VariantDBSCAN

Generated by Doxygen 1.8.6

Tue Feb 28 2017 15:52:38

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

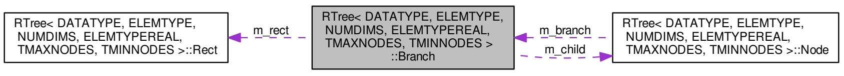
Chapter 3

Data Structure Documentation

3.1 RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Branch Struct Reference

`#include "RTree.h"`

Collaboration diagram for RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Branch:



RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect
m_rect
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > ::Branch
m_branch
m_child
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node

Data Fields

- Rect m_rect

  Bounds.
- Node ∗ m_child

  Child node.
- DATATYPE m_data

  Data Id or Ptr.

3.1.1 Detailed Description

template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2>struct RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Branch

May be data or may be another subtree The parents level determines this. If the parents level is 0, then this is data

3.1.2 Field Documentation

3.1.2.1 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> Node* RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Branch::m_child

Child node.

3.1.2.2 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> DATATYPE RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Branch::m_data

Data Id or Ptr.

3.1.2.3 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> Rect RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Branch::m_rect

Bounds.

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

- RTree.h

3.2 dataElem Struct Reference

```
#include "structs.h"
```

Data Fields

- double x
- double y

3.2.1 Detailed Description

2-D data struct

3.2.2 Field Documentation

3.2.2.1 double dataElem::x

3.2.2.2 double dataElem::y

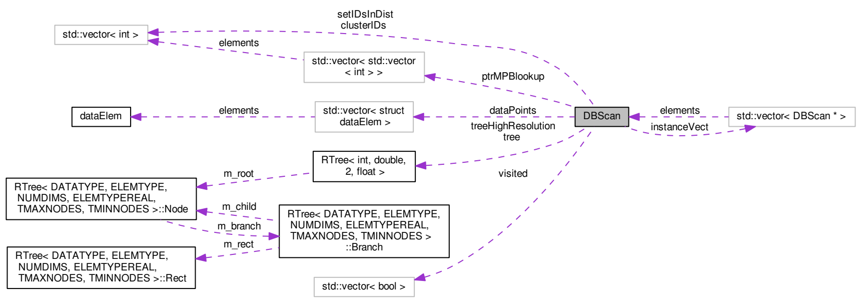
The documentation for this struct was generated from the following file:

- structs.h

3.3 DBScan Class Reference

```
#include "DBScan.h"
```

Collaboration diagram for DBScan:


std::vector< int >
setIDsInDist
clusterIDs
elements
std::vector< std::vector < int > >
ptrMPBlookup
dataElem
elements
std::vector< struct dataElem >
dataPoints
DBScan
elements
instanceVect
std::vector< DBScan * >
treeHighResolution tree
RTree< int, double, 2, float >
visited
m_root
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node
m_child
m_branch
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > ::Branch
m_rect
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect
std::vector< bool >

Public Member Functions

- DBScan (std::vector< struct dataElem > ∗ptrData, double epsilon, int minimumPts, RTree< int, double, 2, float > ∗indexPtr, std::vector< std::vector< int > > ∗ptr_MPB_lookup, RTree< int, double, 2, float > ∗high-ResIndex)
- void algDBScanParallel ()
- void algDBScanParallelReuseClusterResults ()
- void generateMBBAroundCluster (std::vector< int > ∗clusterPoints, double ∗MBB_min, double ∗MBB_max)
- void appendMBBByEpsilon (double ∗MBB_min, double ∗MBB_max, double eps)
- void DBScanParallelMPBSTUMP (int finishedInstanceID, bool ∗destroyedArr, std::vector< int > ∗candidates-ToGrowFrom)
- void assignPointsToPredefinedCluster (int finishedInstanceID)
- void setClusterScheduleDensity (std::vector< int > ∗schedule, std::vector< int > clusterArr[ ], int num-ClustersInOtherInstance)
- void scheduleSelector (std::vector< int > ∗schedule, std::vector< int > clusterArr[ ], int numClustersInOther-Instance)
- int getDBScanNumClusters ()

Static Public Member Functions

- static bool comparedensityStructfn (const densityStruct &a, const densityStruct &b)

Data Fields

- std::vector< int > clusterIDs
- std::vector< DBScan ∗ > instanceVect
- int numClustersForStats

Private Member Functions

- void initializeVisitedPoints (int size)
- void generateMBB (struct dataElem ∗point, double distance, double ∗MBB_min, double ∗MBB_max)
- bool generateMBBNormal (struct dataElem ∗point, double distance, double ∗MBB_min, double ∗MBB_max)
- int filterCandidatesMPB (struct dataElem ∗point, std::vector< int > ∗candidateSet, double distance, std-::vector< int > ∗setIDsInDistPtr)
- void getNeighboursParallelMPB (struct dataElem ∗point, double distance, std::vector< int > ∗setIDsInDist-Ptr)
- double EuclidianDistance (struct dataElem ∗point1, struct dataElem ∗point2)
- void copyVect (std::vector< int > ∗dest, std::vector< int > ∗source)
- void initializeClusterIDs (int size)

Private Attributes

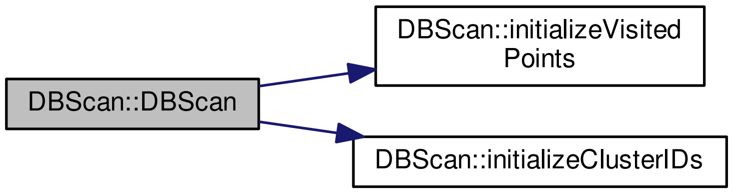
- double distance
- int minPts
- int clusterCnt
- std::vector< bool > visited
- std::vector< struct dataElem > ∗ dataPoints
- std::vector< std::vector< int > > ∗ ptrMPBlookup
- std::vector< int > setIDsInDist
- RTree< int, double, 2, float > ∗ tree
- RTree< int, double, 2, float > ∗ treeHighResolution

3.3.1 Constructor & Destructor Documentation

3.3.1.1 DBScan::DBScan (std::vector< struct **dataElem** > ∗ *ptrData,* double *epsilon,* int *minimumPts,* **RTree**< int, double, 2, float > ∗ *indexPtr,* std::vector< std::vector< int > > ∗ *ptr_MPB_lookup,* **RTree**< int, double, 2, float > ∗ *highResIndex*)

Constructor for the implementation that reuses results from one variant to another variant: Input: pointer to the data elements (∗ptrData) the distance: epsilon the minimum number of points to form a cluster: minimumPts a pointer to the R-tree index: ∗indexPtr a lookup array to the data elements stored in each MBB of the index: ∗ptr_MPB_lookup The high resolution R-tree that's used when "growing the cluster" when reusing data: ∗highResIndex

Here is the call graph for this function:

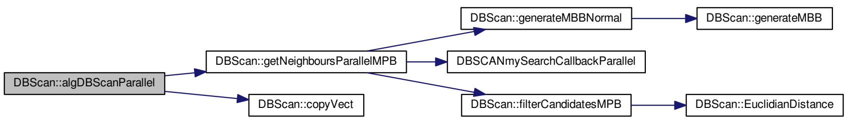

DBScan::DBScan
DBScan::initializeVisited Points
DBScan::initializeClusterIDs

3.3.2 Member Function Documentation

3.3.2.1 void DBScan::algDBScanParallel ( )

Allows clustering in parallel of multiple variants Uses a single R-tree index, but separate buffers for each thread

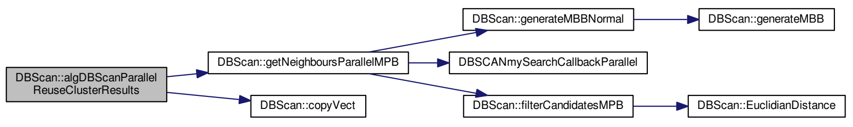
Here is the call graph for this function:


DBScan::algDBScanParallel
DBScan::getNeighboursParallelMPB
DBScan::copyVect
DBScan::generateMBBNormal
DBScan::generateMBB
DBSCANmySearchCallbackParallel
DBScan::filterCandidatesMPB
DBScan::EuclidianDistance

3.3.2.2 void DBScan::algDBScanParallelReuseClusterResults ( )

DBSCAN version that takes the clustered output from one instance and then reuses it for another instance

Here is the call graph for this function:


DBScan::algDBScanParallel ReuseClusterResults
DBScan::getNeighboursParallelMPB
DBScan::copyVect
DBScan::generateMBBNormal
DBSCANmySearchCallbackParallel
DBScan::filterCandidatesMPB
DBScan::generateMBB
DBScan::EuclidianDistance

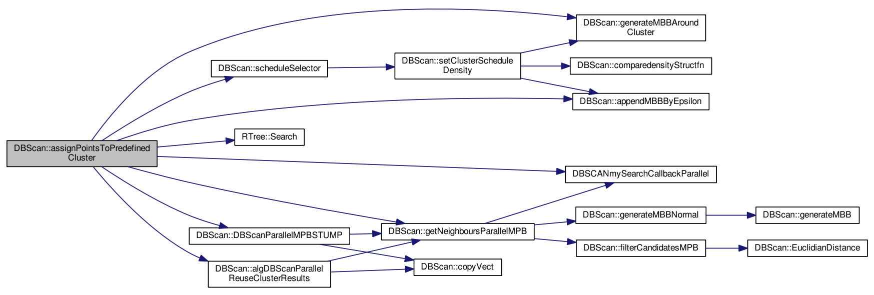
3.3.2.3 void DBScan::appendMBBByEpsilon (double ∗ *MBB_min,* double ∗ *MBB_max,* double *eps*)

Method that appends an MBB by epsilon

3.3.2.4 void DBScan::assignPointsToPredefinedCluster (int *finishedInstanceID*)

Method that builds new clusters from previously generated clusters

Here is the call graph for this function:


DBScan::assignPointsToPredefined Cluster
DBScan::scheduleSelector
DBScan::setClusterSchedule Density
DBScan::generateMBBAround Cluster
DBScan::comparedensityStructfn
DBScan::appendMBBByEpsilon
RTree::Search
DBSCANmySearchCallbackParallel
DBScan::DBScanParallelMPBSTUMP
DBScan::getNeighboursParallelMPB
DBScan::generateMBBNormal
DBScan::generateMBB
DBScan::filterCandidatesMPB
DBScan::EuclidianDistance
DBScan::algDBScanParallel ReuseClusterResults
DBScan::copyVect

3.3.2.5 bool DBScan::comparedensityStructfn (const **densityStruct** & *a,* const **densityStruct** & *b*) `[static]`

