definition at line 2522 of file vgtl_tree.h.

7.36.3.4 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> _Self& rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::operator= (_Node * __*x*) `[inline]`

assign a tree from one node -> make this node the root node. This is useful for making trees out of erased subtrees.

Reimplemented from __Tree< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator, _SequenceCtr< void *, _PtrAlloc >::iterator, _Alloc >.

Definition at line 2677 of file vgtl_tree.h.

7.36.3.5 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> bool rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::pop_child (const __walker_base & __*position*) `[inline]`

erase the last (leaf) child of node `__position`**. This works if and only if the child is a leaf.**

Definition at line 2619 of file vgtl_tree.h.

7.36.3.6 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> _Node* rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::pop_subtree (const __walker_base & __*position*) `[inline]`

erase the subtree position `__position`**, whose top node is the last child of the node, and return its top node.**

Definition at line 2647 of file vgtl_tree.h.

7.36.3.7 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::push_child (const __walker_base & __*position*) `[inline]`

add a child below `__position` **with default data, at the last position in the** `__position` **- node's children container**

Definition at line 2560 of file vgtl_tree.h.

7.36.3.8 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::push_child (const __walker_base & __*position*, const _Tp & __*x*) `[inline]`

add a child below `__position` **with data** `__x`**, at the last position in the** `__position` **- node's children container**

Definition at line 2555 of file vgtl_tree.h.

7.36.3.9 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::push_children (const __walker_base & __*position*, size_type __*n*) `[inline]`

add `__n` **children below** `__position` **with default data, after the last position in the** `__position` **- node's children container**

Definition at line 2571 of file vgtl_tree.h.

7.36.3.10 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::push_children (const __walker_base & __*position*, size_type __*n*, const _Tp & __*x*) `[inline]`

add `__n` **children below** `__position` **with data** `__x`**, after the last position in the** `__position` **- node's children container**

Definition at line 2565 of file vgtl_tree.h.

7.36.3.11 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::push_subtree (const __walker_base & __*position*, _Self & __*subtree*) `[inline]`

add a complete subtree `__subtree` **below position** `__position` **and last children iterator position.**

Definition at line 2599 of file vgtl_tree.h.

7.36.3.12 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> bool rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::shift_child (const __walker_base & __*position*) `[inline]`

erase the first (leaf) child of node `__position`**. This works if and only if the child is a leaf.**

Definition at line 2633 of file vgtl_tree.h.

7.36.3.13 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> _Node* rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::shift_subtree (const __walker_base & __*position*) `[inline]`

erase the subtree position __position, whose top node is the last child of the node, and return its top node.

Definition at line 2662 of file vgtl_tree.h.

7.36.3.14 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::unshift_child (const __walker_base & __*position*) [inline]

add a child below __position with default data, at the first position in the __position - node's children container

Definition at line 2581 of file vgtl_tree.h.

7.36.3.15 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::unshift_child (const __walker_base & __*position*, const _Tp & __*x*) [inline]

add a child below __position with data __x, at the first position in the __position - node's children container

Definition at line 2576 of file vgtl_tree.h.

7.36.3.16 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::unshift_children (const __walker_base & __*position*, size_type __*n*) [inline]

add __n children below __position with default data, after the first position in the __position - node's children container

Definition at line 2592 of file vgtl_tree.h.

7.36.3.17 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::unshift_children (const __walker_base & __*position*, size_type __*n*, const _Tp & __*x*) [inline]

add __n children below __position with data __x, after the first position in the __position - node's children container

Definition at line 2586 of file vgtl_tree.h.

7.36.3.18 template<class _Tp, template< class __Ty, class __AllocT > class _SequenceCtr = std::vector, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Tp)> void rntree< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >::unshift_subtree (const __walker_base & __*position*, _Self & __*subtree*) [inline]

add a complete subtree __subtree below position __position and first children iterator position.

Definition at line 2609 of file vgtl_tree.h.

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

- **vgtl_tree.h**

7.37 rstree Class Template Reference

n-ary forest with unsorted edges.

