ITTF Reference Manual

1

Generated by Doxygen 1.2.12-20011125

Wed Sep 18 18:40:14 2002

Contents

Chapter 1

ITTF Hierarchical Index

1.1 ITTF Class Hierarchy

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

Chapter 2

ITTF Compound Index

2.1 ITTF Compound List

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

Chapter 3

ITTF File Index

3.1 ITTF File List

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

Chapter 4

ITTF Class Documentation

4.1 CombinationIterator Class Template Reference

`#include <CombinationIterator.h>`

Public Types

- typedef vector< T > **tvector**

Public Methods

- **CombinationIterator** ()

  Defualt Constructor.

- virtual ~**CombinationIterator** ()

  Default Destructor.

- void **push_back** (tvector::const_iterator begin, tvector::const_iterator end)

  Add sequence.

- void **clear** ()

  Full internal reset.

- int **size** () const

  Return number of possible combinations.

- bool **valid** () const

  check that each range of points has size>0.

- const tvector::const_iterator & **end** () const

  Access to end, marks termination of forward traversal.

- void **init** ()

  Reset iterator to first combination.

- bool **operator==** (const tvector::const_iterator &rhs) const

  equality.

- bool **operator!=** (const tvector::const_iterator &rhs) const

  inequality.

- CombinationIterator & **operator++** ()

  prefix.

- CombinationIterator **operator++** (int)

  postfix.

- const tvector & **operator** ∗ ()

  dereference iterator.

- void **print** () const

  print utility.

4.1.1 Detailed Description

template<class T> class CombinationIterator< T >

A class to make combinations of elements stored in vectors, over a variable number of vectors. i.e., you can make combinations from points from set 1 with those in set 2, set 3, ..., set n, with an stl iterator interface. This class meets the requirements of an STL input iterator and can thus be used in any STL agloorthim that requires only an input iterator (e.g., find, find_if, etc). Additionally, we define validity via bool **valid**() (p. 19) s.t. the iterator is vailid iff each set is non-empty.

Currently, the iterator works only with sets that are defined by iterators int std::vector<T>. However, with support of templated member functions one could easily extend the calss to deal with ranges defined by iterators from any type of STL container.

As usual, validity is defined via [begin,end).

Author:
M.L. Miller (Yale Software)

Examples:
CombinationIterator_ex.cxx.

Definition at line 79 of file CombinationIterator.h.

4.1.2 Member Function Documentation

4.1.2.1 template<class T> void CombinationIterator< T >::clear () `[inline]`

Full internal reset.

A call to **clear**() (p. 17) removes all sequences that have been added to the iterator via the **push_back**() (p. 19) method. That is, once **clear**() (p. 17) has been called one must again add all ranges via **push_back**() (p. 19).

Definition at line 161 of file CombinationIterator.h.

4.1.2.2 template<class T> const CombinationIterator< T >::tvector::const_iterator & CombinationIterator< T >::end () const `[inline]`

Access to end, marks termination of forward traversal.

We provide a const_iterator that marks the point one past the last valid combination of the CombinationIterator.

Definition at line 238 of file CombinationIterator.h.

Referenced by push_back(), and ~CombinationIterator().

4.1.2.3 template<class T> void CombinationIterator< T >::init ()

Reset iterator to first combination.

Init resets the iterator to the first possible combination. That is, after forward traversal one can return to to the beginning of the traversal via a call to **init**() (p. 17).

Definition at line 248 of file CombinationIterator.h.

Referenced by operator++().

4.1.2.4 template<class T> const CombinationIterator< T >::tvector & CombinationIterator< T >::operator * ()

dereference iterator.

Dereferencing the iterator returns a reference to a vector that containes the current possible combination. This reference points to a private vector member of CombinationIterator.

Definition at line 280 of file CombinationIterator.h.

4.1.2.5 template<class T> bool CombinationIterator< T >::operator!= (const tvector::const_iterator & *rhs*) const `[inline]`

inequality.

We provide an inequality operator that takes a std::vector<T>const_iterator as an arguement.

Definition at line 228 of file CombinationIterator.h.

References operator==().

4.1.2.6 template<class T> CombinationIterator< T > CombinationIterator< T >::operator++ (int)

postfix.

This postfix version of operator++ is implemented via a call to the prefi version. It should be noted that a call to the postfix version requires (as usual) a constructor call to CombinationIterator, and can thus be significantly less efficient than the prefix version.