Comparison function

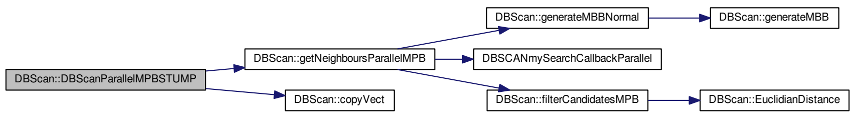
3.3.2.6 void DBScan::copyVect (std::vector< int > ∗ *dest,* std::vector< int > ∗ *source*) `[private]`

copies the contents from the source vector and appends them to the dest vector

3.3.2.7 void DBScan::DBScanParallelMPBSTUMP (int *finishedInstanceID,* bool ∗ *destroyedArr,* std::vector< int > ∗ *candidatesToGrowFrom*)

"Stump" of DBScan used when assigning points to predefined clusters just the part of the algorithm that expands the eps-neighbourhood

Here is the call graph for this function:


DBScan::DBScanParallelMPBSTUMP
DBScan::getNeighboursParallelMPB
DBScan::copyVect
DBScan::generateMBBNormal
DBSCANmySearchCallbackParallel
DBScan::filterCandidatesMPB
DBScan::generateMBB
DBScan::EuclidianDistance

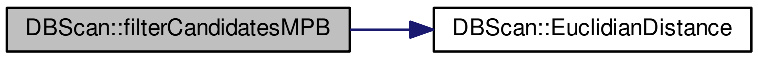
3.3.2.8 double DBScan::EuclidianDistance (struct **dataElem** $*$ *point1*, struct **dataElem** $*$ *point2*) `[private]`

Euclidian distance calculation between two points to filter the candidate set

3.3.2.9 int DBScan::filterCandidatesMPB (struct **dataElem** $*$ *point*, std::vector$<$ int $>$ $*$ *candidateSet*, double *distance*, std::vector$<$ int $>$ $*$ *setIDsInDistPtr*) `[private]`

Filter candidate points from the index search

Here is the call graph for this function:



3.3.2.10 void DBScan::generateMBB (struct **dataElem** $*$ *point*, double *distance*, double $*$ *MBB_min*, double $*$ *MBB_max*) `[private]`

MBB generation. The "Normal" refers to when we had the periodic boundary condition for the datasets and we did not need to wrap around generate a query MBB around the point to search for the values

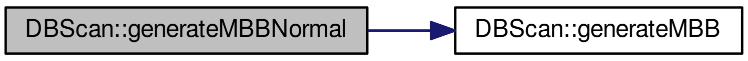
3.3.2.11 void DBScan::generateMBBAroundCluster (std::vector$<$ int $>$ $*$ *clusterPoints*, double $*$ *MBB_min*, double $*$ *MBB_max*)

Method that generates an MBB around a cluster for reusing data between variants

3.3.2.12 bool DBScan::generateMBBNormal (struct **dataElem** $*$ *point*, double *distance*, double $*$ *MBB_min*, double $*$ *MBB_max*) `[private]`

MBB generation. The "Normal" refers to when we had the periodic boundary condition for the datasets and we did not need to wrap around generate a query MBB around the point to search for the values

Here is the call graph for this function:



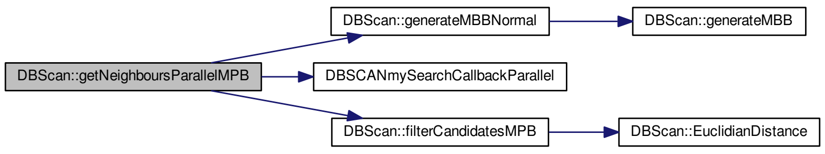
3.3.2.13 int DBScan::getDBScanNumClusters ( )

Gets the number of clusters generated by a DBScan instance for reusing data

3.3.2.14 void DBScan::getNeighboursParallelMPB (struct **dataElem** * *point*, double *distance*, std::vector< int > * *setIDsInDistPtr*) [private]

Epsilon-neighborhood search

Here is the call graph for this function:


DBScan::getNeighboursParallelMPB
DBScan::generateMBBNormal
DBScan::generateMBB
DBSCANmySearchCallbackParallel
DBScan::filterCandidatesMPB
DBScan::EuclidianDistance

3.3.2.15 void DBScan::initializeClusterIDs (int *size*) [private]

initializes the vector storing the IDs of the cluster of the data points

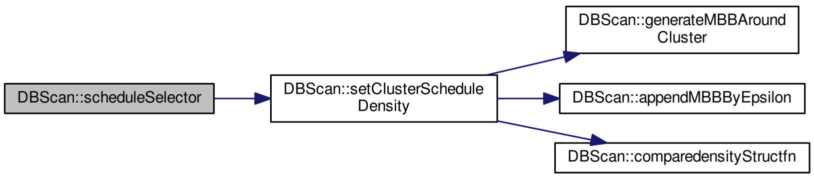
3.3.2.16 void DBScan::initializeVisitedPoints (int *size*) [private]

initialize all of the points to initially not be visited

3.3.2.17 void DBScan::scheduleSelector (std::vector< int > * *schedule*, std::vector< int > *clusterArr[ ]*, int *numClustersInOtherInstance*)

Selects the cluster reuse criteria ClusDensity in the paper only

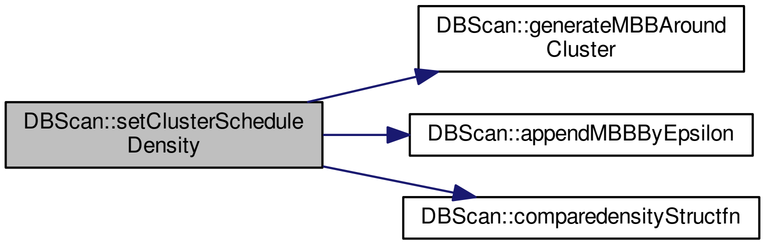
Here is the call graph for this function:


DBScan::scheduleSelector
DBScan::setClusterSchedule Density
DBScan::generateMBBAround Cluster
DBScan::appendMBBByEpsilon
DBScan::comparedensityStructfn

3.3.2.18 void DBScan::setClusterScheduleDensity (std::vector$<$ int $>$ $*$ *schedule,* std::vector$<$ int $>$ *clusterArr[ ],* int *numClustersInOtherInstance*)

Schedule of cluster reuse processing based on density (ClusDensity in the paper)

Here is the call graph for this function:


DBScan::setClusterSchedule Density
DBScan::generateMBBAround Cluster
DBScan::appendMBBByEpsilon
DBScan::comparedensityStructfn

3.3.3 Field Documentation

3.3.3.1 int DBScan::clusterCnt [private]

the number of clusters found after calling the algorithm

3.3.3.2 std::vector$<$int$>$ DBScan::clusterIDs

Vector that keeps track of the assignment of the points to a cluster. Cluster 0 means a noise point. The indices of the vector correspond to the data points. Element i in clusterIDs corresponds to the cluster that data element i is in within the dataPoints struct.

3.3.3.3 std::vector$<$struct dataElem$>$$*$ DBScan::dataPoints [private]

pointer to the data elements

3.3.3.4 double DBScan::distance [private]

DBSCAN Epsilon parameter

3.3.3.5 std::vector<DBScan ∗> DBScan::instanceVect

Vector of pointers to the DBScan instances that may have points for data reuse

3.3.3.6 int DBScan::minPts `[private]`

DBSCAN MinPts parameter

3.3.3.7 int DBScan::numClustersForStats

The number of clusters created (for statistics)

3.3.3.8 std::vector<std::vector<int> >∗ DBScan::ptrMPBlookup `[private]`

pointer to the lookup array for multiple pointers per MBB (MPB)

3.3.3.9 std::vector<int> DBScan::setIDsInDist `[private]`

temporary vector used to store the ids of the candidates that are actually within the threshold distance

3.3.3.10 RTree<int,double,2,float>∗ DBScan::tree `[private]`

pointer to the R-tree index

3.3.3.11 RTree<int,double,2,float>∗ DBScan::treeHighResolution `[private]`

pointer to the R-tree index that has 1 point per box when results can be reused across clustering runs

3.3.3.12 std::vector<bool> DBScan::visited `[private]`

vector that keeps track of the points that have beeen visited

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

- DBScan.h
- DBScan.cpp

3.4 densityStruct Struct Reference

```
#include "structs.h"
```

Data Fields

- int clusterID
- double density

3.4.1 Detailed Description

Struct used to order the clusters that will get reused

3.4.2 Field Documentation

3.4.2.1 int densityStruct::clusterID

3.4.2.2 double densityStruct::density

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

- structs.h

3.5 experiment Struct Reference

```
#include "structs.h"
```

Data Fields

- double epsilon
- int minpts
- unsigned int variantID

3.5.1 Detailed Description

Struct that outlines the parameters for multiple instances of DBScan, each one described per struct.

3.5.2 Field Documentation

3.5.2.1 double experiment::epsilon

3.5.2.2 int experiment::minpts

3.5.2.3 unsigned int experiment::variantID

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

- structs.h

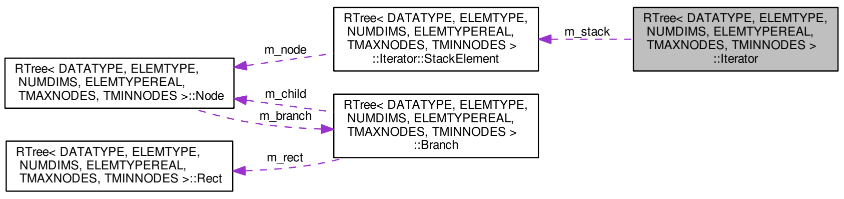
3.6 RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator Class Reference

Iterator is not remove safe.

```
#include "RTree.h"
```

Collaboration diagram for RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMIN-

NODES >::Iterator:


RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > ::Iterator::StackElement
m_stack
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > ::Iterator
m_node
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node
m_child
m_branch
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > ::Branch
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect
m_rect

Data Structures

- struct StackElement

Public Member Functions

- Iterator ()
- ∼Iterator ()
- bool IsNull ()

  Is iterator invalid.
- bool IsNotNull ()

  Is iterator pointing to valid data.
- DATATYPE & operator∗ ()

  Access the current data element. Caller must be sure iterator is not NULL first.
- const DATATYPE & operator∗ () const

  Access the current data element. Caller must be sure iterator is not NULL first.
- bool operator++ ()

  Find the next data element.
- void GetData (DATATYPE ∗t_id, DATATYPE ∗t_segment_id)
- void GetBounds (ELEMTYPE a_min[NUMDIMS], ELEMTYPE a_max[NUMDIMS])

  Get the bounds for this node.

Private Types

- enum { MAX_STACK = 32 }

Private Member Functions

- void Init ()

  Reset iterator.
- bool FindNextData ()

  Find the next data element in the tree (For internal use only)
- void Push (Node ∗a_node, int a_branchIndex)

  Push node and branch onto iteration stack (For internal use only)
- StackElement & Pop ()

  Pop element off iteration stack (For internal use only)

Private Attributes

- StackElement m_stack [MAX_STACK]

    Stack as we are doing iteration instead of recursion.
- int m_tos

    Top Of Stack index.