`#include <vgtl_tree.h>`

Inheritance diagram for rstree:

Collaboration diagram for rstree:

Public Types

- **typedef _I children_iterator**
- **typedef __one_iterator< void * > parents_iterator**

Public Methods

- **_Self & operator= (_Node *__x)**
- **template<class _Output_Iterator> void add_all_children (_Output_Iterator fi, _Node *_parent)**

7.37.1 Detailed Description

template<class _Key, class _Compare = less<_Key>, template< class __Key, class __Compare, class _-_AllocT > class _AssocCtr = std::multiset, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Key&)> class rstree< _Key, _Compare, _-AssocCtr, _PtrAlloc, _Alloc >

This class constructs an n-ary forest without data hooks and unsorted edges. By default, the children are collected in a STL multiset, but the container can be replaced by any other associative set container.

Definition at line 2866 of file vgtl_tree.h.

7.37.2 Member Typedef Documentation

7.37.2.1 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef _I _Tree_base::children_-iterator `[inherited]`

iterator for accessing the children

Definition at line 1444 of file vgtl_tree.h.

7.37.2.2 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef __one_iterator<void *> _-Tree_base::parents_iterator `[inherited]`

iterator for accessing the parents

Definition at line 1446 of file vgtl_tree.h.

7.37.3 Member Function Documentation

7.37.3.1 template<class _Tp, class _Ctr, class _I, class _Alloc> template<class _Output_Iterator> void _Tree_base< _Tp, _Ctr, _I, _Alloc >::add_all_children (_Output_Iterator *fi*, _Node * *_parent*) [inherited]

add all children to the parent `_parent`. `fi` **is a iterator to the children container of the parent**

7.37.3.2 template<class _Key, class _Compare = less<_Key>, template< class __Key, class __Compare, class __AllocT > class _AssocCtr = std::multiset, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Key&)> _Self& rstree< _Key, _Compare, _AssocCtr, _PtrAlloc, _Alloc >::operator= (_Node * __*x*) [inline]

assign a tree from one node -> make this node the root node. This is useful for making trees out of erased subtrees.

Reimplemented from __Tree< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator, _Key &, _Alloc >.

Definition at line 2880 of file vgtl_tree.h.

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

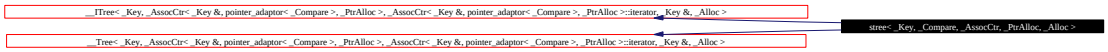
- **vgtl_tree.h**

7.38 stree Class Template Reference

n**-ary forest with unsorted edges.**

`#include <vgtl_tree.h>`

Inheritance diagram for stree:



Collaboration diagram for stree:

Public Types

- **typedef _I children_iterator**
- **typedef __one_iterator< void * > parents_iterator**

Public Methods

- **_Self & operator= (_Node *__x)**
- **template<class _Output_Iterator> void add_all_children (_Output_Iterator fi, _Node *_parent)**

7.38.1 Detailed Description

template<class _Key, class _Compare = less<_Key>, template< class __Key, class __Compare, class __AllocT > class _AssocCtr = multiset, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Key&)> class stree< _Key, _Compare, _AssocCtr, _PtrAlloc, _Alloc >

This class constructs an n-ary forest with data hooks and unsorted edges. By default, the children are collected in a STL multiset, but the container can be replaced by any other associative set container.

Definition at line 1817 of file vgtl_graph.h.

7.38.2 Member Typedef Documentation

7.38.2.1 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef _I _Tree_base::children_iterator [inherited]

iterator for accessing the children

Definition at line 1444 of file vgtl_tree.h.

7.38.2.2 template<class _Tp, class _Ctr, class _I, class _Alloc> typedef __one_iterator<void *> _Tree_base::parents_iterator [inherited]

iterator for accessing the parents

Definition at line 1446 of file vgtl_tree.h.