Definition at line 193 of file CombinationIterator.h.

4.1.2.7 template<class T> bool CombinationIterator< T >::operator== (const tvector::const_iterator & *rhs*) const `[inline]`

equality.

We provied an equality operator that takes a std::vector<T>const_iterator as an arguement.

Definition at line 219 of file CombinationIterator.h.

Referenced by operator!=().

4.1.2.8 template<class T> void CombinationIterator< T >::print () const

print utility.

A call to print streams each range known to the iterator to the screen

Definition at line 293 of file CombinationIterator.h.

4.1.2.9 template<class T> void CombinationIterator< T >::push_back (tvector::const_iterator *begin*, tvector::const_iterator *end*)

Add sequence.

A sequence is defined to be valid over the range [begin,end). If the condition begin==end arises, this sequence is deemed to be invalid and invalidates the instance of CombinationIterator (see method **valid**() (p. 19)).

Definition at line 144 of file CombinationIterator.h.

References end().

4.1.2.10 template<class T> int CombinationIterator< T >::size () const

Return number of possible combinations.

Size is defined by summation of end-begin for each range that has been added to the iterator.

Definition at line 204 of file CombinationIterator.h.

4.1.2.11 template<class T> bool CombinationIterator< T >::valid () const

check that each range of points has size>0.

Validity is defined s.t. end_i!=begin_i for all ranges i. If one does not test the validity of each range i, then an attempt to dereference the CombinationIterator will result in dereferencing an invalid stl iterator.

Definition at line 262 of file CombinationIterator.h.

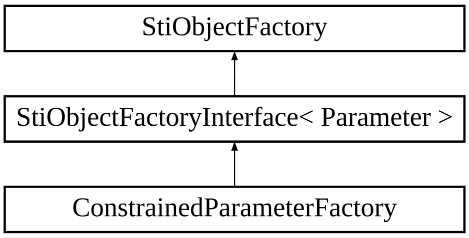
The documentation for this class was generated from the following file:

- Sti/**CombinationIterator.h**

4.2 ConstrainedParameterFactory Class Reference

#include <ConstrainedParameter.h>

Inheritance diagram for ConstrainedParameterFactory::



Public Methods

- **ConstrainedParameterFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

  This is the only constructor available.

- virtual ~**ConstrainedParameterFactory** ()

  Default destructor.

Protected Methods

- virtual void * **makeNewObject** () const

  Return a pointer to a new Parameter object on the heap.

4.2.1 Detailed Description

ConstrainedParameter factory

Definition at line 152 of file ConstrainedParameter.h.

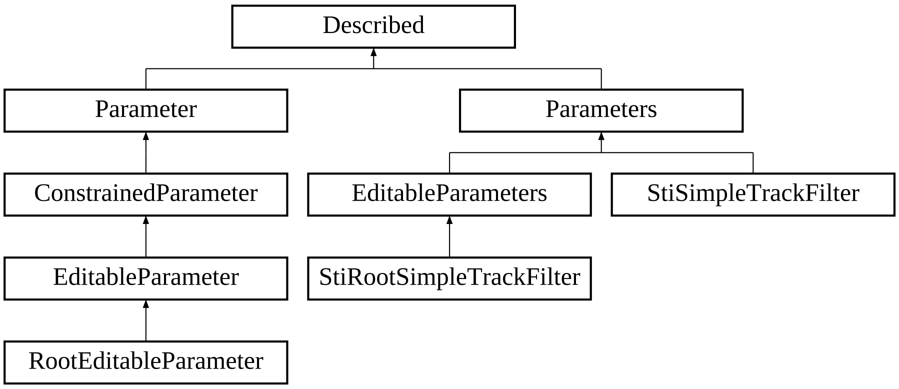
The documentation for this class was generated from the following files:

- Sti/**ConstrainedParameter.h**
- Sti/**ConstrainedParameter.cxx**

4.3 Described Class Reference

`#include <Described.h>`

Inheritance diagram for Described::



Public Methods

- virtual ~**Described** ()
- void **setDescription** (const string &description)

    Set the Describe of the object.

- const string **getDescription** () const

    Get the Describe of the object.

- bool **isDescribed** () const

    Determine whether Describe is set, i.e object has a Describe.