Friends

- class RTree

3.6.1 Detailed Description

template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int T-MINNODES = TMAXNODES / 2>class RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator

Iterator is not remove safe.

3.6.2 Member Enumeration Documentation

3.6.2.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> anonymous enum `[private]`

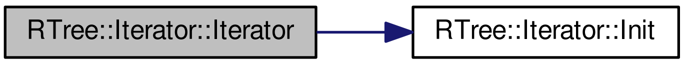
Enumerator

MAX_STACK

3.6.3 Constructor & Destructor Documentation

3.6.3.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::Iterator ( ) `[inline]`

Here is the call graph for this function:



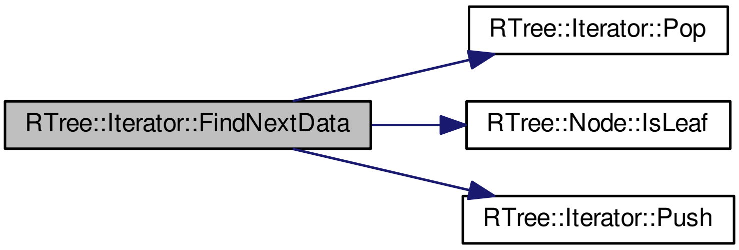
3.6.3.2 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::∼Iterator ( ) `[inline]`

3.6.4 Member Function Documentation

3.6.4.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::FindNextData ( ) [inline], [private]

Find the next data element in the tree (For internal use only)

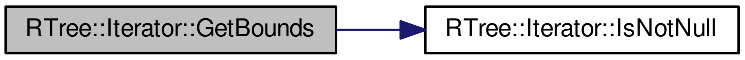
Here is the call graph for this function:



3.6.4.2 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::GetBounds (ELEMTYPE *a_min[NUMDIMS]*, ELEMTYPE *a_max[NUMDIMS]*) [inline]

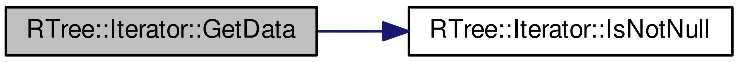
Get the bounds for this node.

Here is the call graph for this function:



3.6.4.3 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::GetData (DATATYPE ∗ *t_id*, DATATYPE ∗ *t_segment_id*) [inline]

Here is the call graph for this function:



3.6.4.4 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::Init ( ) [inline], [private]

Reset iterator.

3.6.4.5 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::IsNotNull ( ) [inline]

Is iterator pointing to valid data.

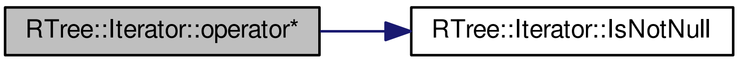
3.6.4.6 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::IsNull ( ) [inline]

Is iterator invalid.

3.6.4.7 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> DATATYPE& RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::operator* ( ) [inline]

Access the current data element. Caller must be sure iterator is not NULL first.

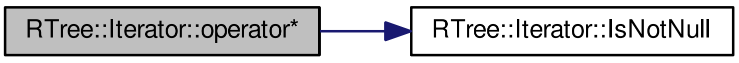
Here is the call graph for this function:



3.6.4.8 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> const DATATYPE& RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::operator* ( ) const [inline]

Access the current data element. Caller must be sure iterator is not NULL first.

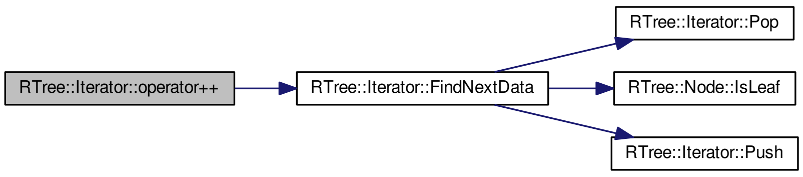
Here is the call graph for this function:



3.6.4.9 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::operator++ ( ) [inline]

Find the next data element.

Here is the call graph for this function:



3.6.4.10 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> StackElement& RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::Pop ( ) [inline], [private]

Pop element off iteration stack (For internal use only)

3.6.4.11 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::Push (Node ∗ *a_node,* int *a_branchIndex*) [inline], [private]

Push node and branch onto iteration stack (For internal use only)

3.6.5 Friends And Related Function Documentation

3.6.5.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> friend class RTree [friend]

3.6.6 Field Documentation

3.6.6.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> StackElement RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::m_stack[MAX_STACK] [private]

Stack as we are doing iteration instead of recursion.

3.6.6.2 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> int RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::m_tos [private]

Top Of Stack index.

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

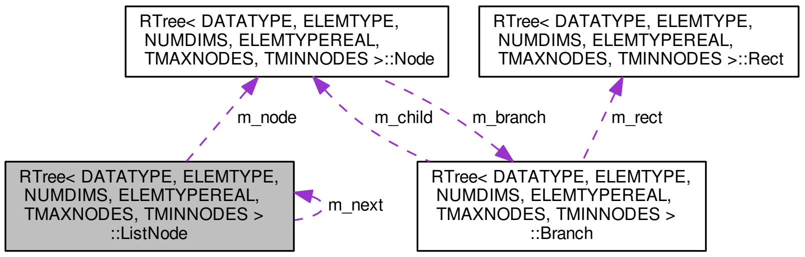
- RTree.h

3.7 RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::ListNode Struct Reference

A link list of nodes for reinsertion after a delete operation.

```
#include "RTree.h"
```

Collaboration diagram for RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::ListNode:


RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect
m_node
m_child
m_branch
m_rect
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > ::ListNode
m_next
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > ::Branch

Data Fields

- ListNode ∗ m_next

  Next in list.
- Node ∗ m_node

  Node.

3.7.1 Detailed Description

template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2>struct RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::ListNode

A link list of nodes for reinsertion after a delete operation.

3.7.2 Field Documentation

3.7.2.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> ListNode∗ RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::ListNode::m_next

Next in list.

3.7.2.2 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> Node∗ RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::ListNode::m_node

Node.

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

- RTree.h

3.8 MPBRect Struct Reference

`#include "structs.h"`

Public Member Functions

- void CreateMBB (std::vector< struct dataElem > ∗dataPoints, std::vector< int > ∗multiplePointsToIndex)

Data Fields

- double MBB_min [2]
- double MBB_max [2]
- int pid

3.8.1 Detailed Description

Used for the index of point objects - multiple points per MBB. They make the MBBs that are inserted into the tree (2-D MBBs only)

3.8.2 Member Function Documentation

3.8.2.1 void MPBRect::CreateMBB (std::vector< struct dataElem > ∗ *dataPoints,* std::vector< int > ∗ *multiplePointsToIndex*) `[inline]`

3.8.3 Field Documentation

3.8.3.1 double MPBRect::MBB_max[2]

3.8.3.2 double MPBRect::MBB_min[2]

3.8.3.3 int MPBRect::pid

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

- structs.h

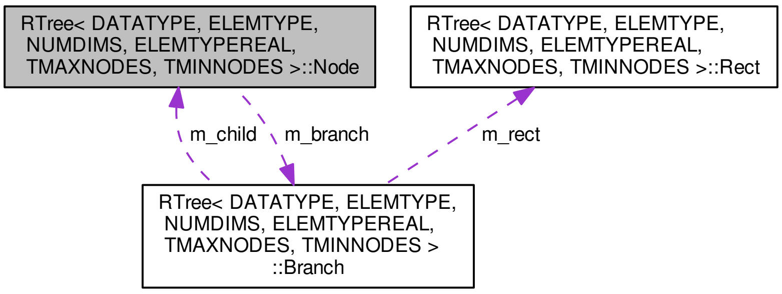
3.9 RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node Struct Reference

Node for each branch level.

`#include "RTree.h"`

Collaboration diagram for RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMIN-

NODES >::Node:


RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect
m_child
m_branch
m_rect
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > ::Branch

Public Member Functions

- bool IsInternalNode ()
- bool IsLeaf ()

Data Fields

- int m_count

  Count.
- int m_level

  Leaf is zero, others positive.
- Branch m_branch [MAXNODES]

  Branch.

3.9.1 Detailed Description

template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2>struct RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node

Node for each branch level.

3.9.2 Member Function Documentation

3.9.2.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node::IsInternalNode ( ) `[inline]`

3.9.2.2 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node::IsLeaf ( ) `[inline]`

3.9.3 Field Documentation

3.9.3.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> Branch RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node::m_branch[MAXNODES]

Branch.

3.9.3.2 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> int RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node::m_count

Count.

3.9.3.3 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> int RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node::m_level

Leaf is zero, others positive.

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

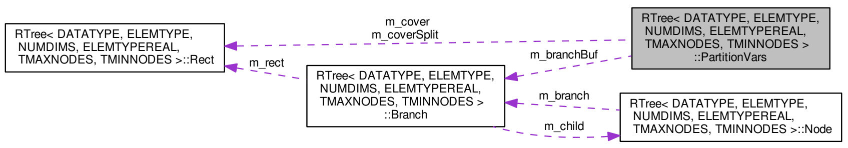
- RTree.h

3.10 RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PartitionVars Struct Reference

Variables for finding a split partition.

```
#include "RTree.h"
```

Collaboration diagram for RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PartitionVars:


RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect
m_cover
m_coverSplit
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > ::PartitionVars
m_rect
m_branchBuf
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > ::Branch
m_branch
m_child
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node

Public Types

- enum { NOT_TAKEN = -1 }

Data Fields

- int m_partition [MAXNODES+1]
- int m_total
- int m_minFill
- int m_count [2]
- Rect m_cover [2]
- ELEMTYPEREAL m_area [2]
- Branch m_branchBuf [MAXNODES+1]

- int m_branchCount
- Rect m_coverSplit
- ELEMTYPEREAL m_coverSplitArea

3.10.1 Detailed Description

template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2>struct RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PartitionVars

Variables for finding a split partition.