7.38.3 Member Function Documentation

7.38.3.1 template<class _Tp, class _Ctr, class _I, class _Alloc> template<class _Output_Iterator> void _Tree_base< _Tp, _Ctr, _I, _Alloc >::add_all_children (_Output_Iterator *fi*, _Node * *_parent*) [inherited]

add all children to the parent `_parent`. `fi` is a iterator to the children container of the parent

7.38.3.2 template<class _Key, class _Compare = less<_Key>, template< class __Key, class __Compare, class __AllocT > class _AssocCtr = multiset, class _PtrAlloc = __VGTL_DEFAULT_ALLOCATOR(void *), class _Alloc = __VGTL_DEFAULT_ALLOCATOR(_Key&)> _Self& stree< _Key, _Compare, _AssocCtr, _PtrAlloc, _Alloc >::operator= (_Node * *__x*) [inline]

assign a tree from one node -> make this node the root node. This is useful for making trees out of erased subtrees.

Reimplemented from __Tree< _Key, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >, _AssocCtr< _Key &, pointer_adaptor< _Compare >, _PtrAlloc >::iterator, _Key &, _Alloc >.

Definition at line 2779 of file vgtl_tree.h.

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

- **vgtl_graph.h**
- **vgtl_tree.h**

8 Vienna Graph Template Library File Documentation

8.1 array_vector.h File Reference

Compounds

- **class array_vector**
  STL vector wrapper for C *array.*

8.1.1 Detailed Description

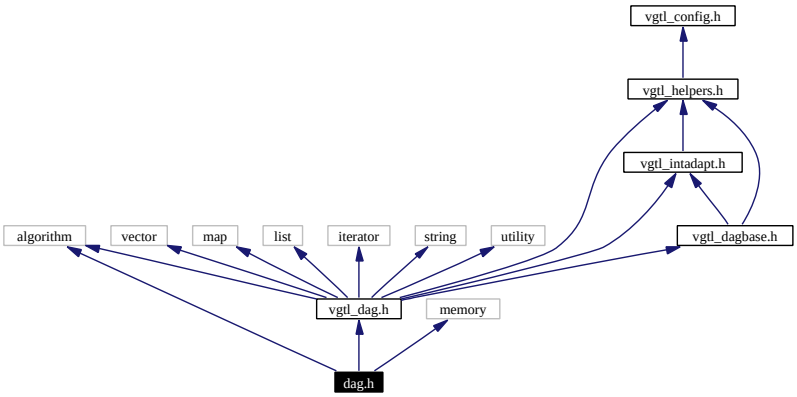
This is the external header file intended for direct use.

Definition in file array_vector.h.

8.2 dag.h File Reference

```
#include <algorithm>
#include <memory>
#include <vgtl_dag.h>
```

Include dependency graph for dag.h:



8.2.1 Detailed Description

This is the external header file intended for direct use.

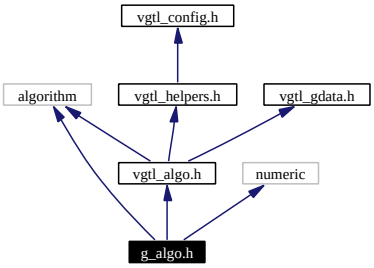
Definition in file dag.h.

8.3 g_algo.h File Reference

```
#include <algorithm>
#include <vgtl_algo.h>
```

`#include <numeric>`

Include dependency graph for g_algo.h:



8.3.1 Detailed Description

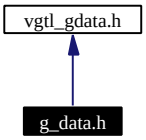
This is the external header file intended for direct use.

Definition in file g_algo.h.

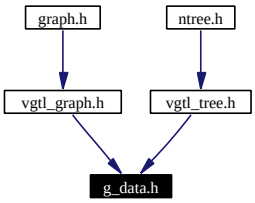
8.4 g_data.h File Reference

`#include <vgtl_gdata.h>`

Include dependency graph for g_data.h:



This graph shows which files directly or indirectly include this file:



8.4.1 Detailed Description

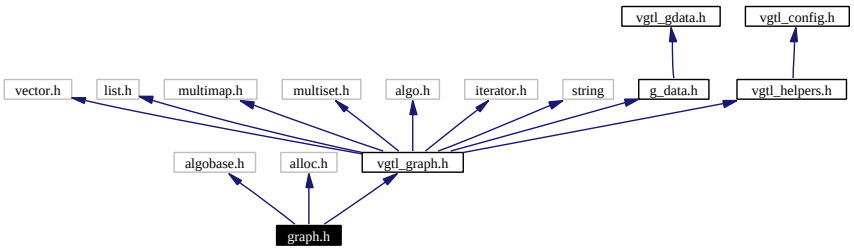
This is the external header file intended for direct use.