- bool **isDescription** (const string &description) const

    Determine whether Describe equals given Describe.

- bool **sameDescriptionAs** (const Described &described) const

    Determine whether Describe equals that of given object.

Protected Methods

- **Described** (const string &aDescribe="")

    Only derived class are Described.

Protected Attributes

- string **_description**

4.3.1 Detailed Description

This class encapsulates the notion of "Described". It should be used as base class to provide a "Described" property to objects.

Author:
Claude A Pruneau

Definition at line 18 of file Described.h.

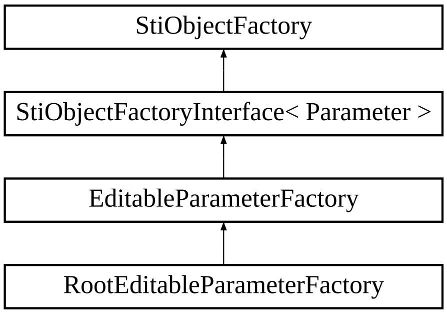
The documentation for this class was generated from the following files:

- Sti/**Described.h**
- Sti/**Described.cxx**

4.4 EditableParameterFactory Class Reference

`#include <EditableParameter.h>`

Inheritance diagram for EditableParameterFactory::



Public Methods

- **EditableParameterFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

  This is the only constructor available.

- virtual ~**EditableParameterFactory** ()

  Default destructor.

Protected Methods

- virtual void ∗ **makeNewObject** () const

  Return a pointer to a new Parameter object on the heap.

4.4.1 Detailed Description

EditableParameter factory

Definition at line 80 of file EditableParameter.h.

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

- Sti/**EditableParameter.h**
- Sti/**EditableParameter.cxx**

4.5 IteratorTriplet Class Template Reference

Helper class:.

`#include <CombinationIterator.h>`

Public Methods

- **IteratorTriplet** (T &begin, T &end)
- void **init** ()

Public Attributes

- T **beginIt**
- T **endIt**
- T **currentIt**

4.5.1 Detailed Description

template<class T> class IteratorTriplet< T >

Helper class:.

Definition at line 38 of file CombinationIterator.h.

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

- Sti/**CombinationIterator.h**

4.6 MessageType Class Reference

Public Methods

- **ADD_MESSAGE** (Hit)
- **ADD_MESSAGE** (Track)
- **ADD_MESSAGE** (Node)
- **ADD_MESSAGE** (Detector)
- **ADD_MESSAGE** (Geometry)
- **ADD_MESSAGE** (SeedFinder)
- **MessageType** (string name)
- virtual ∼**MessageType** ()

  deletes our ostream if it is not cout or cerr.

- string **getName** ()

  returns the name.

- unsigned int **getIndex** ()

  returns the index (0, 1, 2, 3, etc).

- unsigned int **getCode** ()

  returns the code (0x01, 0x02, 0x04, etc).

- ostream ∗ **getOstream** ()

  returns the ostream.

- void **setOstream** (ostream ∗pOstream)

  sets the ostream.

Static Public Methods

- unsigned int **getNtypes** ()

  returns the number of message types.

- MessageType ∗ **getTypeByIndex** (unsigned int iIndex)

  returns the MessageType with the given index, or NULL if none exists.

- MessageType ∗ **getTypeByCode** (unsigned int iCode)

  returns the MessageType with the given code, or NULL if none exists.

Protected Attributes

- string **m_name**
  name (HitMessage, TrackMessage, etc).
- unsigned int **m_iIndex**
  index (log_2(code)).
- unsigned int **m_iCode**
  bit code (0x01, 0x02, 0x04, etc).
- ostream * **m_pOstream**
  the ostream where messages of this type should go.

Static Protected Attributes

- unsigned int **s_nTypes** = 0
  number of message types.
- MessageType * **s_apTypes** [32] = {0}
  look up type by index.

4.6.1 Detailed Description

type of informational message

Author:
Ben Norman (Kent State)

Definition at line 23 of file MessageType.h.

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

- Sti/**MessageType.h**
- Sti/**MessageType.cxx**

4.7 Messenger Class Reference

Public Methods

- unsigned int **getRoutingCode** ()

  return the instance routing code.

- virtual ~**Messenger** ()

  Destructor;.