3.10.2 Member Enumeration Documentation

3.10.2.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> anonymous enum

Enumerator

NOT_TAKEN

3.10.3 Field Documentation

3.10.3.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> ELEMTYPEREAL RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PartitionVars::m_area[2]

3.10.3.2 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> Branch RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PartitionVars::m_branchBuf[MAXNODES+1]

3.10.3.3 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> int RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PartitionVars::m_branchCount

3.10.3.4 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> int RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PartitionVars::m_count[2]

3.10.3.5 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> Rect RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PartitionVars::m_cover[2]

3.10.3.6 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> Rect RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PartitionVars::m_coverSplit

3.10.3.7 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> ELEMTYPEREAL RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PartitionVars::m_coverSplitArea

3.10.3.8 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> int RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PartitionVars::m_minFill

3.10.3.9 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> int RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PartitionVars::m_partition[MAXNODES+1]

3.10.3.10 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> int RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PartitionVars::m_total

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

- RTree.h

3.11 QueryRect Struct Reference

```
#include "structs.h"
```

Public Member Functions

- QueryRect ()
- void CreateMBB ()

Data Fields

- double P1 [2]
- double P2 [2]
- double MBB_min [2]
- double MBB_max [2]

3.11.1 Detailed Description

MBB for querying the R-tree

3.11.2 Constructor & Destructor Documentation

3.11.2.1 QueryRect::QueryRect ( ) [inline]

3.11.3 Member Function Documentation

3.11.3.1 void QueryRect::CreateMBB ( ) [inline]

3.11.4 Field Documentation

3.11.4.1 double QueryRect::MBB_max[2]

3.11.4.2 double QueryRect::MBB_min[2]

3.11.4.3 double QueryRect::P1[2]

3.11.4.4 double QueryRect::P2[2]

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

- structs.h

3.12 RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect Struct Reference

Minimal bounding rectangle (n-dimensional)

```
#include "RTree.h"
```

Data Fields

- ELEMTYPE m_min [NUMDIMS]

  Min dimensions of bounding box.
- ELEMTYPE m_max [NUMDIMS]

  Max dimensions of bounding box.
- DATATYPE m_traj_id
- DATATYPE m_segment_id

  mine: FOR TRAJECTORY ID

3.12.1 Detailed Description

template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2>struct RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect

Minimal bounding rectangle (n-dimensional)

3.12.2 Field Documentation

3.12.2.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> ELEMTYPE RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect::m_max[NUMDIMS]

Max dimensions of bounding box.

3.12.2.2 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> ELEMTYPE RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect::m_min[NUMDIMS]

Min dimensions of bounding box.

3.12.2.3 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> DATATYPE RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect::m_segment_id

mine: FOR TRAJECTORY ID

3.12.2.4 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> DATATYPE RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect::m_traj_id

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

- RTree.h

3.13 Rect Struct Reference

```
#include "structs.h"
```

Public Member Functions

- Rect ()
- void CreateMBB ()

Data Fields

- double P1 [2]
- double MBB_min [2]
- double MBB_max [2]
- int pid

3.13.1 Detailed Description

Used for the index of point objects. They make the MBBs that are inserted into the tree (2-D MBBs)

3.13.2 Constructor & Destructor Documentation

3.13.2.1 Rect::Rect ( ) `[inline]`

3.13.3 Member Function Documentation

3.13.3.1 void Rect::CreateMBB ( ) `[inline]`

3.13.4 Field Documentation

3.13.4.1 double Rect::MBB_max[2]

3.13.4.2 double Rect::MBB_min[2]

3.13.4.3 double Rect::P1[2]

3.13.4.4 int Rect::pid

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

- structs.h

3.14 RTFileStream Class Reference

```
#include "RTree.h"
```

Public Member Functions

- RTFileStream ()
- ∼RTFileStream ()

- bool OpenRead (const char ∗a_fileName)
- bool OpenWrite (const char ∗a_fileName)
- void Close ()
- template<typename TYPE >
  size_t Write (const TYPE &a_value)
- template<typename TYPE >
  size_t WriteArray (const TYPE ∗a_array, int a_count)
- template<typename TYPE >
  size_t Read (TYPE &a_value)
- template<typename TYPE >
  size_t ReadArray (TYPE ∗a_array, int a_count)

Private Attributes

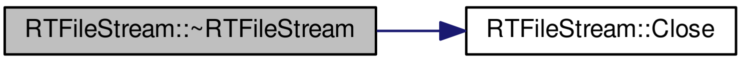
- FILE ∗ m_file

3.14.1 Constructor & Destructor Documentation

3.14.1.1 RTFileStream::RTFileStream ( ) [inline]

3.14.1.2 RTFileStream::∼RTFileStream ( ) [inline]

Here is the call graph for this function:



3.14.2 Member Function Documentation

3.14.2.1 void RTFileStream::Close ( ) [inline]

3.14.2.2 bool RTFileStream::OpenRead (const char ∗ *a_fileName*) [inline]

3.14.2.3 bool RTFileStream::OpenWrite (const char ∗ *a_fileName*) [inline]

3.14.2.4 template<typename TYPE > size_t RTFileStream::Read (TYPE & *a_value*) [inline]

3.14.2.5 template<typename TYPE > size_t RTFileStream::ReadArray (TYPE ∗ *a_array*, int *a_count*) [inline]

3.14.2.6 template<typename TYPE > size_t RTFileStream::Write (const TYPE & *a_value*) [inline]

3.14.2.7 template<typename TYPE > size_t RTFileStream::WriteArray (const TYPE ∗ *a_array*, int *a_count*) [inline]

3.14.3 Field Documentation

3.14.3.1 FILE∗ RTFileStream::m_file [private]

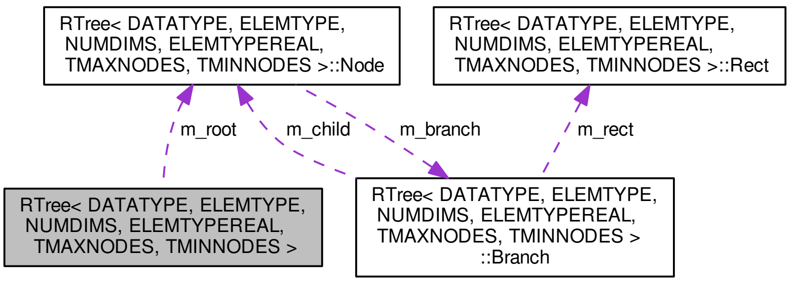
The documentation for this class was generated from the following file:

- RTree.h

3.15 RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > Class Template Reference

`#include "RTree.h"`

Collaboration diagram for RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >:


RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect
m_root
m_child
m_branch
m_rect
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > ::Branch

Data Structures

- struct Branch
- class Iterator

  Iterator is not remove safe.
- struct ListNode

  A link list of nodes for reinsertion after a delete operation.
- struct Node

  Node for each branch level.
- struct PartitionVars

  Variables for finding a split partition.
- struct Rect

  Minimal bounding rectangle (n-dimensional)

Public Types

- enum { MAXNODES = TMAXNODES, MINNODES = TMINNODES }
- typedef bool(∗ t_resultCallback)(DATATYPE, void ∗)

Public Member Functions

- RTree ()
- virtual ∼RTree ()
- void Insert (const ELEMTYPE a_min[NUMDIMS], const ELEMTYPE a_max[NUMDIMS], const DATATYPE &a_dataId)
- void Remove (const ELEMTYPE a_min[NUMDIMS], const ELEMTYPE a_max[NUMDIMS], const DATATYPE &a_dataId)

- int Search (const ELEMTYPE a_min[NUMDIMS], const ELEMTYPE a_max[NUMDIMS], t_resultCallback a-_resultCallback, void ∗a_context)
- void RemoveAll ()

  Remove all entries from tree.
- int Count ()

  Count the data elements in this container. This is slow as no internal counter is maintained.
- bool Load (const char ∗a_fileName)

  Load tree contents from file.
- bool Load (RTFileStream &a_stream)

  Load tree contents from stream.
- bool Save (const char ∗a_fileName)

  Save tree contents to file.
- bool Save (RTFileStream &a_stream)

  Save tree contents to stream.
- void GetFirst (Iterator &a_it)

  Get 'first' for iteration.
- void GetNext (Iterator &a_it)

  Get Next for iteration.
- bool IsNull (Iterator &a_it)

  Is iterator NULL, or at end?
- DATATYPE & GetAt (Iterator &a_it)

  Get object at iterator position.

Protected Member Functions

- Node ∗ AllocNode ()
- void FreeNode (Node ∗a_node)
- void InitNode (Node ∗a_node)
- void InitRect (Rect ∗a_rect)
- bool InsertRectRec (const Branch &a_branch, Node ∗a_node, Node ∗∗a_newNode, int a_level)
- bool InsertRect (const Branch &a_branch, Node ∗∗a_root, int a_level)
- Rect NodeCover (Node ∗a_node)
- bool AddBranch (const Branch ∗a_branch, Node ∗a_node, Node ∗∗a_newNode)
- void DisconnectBranch (Node ∗a_node, int a_index)
- int PickBranch (const Rect ∗a_rect, Node ∗a_node)
- Rect CombineRect (const Rect ∗a_rectA, const Rect ∗a_rectB)
- void SplitNode (Node ∗a_node, const Branch ∗a_branch, Node ∗∗a_newNode)
- ELEMTYPEREAL RectSphericalVolume (Rect ∗a_rect)
- ELEMTYPEREAL RectVolume (Rect ∗a_rect)
- ELEMTYPEREAL CalcRectVolume (Rect ∗a_rect)
- void GetBranches (Node ∗a_node, const Branch ∗a_branch, PartitionVars ∗a_parVars)
- void ChoosePartition (PartitionVars ∗a_parVars, int a_minFill)
- void LoadNodes (Node ∗a_nodeA, Node ∗a_nodeB, PartitionVars ∗a_parVars)
- void InitParVars (PartitionVars ∗a_parVars, int a_maxRects, int a_minFill)
- void PickSeeds (PartitionVars ∗a_parVars)
- void Classify (int a_index, int a_group, PartitionVars ∗a_parVars)
- bool RemoveRect (Rect ∗a_rect, const DATATYPE &a_id, Node ∗∗a_root)
- bool RemoveRectRec (Rect ∗a_rect, const DATATYPE &a_id, Node ∗a_node, ListNode ∗∗a_listNode)
- ListNode ∗ AllocListNode ()
- void FreeListNode (ListNode ∗a_listNode)
- bool Overlap (Rect ∗a_rectA, Rect ∗a_rectB)
- void ReInsert (Node ∗a_node, ListNode ∗∗a_listNode)

- bool Search (Node ∗a_node, Rect ∗a_rect, int &a_foundCount, t_resultCallback a_resultCallback, void ∗a_-context)
- void RemoveAllRec (Node ∗a_node)
- void Reset ()
- void CountRec (Node ∗a_node, int &a_count)
- bool SaveRec (Node ∗a_node, RTFileStream &a_stream)
- bool LoadRec (Node ∗a_node, RTFileStream &a_stream)

Protected Attributes

- Node ∗ m_root

  Root of tree.
- ELEMTYPEREAL m_unitSphereVolume

  Unit sphere constant for required number of dimensions.

3.15.1 Detailed Description

template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int T-MINNODES = TMAXNODES / 2>class RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >

Implementation of RTree, a multidimensional bounding rectangle tree. Example usage: For a 3-dimensional tree use RTree<Object∗, float, 3> myTree;

This modified, templated C++ version by Greg Douglas at Auran (`http://www.auran.com`)

DATATYPE Referenced data, should be int, void∗, obj∗ etc. no larger than sizeof<void∗> and simple type ELEM-TYPE Type of element such as int or float NUMDIMS Number of dimensions such as 2 or 3 ELEMTYPEREAL Type of element that allows fractional and large values such as float or double, for use in volume calcs

NOTES: Inserting and removing data requires the knowledge of its constant Minimal Bounding Rectangle. This version uses new/delete for nodes, I recommend using a fixed size allocator for efficiency. Instead of using a callback function for returned results, I recommend and efficient pre-sized, grow-only memory array similar to MFC CArray or STL Vector for returning search query result.