Definition in file g_data.h.

8.5 graph.h File Reference

```
#include <algobase.h>
#include <alloc.h>
#include <vgtl_graph.h>
```

Include dependency graph for graph.h:



8.5.1 Detailed Description

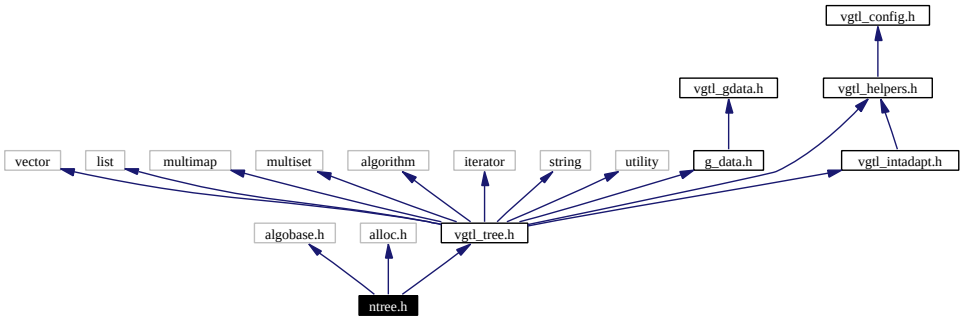
This is the external header file intended for direct use.

Definition in file graph.h.

8.6 ntree.h File Reference

```
#include <algobase.h>
#include <alloc.h>
#include <vgtl_tree.h>
```

Include dependency graph for ntree.h:



8.6.1 Detailed Description

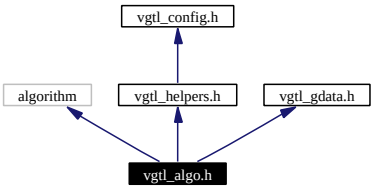
This is the external header file intended for direct use.

Definition in file ntree.h.

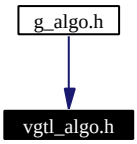
8.7 vgtl_algo.h File Reference

```
#include <algorithm>
#include <vgtl_helpers.h>
#include <vgtl_gdata.h>
```

Include dependency graph for vgtl_algo.h:



This graph shows which files directly or indirectly include this file:



Compounds

- **class __Child_data_iterator**

  iterator adapter for iterating through children data hooks.

- **class _Visitor**
- **class child_data_iterator**

  Iterator which iterates through the data hooks of all children.

Functions

- **template<class _IterativeWalker, class _Function> _Function walk (_IterativeWalker __first, _IterativeWalker __last, _Function __f)**
- **template<class _PrePostWalker, class _Function> _Function pre_post_walk (_PrePostWalker __first, _PrePostWalker __last, _Function __f)**
- **template<class _PrePostWalker, class _Function1, class _Function2> _Function2 pre_post_walk (_PrePostWalker __first, _PrePostWalker __last, _Function1 __f1, _Function2 __f2)**
- **template<class _PrePostWalker, class _Function> _Function var_walk (_PrePostWalker __first, _PrePostWalker __last, _Function __f)**
- **template<class _PrePostWalker, class _Function1, class _Function2> _Function2 var_walk (_PrePostWalker __first, _PrePostWalker __last, _Function1 __f1, _Function2 __f2)**
- **template<class _PrePostWalker, class _Function, class _Predicate> _Function walk_if (_PrePostWalker __first, _PrePostWalker __last, _Function __f, _Predicate __pred)**