- bool **canWrite** ()

  determines whether or not the **MessengerBuf** (p. 29) *'s routing code and the static routing mask allow this Messenger to write anything. Always returns false when DEBUG is defined.*

Static Public Methods

- Messenger ∗ **instance** (unsigned int routing=0)

  Return a Messenger instance corresponding to the given routing code, or all routes allowed by the global mask if no routing is specified.

- unsigned int **setRoutingMask** (unsigned int routing)

  Set the global routing mask which tells which messages are actually delivered to a given stream. This mask is AND-ed with the routing code of a given Messenger to determine if a message should be printed. Returns the original routing mask.

- unsigned int **getRoutingMask** ()

  Returns the global routing mask.

- unsigned int **setRoutingBits** (unsigned int messages)

  Sets only the given bits in the global routing mask. Returns the original state of the bits.

- unsigned int **clearRoutingBits** (unsigned int messages)

  Clears only the given bits in the global routing mask. Returns the original state of the bits.

- unsigned int **getRoutingBits** (unsigned int messages)

  Returns the given bits in the global routing mask.

- void **init** (unsigned int routing=0)

Initialize the output streams for the message maps. This must be called before using a Messenger. If a routing code is specified, it is used as the global routing mask.

- void **kill** ()

  Delete any created Messenger & ofstream objects. This must be called after messenging is finished.

Protected Methods

- **Messenger** (unsigned int routing=0)

  Construct a Messenger with a MessengerMap using the given routing.

Static Protected Methods

- void **updateStates** ()

  updates the ios state bits of all Messengers based on whether or not they can read given the current global routing mask.

Static Protected Attributes

- messengerMap **s_messengerMap**

  static map of Messenger instances indexed by routing code.

- unsigned int **s_routing** = 0

  static routing mask, ANDed with the instance routing code.

4.7.1 Detailed Description

informational message routing.

Author:

Ben Norman (Kent State)

Definition at line 23 of file Messenger.h.

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

- Sti/**Messenger.h**
- Sti/**Messenger.cxx**

4.8 MessengerBuf Class Reference

Public Methods

- **MessengerBuf** (unsigned int routing)
- virtual ∼**MessengerBuf** ()
- virtual int **overflow** (int ch)

  this delivers the given character to all output streams.

- virtual streamsize **xsputn** (const char ∗text, streamsize n)
- unsigned int **getRoutingCode** ()

  return the instance routing code.

Protected Methods

- int **dispatchMessage** (const char ∗szMessage, streamsize iLen)

  sends the current message to all appropriate output streams. returns 0 (ok) or EOF.

Protected Attributes

- unsigned int **m_routing**

  routing code for this MessengerBuf tells which streams should be output.

4.8.1 Detailed Description

streambuf class used in **Messenger** (p. 27)

Author:

Ben Norman (Kent State)

Definition at line 16 of file MessengerBuf.h.

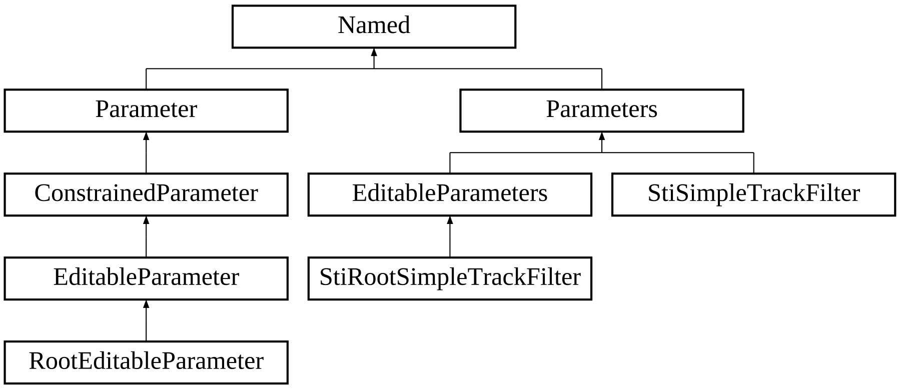
The documentation for this class was generated from the following files:

- Sti/**MessengerBuf.h**
- Sti/**MessengerBuf.cxx**

4.9 Named Class Reference

`#include <Named.h>`

Inheritance diagram for Named::



Public Methods

- virtual ∼**Named** ()
- void **setName** (const string &newName)

  Set the name of the object.

- const string **getName** () const

  Get the name of the object.