3.15.2 Member Typedef Documentation

3.15.2.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> typedef bool(∗ RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::t_resultCallback)(DATATYPE, void ∗)

3.15.3 Member Enumeration Documentation

3.15.3.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> anonymous enum

Enumerator

MAXNODES	Max elements in node.
MINNODES	Min elements in node.

3.15.4 Constructor & Destructor Documentation

3.15.4.1 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::RTree ( )

3.15.4.2 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> virtual RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::∼RTree ( ) [virtual]

3.15.5 Member Function Documentation

3.15.5.1 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::AddBranch (const Branch ∗ *a_branch*, Node ∗ *a_node*, Node ∗∗ *a_newNode*) [protected]

3.15.5.2 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> ListNode∗ RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::AllocListNode ( ) [protected]

3.15.5.3 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> Node∗ RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::AllocNode ( ) [protected]

3.15.5.4 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> ELEMTYPEREAL RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::CalcRectVolume (Rect ∗ *a_rect*) [protected]

3.15.5.5 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::ChoosePartition (PartitionVars ∗ *a_parVars*, int *a_minFill*) [protected]

3.15.5.6 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Classify (int *a_index*, int *a_group*, PartitionVars ∗ *a_parVars*) [protected]

3.15.5.7 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> Rect RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::CombineRect (const Rect ∗ *a_rectA*, const Rect ∗ *a_rectB*) [protected]

3.15.5.8 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> int RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Count ( )

Count the data elements in this container. This is slow as no internal counter is maintained.

3.15.5.9 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::CountRec (Node ∗ *a_node*, int & *a_count*) [protected]

3.15.5.10 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::DisconnectBranch (Node ∗ *a_node*, int *a_index*) [protected]

3.15.5.11 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::FreeListNode (ListNode ∗ *a_listNode*) [protected]

3.15.5.12 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::FreeNode (Node * *a_node*) `[protected]`

3.15.5.13 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> DATATYPE& RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::GetAt (Iterator & *a_it*) `[inline]`

Get object at iterator position.

3.15.5.14 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::GetBranches (Node * *a_node*, const Branch * *a_branch*, PartitionVars * *a_parVars*) `[protected]`

3.15.5.15 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::GetFirst (Iterator & *a_it*) `[inline]`

Get 'first' for iteration.

3.15.5.16 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::GetNext (Iterator & *a_it*) `[inline]`

Get Next for iteration.

3.15.5.17 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::InitNode (Node * *a_node*) `[protected]`

3.15.5.18 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::InitParVars (PartitionVars * *a_parVars*, int *a_maxRects*, int *a_minFill*) `[protected]`

3.15.5.19 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::InitRect (Rect * *a_rect*) `[protected]`

3.15.5.20 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Insert (const ELEMTYPE *a_min[NUMDIMS]*, const ELEMTYPE *a_max[NUMDIMS]*, const DATATYPE & *a_dataId*)

Insert entry

Parameters

a_min	Min of bounding rect
a_max	Max of bounding rect
a_dataId	Positive Id of data. Maybe zero, but negative numbers not allowed.

3.15.5.21 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::InsertRect (const Branch & *a_branch*, Node ** *a_root*, int *a_level*) [protected]

3.15.5.22 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::InsertRectRec (const Branch & *a_branch*, Node * *a_node*, Node ** *a_newNode*, int *a_level*) [protected]

3.15.5.23 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::IsNull (Iterator & *a_it*) [inline]

Is iterator NULL, or at end?

3.15.5.24 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Load (const char * *a_fileName*)

Load tree contents from file.

3.15.5.25 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Load (RTFileStream & *a_stream*)

Load tree contents from stream.

3.15.5.26 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::LoadNodes (Node * *a_nodeA*, Node * *a_nodeB*, PartitionVars * *a_parVars*) [protected]

3.15.5.27 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::LoadRec (Node * *a_node*, RTFileStream & *a_stream*) [protected]

3.15.5.28 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> Rect RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::NodeCover (Node * *a_node*) [protected]

3.15.5.29 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Overlap (Rect * *a_rectA*, Rect * *a_rectB*) [protected]

3.15.5.30 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> int RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PickBranch (const Rect * *a_rect*, Node * *a_node*) [protected]

3.15.5.31 template< class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::PickSeeds (PartitionVars * *a_parVars*) [protected]

3.15.5.32 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> ELEMTYPEREAL RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::RectSphericalVolume (Rect * *a_rect*) [protected]

3.15.5.33 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> ELEMTYPEREAL RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::RectVolume (Rect * *a_rect*) [protected]

3.15.5.34 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::ReInsert (Node * *a_node*, ListNode ** *a_listNode*) [protected]

3.15.5.35 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Remove (const ELEMTYPE *a_min[NUMDIMS]*, const ELEMTYPE *a_max[NUMDIMS]*, const DATATYPE & *a_dataId*)

Remove entry

Parameters

a_min	Min of bounding rect
a_max	Max of bounding rect
a_dataId	Positive Id of data. Maybe zero, but negative numbers not allowed.

3.15.5.36 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::RemoveAll ( )

Remove all entries from tree.

3.15.5.37 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::RemoveAllRec (Node * *a_node*) [protected]

3.15.5.38 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::RemoveRect (Rect * *a_rect*, const DATATYPE & *a_id*, Node ** *a_root*) [protected]

3.15.5.39 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::RemoveRectRec (Rect * *a_rect*, const DATATYPE & *a_id*, Node * *a_node*, ListNode ** *a_listNode*) [protected]

3.15.5.40 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Reset ( ) [protected]

3.15.5.41 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Save (const char * *a_fileName*)

Save tree contents to file.

3.15.5.42 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Save (RTFileStream & *a_stream*)

Save tree contents to stream.

3.15.5.43 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::SaveRec (Node * *a_node*, RTFileStream & *a_stream*) [protected]

3.15.5.44 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> int RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Search (const ELEMTYPE *a_min[NUMDIMS]*, const ELEMTYPE *a_max[NUMDIMS]*, t_resultCallback *a_resultCallback*, void * *a_context*)

Find all within search rectangle

Parameters

a_min	Min of search bounding rect
a_max	Max of search bounding rect
a_searchResult	Search result array. Caller should set grow size. Function will reset, not append to array.
a_resultCallback	Callback function to return result. Callback should return 'true' to continue searching
a_context	User context to pass as parameter to a_resultCallback

Returns

Returns the number of entries found

3.15.5.45 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> bool RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Search (Node * *a_node*, Rect * *a_rect*, int & *a_foundCount*, t_resultCallback *a_resultCallback*, void * *a_context*) [protected]

3.15.5.46 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> void RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::SplitNode (Node * *a_node*, const Branch * *a_branch*, Node ** *a_newNode*) [protected]

3.15.6 Field Documentation

3.15.6.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> Node* RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::m_root [protected]

Root of tree.

3.15.6.2 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> ELEMTYPEREAL RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::m_unitSphereVolume [protected]

Unit sphere constant for required number of dimensions.

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

- RTree.h

3.16 schedInfo Struct Reference

```
#include "structs.h"
```

Data Fields

- bool clusterScratch
- bool status

3.16.1 Field Documentation

3.16.1.1 bool schedInfo::clusterScratch

3.16.1.2 bool schedInfo::status

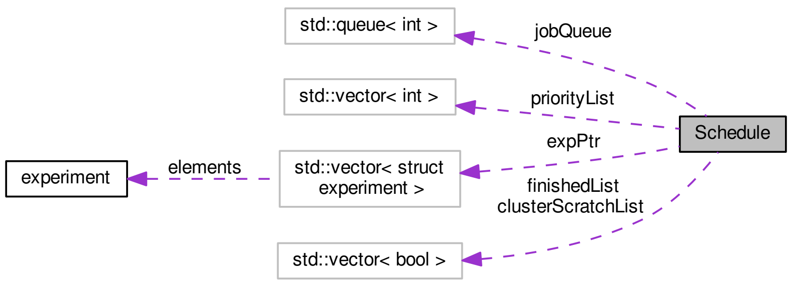
The documentation for this struct was generated from the following file:

- structs.h

3.17 Schedule Class Reference

```
#include "schedule.h"
```

Collaboration diagram for Schedule:



Public Member Functions

- Schedule (vector< struct experiment > ∗experimentList)
- bool determineReuse (int inExperID, int ∗outIDReuse, int ∗outInstanceCluster)

Data Fields

- std::vector< bool > finishedList
- std::vector< struct experiment > ∗ expPtr

Private Member Functions

- bool schedGreedy (int ∗outID, int ∗outInstanceCluster)

Private Attributes

- std::vector< int > priorityList
- std::vector< bool > clusterScratchList
- std::queue< int > jobQueue

3.17.1 Constructor & Destructor Documentation

3.17.1.1 Schedule::Schedule (vector< struct **experiment** > ∗ *experimentList*)

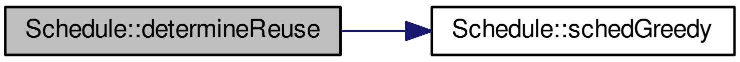
Schedule constructor

3.17.2 Member Function Documentation

3.17.2.1 bool Schedule::determineReuse (int *inExperID,* int ∗ *outIDReuse,* int ∗ *outInstanceCluster*)

Determines if the variant should be clustered from scratch or not If reusing data, then it uses the completed outID-Reuse, and clusters variant outInstanceCluster

Here is the call graph for this function:



3.17.2.2 bool Schedule::schedGreedy (int ∗ *outID,* int ∗ *outInstanceCluster*) `[private]`

The schedule type: only use greedy schedule from the paper

3.17.3 Field Documentation

3.17.3.1 std::vector<bool> Schedule::clusterScratchList `[private]`

Corresponding list of variants that must be clustered from scratch

3.17.3.2 std::vector<struct **experiment**>∗ Schedule::expPtr

Pointer to the vector of defined variants

3.17.3.3 std::vector<bool> Schedule::finishedList

Keeps track of variants that have completely finished

3.17.3.4 std::queue<int> Schedule::jobQueue `[private]`

Queue for variant priority ordering

3.17.3.5 std::vector<int> Schedule::priorityList [private]