- template<class _PrePostWalker, class _Function1, class _Function2, class _Predicate> _Function2 walk_if (_PrePostWalker __first, _PrePostWalker __last, _Function1 __f1, _Function2 __f2, _Predicate __pred)
- template<class _PrePostWalker, class _Function1, class _Function2, class _Predicate1, class _Predicate2> _Function2 walk_if (_PrePostWalker __first, _PrePostWalker __last, _Function1 __f1, _Function2 __f2, _Predicate1 __pred1, _Predicate2 __pred2)
- template<class _PrePostWalker, class _Function1, class _Function2, class _Predicate> _Function2 cached_walk_if (_PrePostWalker __first, _PrePostWalker __last, _Function1 __f1, _Function2 __f2, _Predicate __pred)
- template<class _PrePostWalker, class _Function1, class _Function2, class _Predicate> _Function2 multi_walk_if (_PrePostWalker __first, _PrePostWalker __last, _Function1 __f1, _Function2 __f2, _Predicate __pred)
- template<class _Walker, class _Function> _Function walk_up (_Walker __w, _Function __f)
- template<class _Walker, class _Function> _Function var_walk_up (_Walker __w, _Function __f)
- template<class _Walker, class _Function, class _Predicate> _Function walk_up_if (_Walker __w, _Function __f, _Predicate __p)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_preorder_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_preorder_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_postorder_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_postorder_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_preorder_walk_if (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_preorder_walk_if (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_preorder_walk_if (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_preorder_walk_if (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_postorder_walk_if (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_postorder_walk_if (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_walk_if (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_walk_if (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_cached_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_cached_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_multi_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_multi_walk (_Walker __w, _Visitor __f)

- template<class _Walker, class _Visitor, class _Predicate1, class _Predicate2> _Visitor::return_value recursive_walk_if (_Walker __w, _Visitor __f, _Predicate1 __p1, _Predicate2 __p2)
- template<class _Walker, class _Visitor, class _Predicate1, class _Predicate2> _Visitor::return_value _recursive_walk_if (_Walker __w, _Visitor __f, _Predicate1 __p1, _Predicate2 __p2)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_cached_walk (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_cached_walk (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_multi_walk (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_multi_walk (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_preorder_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_preorder_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_preorder_walk_up_if (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_preorder_walk_up_if (_Walker __w, _Visitor __f)
- ∗recursive walk towards the root node (i.e.come back!) template< class _Walker
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_postorder_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_postorder_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_postorder_walk_up_if (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_postorder_walk_up_if (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_walk_up_if (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_walk_up_if (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor, class _Predicate1, class _Predicate2> _Visitor::return_value recursive_walk_up_if (_Walker __w, _Visitor __f, _Predicate1 __p1, _Predicate2 __p2)
- template<class _Walker, class _Visitor, class _Predicate1, class _Predicate2> _Visitor::return_value _recursive_walk_up_if (_Walker __w, _Visitor __f, _Predicate1 __p1, _Predicate2 __p2)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_cached_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_cached_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_multi_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value _recursive_multi_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_cached_walk_up (_Walker __w, _Visitor __f, _Predicate __p)

- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_cached_walk_up (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value recursive_multi_walk_up (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor, class _Predicate> _Visitor::return_value _recursive_multi_walk_up (_Walker __w, _Visitor __f, _Predicate __p)
- template<class _Walker, class _Visitor> _Visitor::return_value general_directed_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value general_directed_walk_down (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value general_directed_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_general_directed_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_general_directed_walk_down (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_general_directed_walk_up (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value general_walk (_Walker __w, _Visitor __f)
- template<class _Walker, class _Visitor> _Visitor::return_value recursive_general_walk (_Walker __w, _Visitor __f)

Variables

- _Visitor _Walker
- _Visitor _Visitor
- _Visitor __w
- _Visitor __f
- _Visitor __it
- _Visitor __e

8.7.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

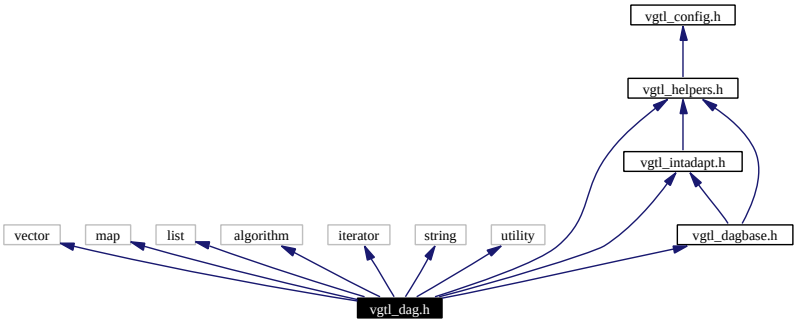
Definition in file vgtl_algo.h.

8.8 vgtl_dag.h File Reference

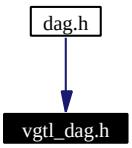
```
#include <vector>
#include <map>
#include <list>
#include <algorithm>
#include <iterator>
#include <string>
#include <utility>
```

```
#include <vgtl_helpers.h>
#include <vgtl_intadapt.h>
#include <vgtl_dagbase.h>
```

Include dependency graph for vgtl_dag.h:



This graph shows which files directly or indirectly include this file:



Compounds

- **class __DG**
  Directed graph base class.
- **class _DG_iterator**
  iterator through the directed graph.
- **class _DG_walker**
  recursive directed graph walkers.
- **class dag**
  unlabeled directed acyclic graph (DAG).
- **class dgraph**
  unlabeled directed graph.

8.8.1 Detailed Description

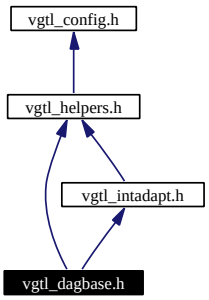
This is an internal header file, included by other library headers. You should not attempt to use it directly.

Definition in file vgtl_dag.h.

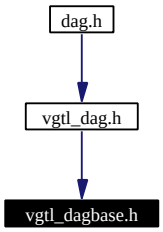
8.9 vgtl_dagbase.h File Reference

```
#include <vgtl_helpers.h>
#include <vgtl_intadapt.h>
```

Include dependency graph for vgtl_dagbase.h:



This graph shows which files directly or indirectly include this file:



Compounds

- **class _DG_alloc_base**
  Directed graph base class for general standard-conforming allocators.
- **class _DG_alloc_base< _Tp, _Ctr, _I, _Allocator, true >**
  Directed graph base class specialization for instanceless allocators.
- **class _DG_base**
  Directed graph base class for allocator encapsulation.
- **class _DG_node**
  directed graph node.

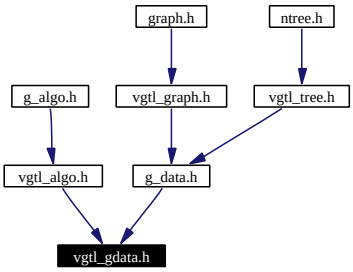
8.9.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

Definition in file vgtl_dagbase.h.

8.10 vgtl_gdata.h File Reference

This graph shows which files directly or indirectly include this file:



Compounds

- **union _Tree_data_hook**

Typedefs

- **typedef __VGTL_BEGIN_NAMESPACE union _Tree_data_hook ctree_data_hook**

8.10.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

Definition in file vgtl_gdata.h.

8.10.2 Typedef Documentation

8.10.2.1 typedef __VGTL_BEGIN_NAMESPACE union _Tree_data_hook ctree_data_hook

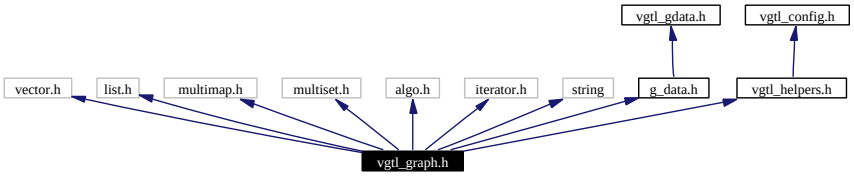
This is a mixed-type union for data hooks on trees. A data hook can be used for non-recursive walks.

8.11 vgtl_graph.h File Reference

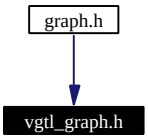
```
#include <vector.h>
#include <list.h>
#include <multimap.h>
#include <multiset.h>
#include <algo.h>
#include <iterator.h>
#include <string>
#include <g_data.h>
```

```
#include <vgtl_helpers.h>
```

Include dependency graph for vgtl_graph.h:



This graph shows which files directly or indirectly include this file:



Compounds

- **class __Tree**

  Tree base class without data hooks.