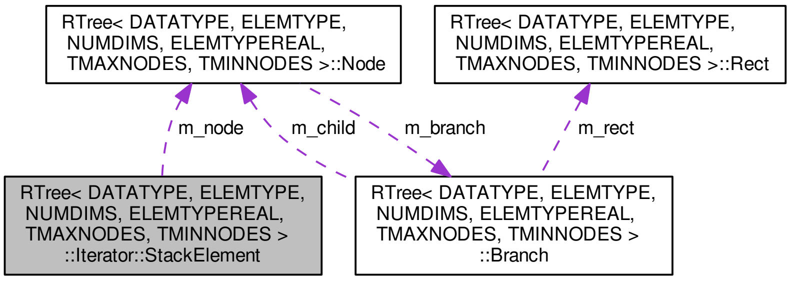
The order in which the variants should be clustered; stores the ids of the variant list

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

- schedule.h
- schedule.cpp

3.18 RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::StackElement Struct Reference

Collaboration diagram for RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::StackElement:


RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect
m_node
m_child
m_branch
m_rect
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > ::Iterator::StackElement
RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES > ::Branch

Data Fields

- Node ∗ m_node
- int m_branchIndex

3.18.1 Field Documentation

3.18.1.1 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> int RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::StackElement::m_branchIndex

3.18.1.2 template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL = ELEMTYPE, int TMAXNODES = 8, int TMINNODES = TMAXNODES / 2> Node∗ RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::StackElement::m_node

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

- RTree.h

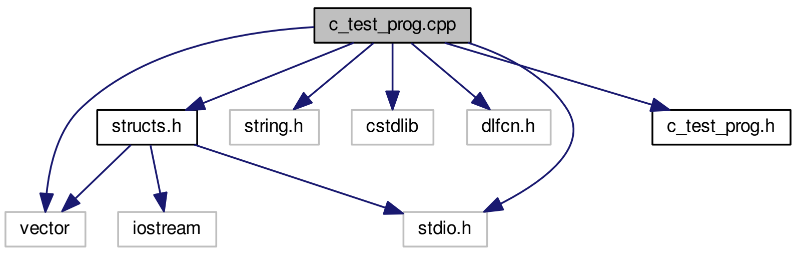
Chapter 4

File Documentation

4.1 c_test_prog.cpp File Reference

```
#include <vector>
#include <stdio.h>
#include <string.h>
#include <cstdlib>
#include <dlfcn.h>
#include "structs.h"
#include "c_test_prog.h"
```

Include dependency graph for c_test_prog.cpp:



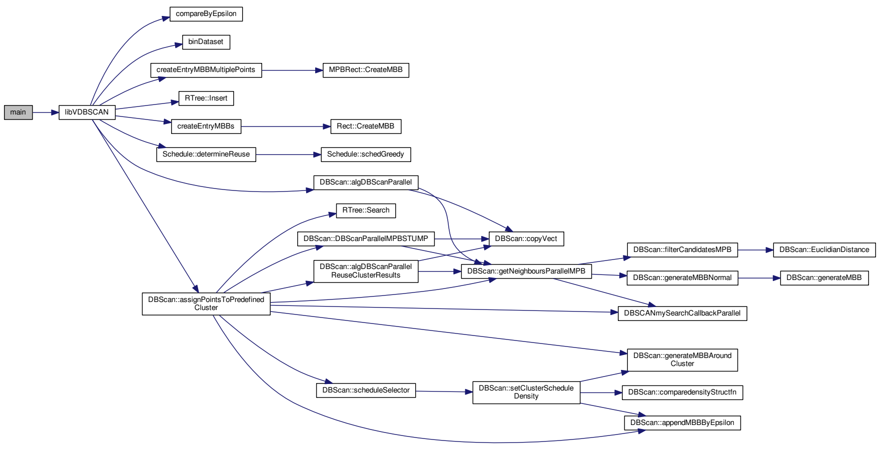
Functions

- int main ()

4.1.1 Function Documentation

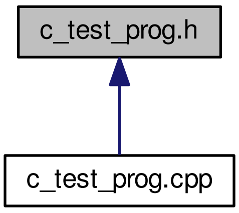
4.1.1.1 int main ( )

Here is the call graph for this function:


main
libVDBSCAN
compareByEpsilon
binDataset
createEntryMBBMultiplePoints
MPBRect::CreateMBB
RTree::Insert
createEntryMBBs
Rect::CreateMBB
Schedule::determineReuse
Schedule::schedGreedy
DBScan::algDBScanParallel
RTree::Search
DBScan::DBScanParallelMPBSTUMP
DBScan::copyVect
DBScan::algDBScanParallel ReuseClusterResults
DBScan::getNeighboursParallelMPB
DBScan::filterCandidatesMPB
DBScan::EuclidianDistance
DBScan::generateMBBNormal
DBScan::generateMBB
DBScan::assignPointsToPredefined Cluster
DBSCANmySearchCallbackParallel
DBScan::generateMBBAround Cluster
DBScan::scheduleSelector
DBScan::setClusterSchedule Density
DBScan::comparedensityStructfn
DBScan::appendMBBByEpsilon

4.2 c_test_prog.h File Reference

This graph shows which files directly or indirectly include this file:



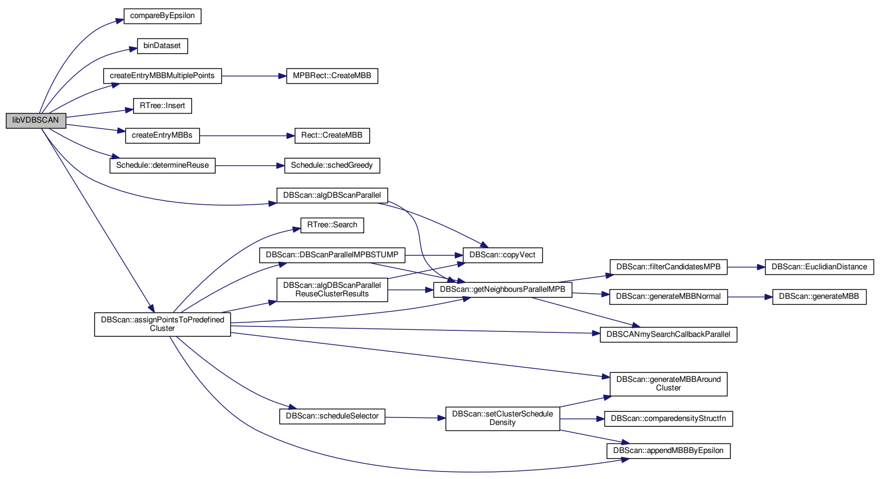
Functions

- int libVDBSCAN (double ∗inputx, double ∗inputy, unsigned int datasetSize, double ∗inputEpsilon, unsigned int ∗inputMinpts, unsigned int numVariants, int MBBsize, unsigned int ∗retArr, bool verbose)

4.2.1 Function Documentation

4.2.1.1 int libVDBSCAN (double ∗ *inputx,* double ∗ *inputy,* unsigned int *datasetSize,* double ∗ *inputEpsilon,* unsigned int ∗ *inputMinpts,* unsigned int *numVariants,* int *MBBsize,* unsigned int ∗ *retArr,* bool *verbose*)

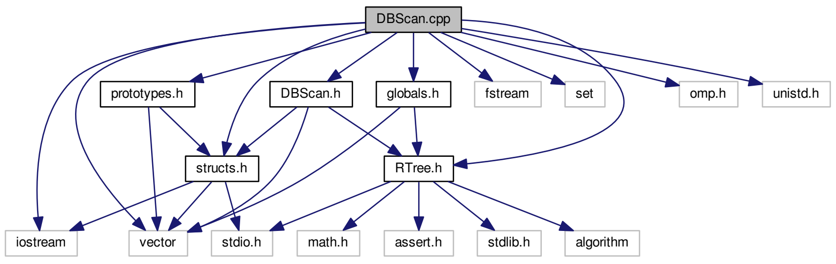
Here is the call graph for this function:


libVDBSCAN
compareByEpsilon
binDataset
createEntryMBBMultiplePoints
MPBRect::CreateMBB
RTree::Insert
createEntryMBBs
Rect::CreateMBB
Schedule::determineReuse
Schedule::schedGreedy
DBScan::algDBScanParallel
RTree::Search
DBScan::DBScanParallelMPBSTUMP
DBScan::copyVect
DBScan::algDBScanParallel ReuseClusterResults
DBScan::getNeighboursParallelMPB
DBScan::filterCandidatesMPB
DBScan::EuclidianDistance
DBScan::generateMBBNormal
DBScan::generateMBB
DBScan::assignPointsToPredefined Cluster
DBSCANmySearchCallbackParallel
DBScan::generateMBBAround Cluster
DBScan::scheduleSelector
DBScan::setClusterSchedule Density
DBScan::comparedensityStructfn
DBScan::appendMBBByEpsilon

4.3 DBScan.cpp File Reference

```
#include "structs.h"
#include "prototypes.h"
#include "globals.h"
#include <fstream>
#include <vector>
#include <set>
#include "RTree.h"
#include "DBScan.h"
#include <omp.h>
#include <iostream>
#include <unistd.h>
```

Include dependency graph for DBScan.cpp:



Variables

- const int NUMMINREUSEPTS =100
- std::vector< int > neighbourList

- std::vector< int > neighbourListParallel [NSEARCHTHREADS]

4.3.1 Variable Documentation

4.3.1.1 std::vector<int> neighbourList

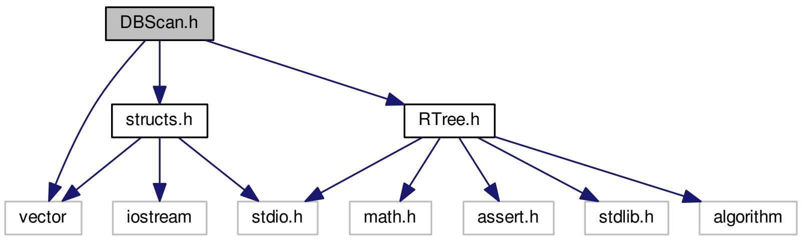
4.3.1.2 std::vector<int> neighbourListParallel[NSEARCHTHREADS]

4.3.1.3 const int NUMMINREUSEPTS =100

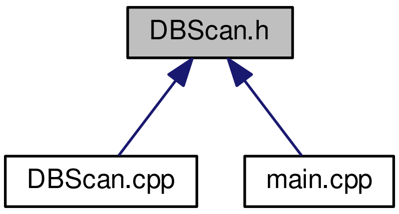
4.4 DBScan.h File Reference

```
#include "structs.h"
#include <vector>
#include "RTree.h"
```

Include dependency graph for DBScan.h:



This graph shows which files directly or indirectly include this file:



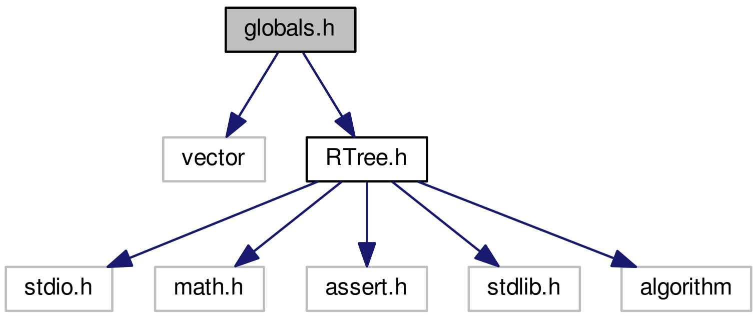
Data Structures

- class DBScan

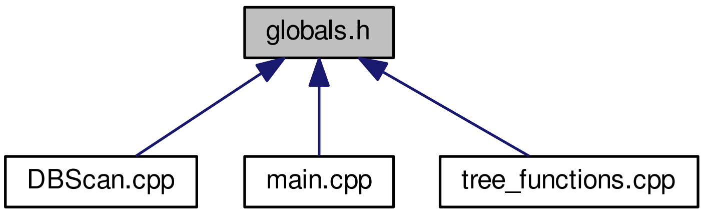
4.5 globals.h File Reference

```
#include <vector>
#include "RTree.h"
```

Include dependency graph for globals.h:



This graph shows which files directly or indirectly include this file:



Variables

- struct dataElem ∗ dataPoints
- std::vector< int > neighbourListParallel []

4.5.1 Variable Documentation

4.5.1.1 struct **dataElem**∗ dataPoints

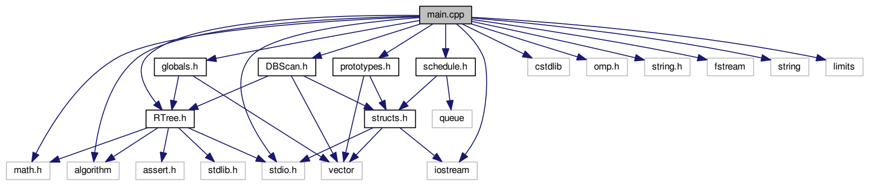
4.5.1.2 std::vector<int> neighbourListParallel[ ]

4.6 main.cpp File Reference

```
#include <math.h>
```

```
#include <cstdlib>
#include <stdio.h>
#include "prototypes.h"
#include "globals.h"
#include "RTree.h"
#include "omp.h"
#include "DBScan.h"
#include "schedule.h"
#include <algorithm>
#include <string.h>
#include <fstream>
#include <iostream>
#include <string>
#include <limits>
```

Include dependency graph for main.cpp:



Functions

- int libVDBSCAN (double ∗inputx, double ∗inputy, unsigned int datasetSize, double ∗inputEpsilon, unsigned int ∗inputMinpts, unsigned int numVariants, int MBBsize, unsigned int ∗retArr, bool verbose)
- void createEntryMBBMultiplePoints (std::vector< dataElem > ∗dataPoints, std::vector< std::vector< int > > ∗MPB_ids, MPBRect ∗dataRectsMPB, int MBBSize)

  Generates MBBs for for the R-tree when indexing multiple points per MBB.
- void createEntryMBBs (std::vector< dataElem > ∗dataPoints, Rect ∗dataRects)

  Generates MBBs for the R-tree.
- void binDataset (std::vector< dataElem > ∗dataPoints, int numBins, std::vector< int > ∗mapping, bool verbose)

  Bins the 2-D input dataset, and keeps track of where the points in space were mapped to the original input dataset.
- bool compareByEpsilon (const experiment &a, const experiment &b)

  Comparison function for sorting.
- bool compareDataElemStructFunc (const dataElem &elem1, const dataElem &elem2)

  Comparison function for sorting.
- int bin_x (double x)

  Used for binning the input dataset.
- int bin_y (double x)

  Used for binning the input dataset.

4.6.1 Function Documentation

4.6.1.1 int bin_x (double *x*)

Used for binning the input dataset.

4.6.1.2 int bin_y (double *x*)

Used for binning the input dataset.

4.6.1.3 void binDataset (std::vector< **dataElem** > * *dataPoints,* int *numBins,* std::vector< int > * *mapping,* bool *verbose*)

Bins the 2-D input dataset, and keeps track of where the points in space were mapped to the original input dataset.

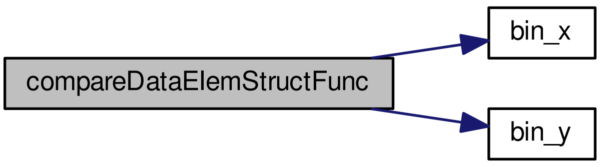
4.6.1.4 bool compareByEpsilon (const **experiment** & *a,* const **experiment** & *b*)

Comparison function for sorting.

4.6.1.5 bool compareDataElemStructFunc (const **dataElem** & *elem1,* const **dataElem** & *elem2*)

Comparison function for sorting.

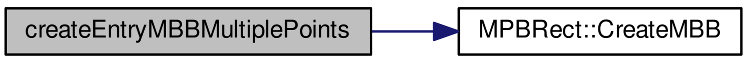
Here is the call graph for this function:



4.6.1.6 void createEntryMBBMultiplePoints (std::vector< **dataElem** > * *dataPoints,* std::vector< std::vector< int > > * *MPB_ids,* **MPBRect** * *dataRectsMPB,* int *MBBSize*)

Generates MBBs for for the R-tree when indexing multiple points per MBB.

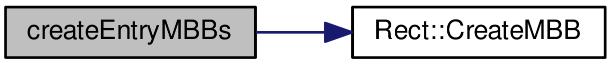
Here is the call graph for this function:



4.6.1.7 void createEntryMBBs (std::vector< **dataElem** > * *dataPoints,* **Rect** * *dataRects*)

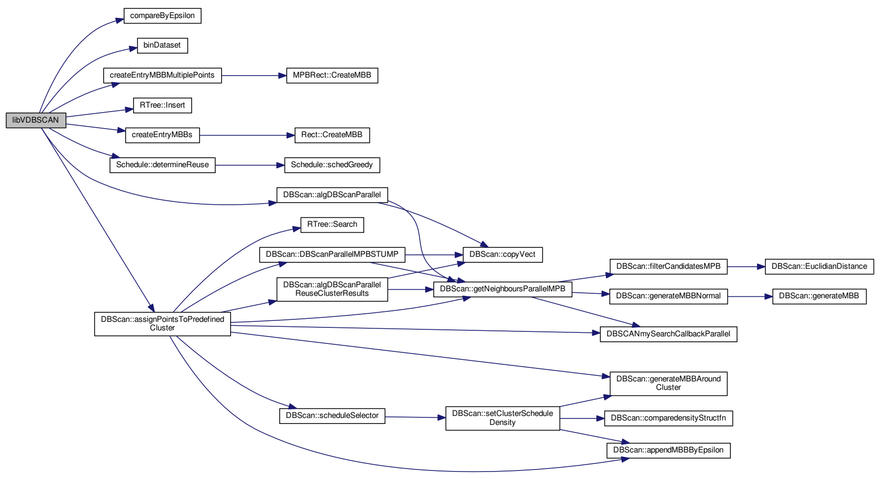
Generates MBBs for the R-tree.

Here is the call graph for this function:



4.6.1.8 int libVDBSCAN (double ∗ *inputx,* double ∗ *inputy,* unsigned int *datasetSize,* double ∗ *inputEpsilon,* unsigned int ∗ *inputMinpts,* unsigned int *numVariants,* int *MBBsize,* unsigned int ∗ *retArr,* bool *verbose*)

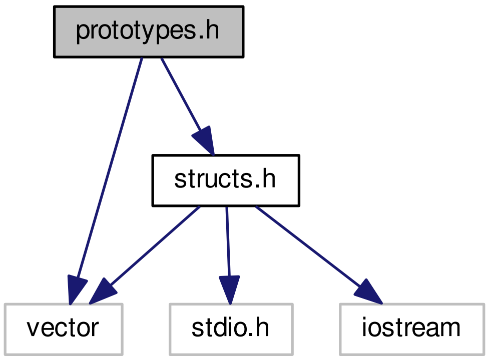
Here is the call graph for this function:


libVDBSCAN
compareByEpsilon
binDataset
createEntryMBBMultiplePoints
MPBRect::CreateMBB
RTree::Insert
createEntryMBBs
Rect::CreateMBB
Schedule::determineReuse
Schedule::schedGreedy
DBScan::algDBScanParallel
RTree::Search
DBScan::DBScanParallelMPBSTUMP
DBScan::copyVect
DBScan::algDBScanParallel ReuseClusterResults
DBScan::getNeighboursParallelMPB
DBScan::filterCandidatesMPB
DBScan::EuclidianDistance
DBScan::generateMBBNormal
DBScan::generateMBB
DBScan::assignPointsToPredefined Cluster
DBSCANmySearchCallbackParallel
DBScan::generateMBBAround Cluster
DBScan::scheduleSelector
DBScan::setClusterSchedule Density
DBScan::comparedensityStructfn
DBScan::appendMBBByEpsilon

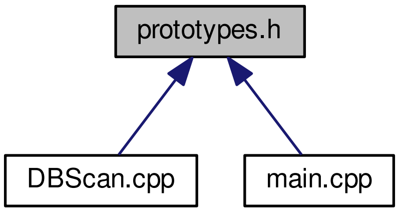
4.7 prototypes.h File Reference

```
#include "structs.h"
#include <vector>
```

Include dependency graph for prototypes.h:



This graph shows which files directly or indirectly include this file:



Functions

- void importDataset (std::vector< dataElem > ∗dataPoints, char ∗fname)

  Imports the 2-D dataset.
- void createEntryMBBs (std::vector< dataElem > ∗dataPoints, Rect ∗dataRects)

  Generates MBBs for the R-tree.
- bool DBSCANmySearchCallbackParallel (int id, void ∗arg)

  Callback function for the R-tree.
- bool compareDataElemStructFunc (const dataElem &elem1, const dataElem &elem2)

  Comparison function for sorting.
- void createEntryMBBMultiplePoints (std::vector< dataElem > ∗dataPoints, std::vector< std::vector< int > > ∗MPB_ids, MPBRect ∗dataRectsMPB, int MBBsize)

  Generates MBBs for for the R-tree when indexing multiple points per MBB.
- void importDBScanInstances (std::vector< struct experiment > ∗exper, char ∗fname)

  Imports the list of DBSCAN instances (not used in the shared library version)
- bool compareByEpsilon (const experiment &a, const experiment &b)

Comparison function for sorting.

- int bin_x (double x)

  Used for binning the input dataset.
- int bin_y (double x)

  Used for binning the input dataset.
- void binDataset (std::vector< dataElem > ∗dataPoints, int numBins, std::vector< int > ∗mapping, bool verbose)

  Bins the 2-D input dataset, and keeps track of where the points in space were mapped to the original input dataset.

4.7.1 Function Documentation

4.7.1.1 int bin_x (double *x*)

Used for binning the input dataset.