- **class _Graph_node**
- **class _Graph_walker**
- **class _Graph_walker_base**
- **class _RTree_walker**

  recursive tree walkers.

- **class _Tree_alloc_base**

  Tree base class for general standard-conforming allocators.

- **class _Tree_alloc_base< _Tp, _Ctr, _I, _Allocator, true >**
- **class _Tree_base**

  Tree base class for allocator encapsulation.

- **class _Tree_iterator**

  iterator through the tree.

- **class atree**

  n-ary forest with labelled edges.

- **class ntree**

  n-ary forest.

- **class stree**

  n-ary forest with unsorted edges.

8.11.1 Detailed Description

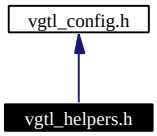
This is an internal header file, included by other library headers. You should not attempt to use it directly.

Definition in file vgtl_graph.h.

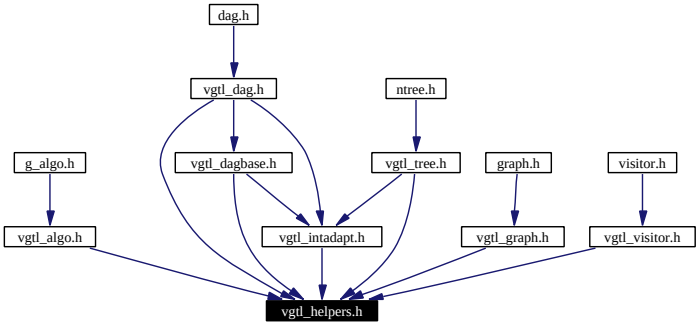
8.12 vgtl_helpers.h File Reference

```
#include <vgtl_config.h>
```

Include dependency graph for vgtl_helpers.h:



This graph shows which files directly or indirectly include this file:



Functions

- **template<class _BidirIter, class _Tp> _BidirIter rfind (_BidirIter __first, _BidirIter __last, const _Tp &__val, std::bidirectional_iterator_tag)**
- **template<class _BidirIter, class _Predicate> _BidirIter rfind_if (_BidirIter __first, _BidirIter __last, _Predicate __pred, std::bidirectional_iterator_tag)**
- **template<class _RandomAccessIter, class _Tp> _RandomAccessIter rfind (_RandomAccessIter __first, _RandomAccessIter __last, const _Tp &__val, std::random_access_iterator_tag)**
- **template<class _RandomAccessIter, class _Predicate> _RandomAccessIter rfind_if (_RandomAccessIter __first, _RandomAccessIter __last, _Predicate __pred, std::random_access_iterator_tag)**
- **template<class _BidirIter, class _Tp> _BidirIter rfind (_BidirIter __first, _BidirIter __last, const _Tp &__val)**
- **template<class _BidirIter, class _Predicate> _BidirIter rfind_if (_BidirIter __first, _BidirIter __last, _Predicate __pred)**

8.12.1 Detailed Description

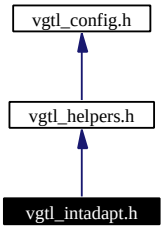
This is an internal header file, included by other library headers. You should not attempt to use it directly.

Definition in file vgtl_helpers.h.

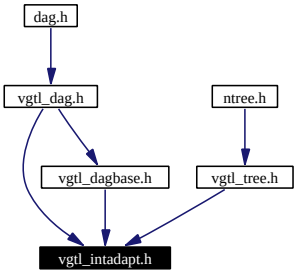
8.13 vgtl_intadapt.h File Reference

```
#include <vgtl_helpers.h>
```

Include dependency graph for vgtl_intadapt.h:



This graph shows which files directly or indirectly include this file:



Compounds

- **class __one_iterator**

  make an iterator out of one pointer.

- **class _G_compare_adaptor**

  Adaptor for data comparison in graph nodes.

- **class pair_adaptor**

  adaptor for an iterator over a pair to an iterator returning the second element.

- **class pointer_adaptor**

  adaptor transforming a comparison predicate to pointers.

8.13.1 Detailed Description

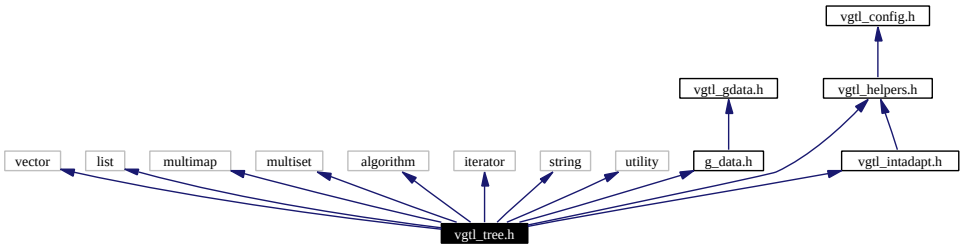
This is an internal header file, included by other library headers. You should not attempt to use it directly.

Definition in file vgtl_intadapt.h.

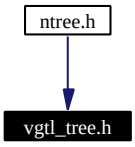
8.14 vgtl_tree.h File Reference

```
#include <vector>
#include <list>
#include <multimap>
#include <multiset>
#include <algorithm>
#include <iterator>
#include <string>
#include <utility>
#include <g_data.h>
#include <vgtl_helpers.h>
#include <vgtl_intadapt.h>
```

Include dependency graph for vgtl_tree.h:



This graph shows which files directly or indirectly include this file:



Compounds

- **class __ITree**

  Tree base class with data hooks.

- class __Tree
  Tree base class without data hooks.
- class __Tree_t
  Tree base class.
- class _RTree_walker
  recursive tree walkers.
- class _Tree_alloc_base
  Tree base class for general standard-conforming allocators.
- class _Tree_alloc_base< _Tp, _Ctr, _I, _Node, _Allocator, true >
  Tree base class specialization for instanceless allocators.
- class _Tree_base
  Tree base class for allocator encapsulation.
- class _Tree_iterator
  iterator through the tree.
- class _ITree_node
  tree node for trees with data hooks.
- class _Tree_node
  tree node for trees w/o data hooks.
- class _Tree_walker
  automatic tree walkers.
- class _Tree_walker_base
  base class for all tree walkers.
- class atree
  n*-ary forest with labelled edges.*
- class ntree
  n*-ary forest.*
- class ratree
  n*-ary forest with labelled edges.*
- class rntree
  n*-ary forest.*
- class rstree
  n*-ary forest with unsorted edges.*
- class stree
  n*-ary forest with unsorted edges.*

Defines

- **#define _C_W_preorder 1**
- **#define _C_W_postorder 2**

Enumerations

- **enum walker_type**

8.14.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

Definition in file vgtl_tree.h.

8.14.2 Define Documentation

8.14.2.1 #define _C_W_postorder 2

The walker is in postorder mode

Definition at line 46 of file vgtl_tree.h.

8.14.2.2 #define _C_W_preorder 1

The walker is in preorder mode

Definition at line 44 of file vgtl_tree.h.

8.14.3 Enumeration Type Documentation

8.14.3.1 enum walker_type

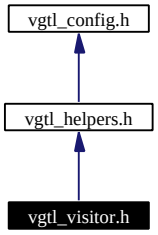
enum for walker types: preorder, postorder, pre+postorder

Definition at line 49 of file vgtl_tree.h.

8.15 vgtl_visitor.h File Reference

```
#include <vgtl_helpers.h>
```

Include dependency graph for vgtl_visitor.h:



This graph shows which files directly or indirectly include this file:

Compounds

- **class postorder_visitor**
  postorder visitor base class.
- **class preorder_visitor**
  preorder visitor base class.
- **class prepost_visitor**
  pre+postorder visitor base class.

8.15.1 Detailed Description

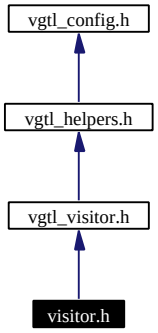
This is an internal header file, included by other library headers. You should not attempt to use it directly.

Definition in file vgtl_visitor.h.

8.16 visitor.h File Reference

`#include <vgtl_visitor.h>`

Include dependency graph for visitor.h:



8.16.1 Detailed Description

This is the external header file intended for direct use.

Definition in file visitor.h.

Index