4.7.1.2 int bin_y (double *x*)

Used for binning the input dataset.

4.7.1.3 void binDataset (std::vector< **dataElem** > ∗ *dataPoints,* int *numBins,* std::vector< int > ∗ *mapping,* bool *verbose*)

Bins the 2-D input dataset, and keeps track of where the points in space were mapped to the original input dataset.

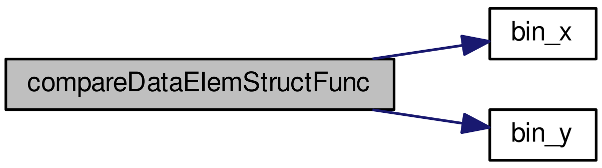
4.7.1.4 bool compareByEpsilon (const **experiment** & *a,* const **experiment** & *b*)

Comparison function for sorting.

4.7.1.5 bool compareDataElemStructFunc (const **dataElem** & *elem1,* const **dataElem** & *elem2*)

Comparison function for sorting.

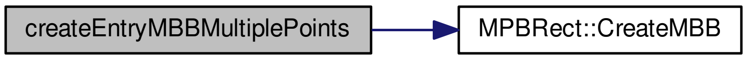
Here is the call graph for this function:



4.7.1.6 void createEntryMBBMultiplePoints (std::vector< **dataElem** > ∗ *dataPoints,* std::vector< std::vector< int > > ∗ *MPB_ids,* **MPBRect** ∗ *dataRectsMPB,* int *MBBsize*)

Generates MBBs for for the R-tree when indexing multiple points per MBB.

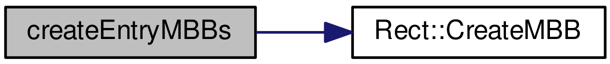
Here is the call graph for this function:



4.7.1.7 void createEntryMBBs (std::vector< **dataElem** > ∗ *dataPoints*, **Rect** ∗ *dataRects*)

Generates MBBs for the R-tree.

Here is the call graph for this function:



4.7.1.8 bool DBSCANmySearchCallbackParallel (int *id*, void ∗ *arg*)

Callback function for the R-tree.

4.7.1.9 void importDataset (std::vector< **dataElem** > ∗ *dataPoints*, char ∗ *fname*)

Imports the 2-D dataset.

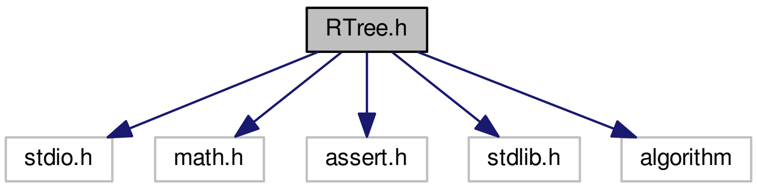
4.7.1.10 void importDBScanInstances (std::vector< struct **experiment** > ∗ *exper*, char ∗ *fname*)

Imports the list of DBSCAN instances (not used in the shared library version)

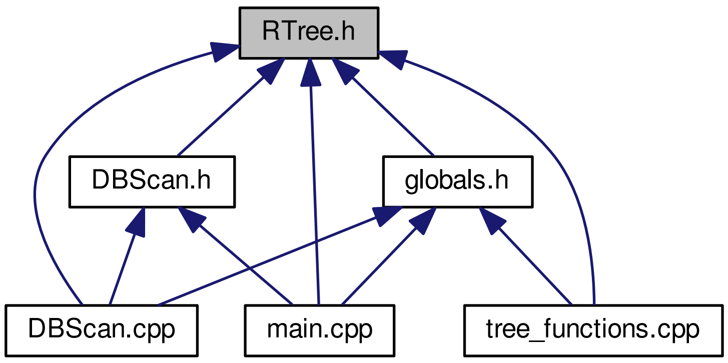
4.8 RTree.h File Reference

```
#include <stdio.h>
#include <math.h>
#include <assert.h>
#include <stdlib.h>
#include <algorithm>
```

Include dependency graph for RTree.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- class RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >
- class RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator

  Iterator is not remove safe.
- struct RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Iterator::StackElement
- struct RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Rect

  Minimal bounding rectangle (n-dimensional)
- struct RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Branch
- struct RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::Node

  Node for each branch level.
- struct RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::ListNode

  A link list of nodes for reinsertion after a delete operation.

- struct RTree< DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES >::-PartitionVars

  Variables for finding a split partition.

- class RTFileStream

Macros

- #define ASSERT assert
- #define Min std::min
- #define Max std::max
- #define RTREE_TEMPLATE template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMT-YPEREAL, int TMAXNODES, int TMINNODES>
- #define RTREE_QUAL RTree<DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, T-MINNODES>
- #define RTREE_DONT_USE_MEMPOOLS
- #define RTREE_USE_SPHERICAL_VOLUME

4.8.1 Macro Definition Documentation

4.8.1.1 #define ASSERT assert

4.8.1.2 #define Max std::max

4.8.1.3 #define Min std::min

4.8.1.4 #define RTREE_DONT_USE_MEMPOOLS

4.8.1.5 #define RTREE_QUAL RTree<DATATYPE, ELEMTYPE, NUMDIMS, ELEMTYPEREAL, TMAXNODES, TMINNODES>

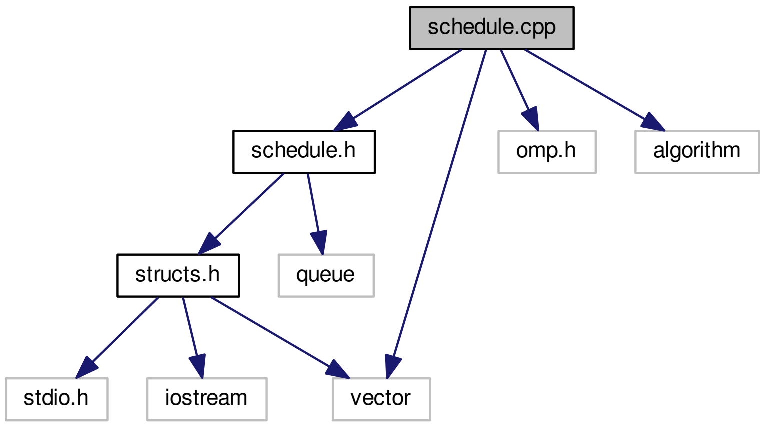
4.8.1.6 #define RTREE_TEMPLATE template<class DATATYPE, class ELEMTYPE, int NUMDIMS, class ELEMTYPEREAL, int TMAXNODES, int TMINNODES>

4.8.1.7 #define RTREE_USE_SPHERICAL_VOLUME

4.9 schedule.cpp File Reference

```
#include "schedule.h"
#include <omp.h>
#include <vector>
#include <algorithm>
```

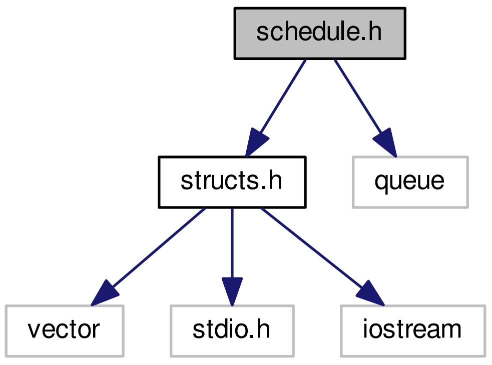
Include dependency graph for schedule.cpp:



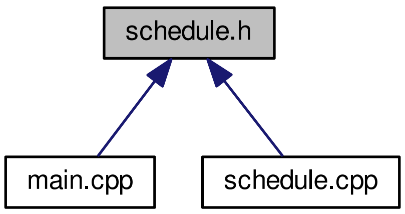
4.10 schedule.h File Reference

```
#include "structs.h"
#include <queue>
```

Include dependency graph for schedule.h:



This graph shows which files directly or indirectly include this file:

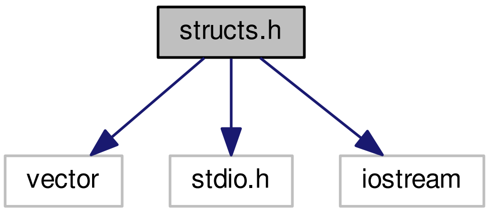


Data Structures

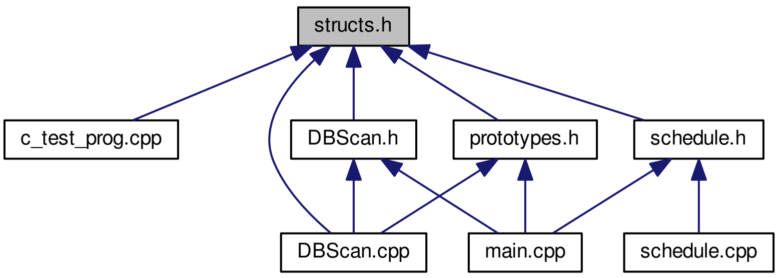
- class Schedule

4.11 structs.h File Reference

```
#include <vector>
#include <stdio.h>
#include <iostream>
```
Include dependency graph for structs.h:



This graph shows which files directly or indirectly include this file:



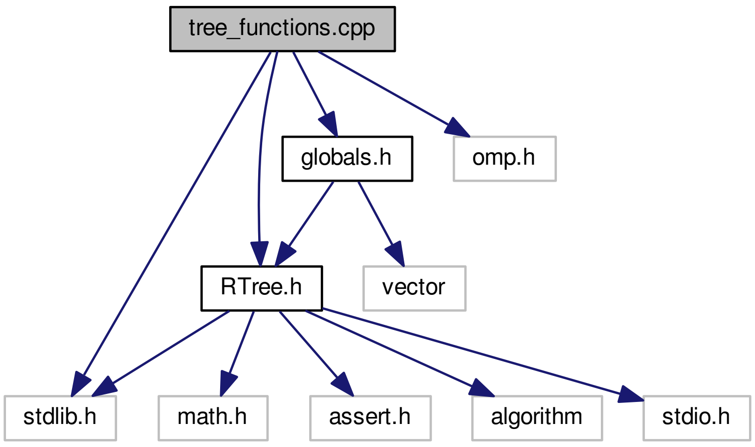
Data Structures

- struct experiment
- struct schedInfo
- struct dataElem
- struct densityStruct
- struct MPBRect
- struct Rect
- struct QueryRect

4.12 tree_functions.cpp File Reference

```
#include "RTree.h"
#include "globals.h"
#include <omp.h>
#include <stdlib.h>
```

Include dependency graph for tree_functions.cpp:



Functions

- bool DBSCANmySearchCallbackParallel (int id, void ∗arg)

  Callback function for the R-tree.

4.12.1 Function Documentation

4.12.1.1 bool DBSCANmySearchCallbackParallel (int *id,* void ∗ *arg*)

Callback function for the R-tree.

Index