- bool **isNamed** () const

  Determine whether name is set, i.e object has a name.

- bool **isName** (const string &aName) const

  Determine whether name equals given name.

- bool **isNamedAs** (const Named &named) const

  Determine whether name equals that of given object.

Protected Methods

- **Named** (const string &aName="")

  Only derived class are Named.

Protected Attributes

- string **_name**

4.9.1 Detailed Description

This class encapsulates the notion of ”name”. It should be used as base class to provide a ”named” property to objects.include <string> use STD;

Author:
Claude A Pruneau

Definition at line 19 of file Named.h.

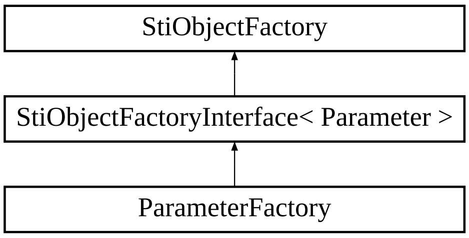
The documentation for this class was generated from the following files:

- Sti/**Named.h**
- Sti/**Named.cxx**

4.10 ParameterFactory Class Reference

`#include <Parameter.h>`

Inheritance diagram for ParameterFactory::



Public Methods

- **ParameterFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

    This is the only constructor available.

- virtual ~**ParameterFactory** ()

    Default destructor.

Protected Methods

- virtual void * **makeNewObject** () const

    Return a pointer to a new Parameter object on the heap.

4.10.1 Detailed Description

Parameter factory

Definition at line 110 of file Parameter.h.

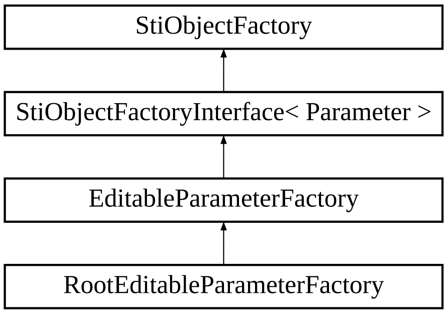
The documentation for this class was generated from the following files:

- Sti/**Parameter.h**
- Sti/**Parameter.cxx**

4.11 RootEditableParameterFactory Class Reference

`#include <RootEditableParameter.h>`

Inheritance diagram for RootEditableParameterFactory::



Public Methods

- **RootEditableParameterFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

  This is the only constructor available.

- virtual ~**RootEditableParameterFactory** ()

  Default destructor.

Protected Methods

- virtual void ∗ **makeNewObject** () const

  Return a pointer to a new Parameter object on the heap.

4.11.1 Detailed Description

ConstrainedParameter factory

Definition at line 57 of file RootEditableParameter.h.

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

- StiMaker/**RootEditableParameter.h**

- StiMaker/**RootEditableParameter.cxx**

4.12 StFastLineFitter Class Reference

`#include <StFastLineFitter.h>`

Public Methods

- **StFastLineFitter** ()

  Default Constructor.

- virtual ~**StFastLineFitter** ()

  Default Destructor.

- double **slope** () const

  Return the slope of fit.

- double **intercept** () const

  Return the intercept of fit.

- double **chiSquared** () const

  Return the chi2 of fit.

- double **sigmaA** () const

  Return error on slope.

- double **sigmaB** () const

  Return error on intercept.

- int **numberOfPoints** () const

  Return number of points to be fit.

- int **rc** () const

  Return code of fit.

- void **addPoint** (double x, double y, double weight)

  Add a point to be used in fit.

- void **clear** ()

  Clear points stored to be fit.

- bool **fit** ()

  Perform the fit.

- void **print** () const

  Stream the points to be fit to the screen.

4.12.1 Detailed Description

Adapted from uitLineFit.c. This class performs a linear-least squares regression in two dimensions. It assumes that there is no error on the x-component and that all error can be projected onto the y-component.

Author:

Jawluen Tang, Physics department, UT-Austin
, J. T. Mitchell - adapted for PHENIX use. Converted to C.
, M. L. Miller - adapted for STAR use, Converted to C++

Examples:

StFastLineFitter_ex.cxx.

Definition at line 27 of file StFastLineFitter.h.

4.12.2 Member Function Documentation

4.12.2.1 void StFastLineFitter::addPoint (double *x*, double *y*, double *weight*) `[inline]`

Add a point to be used in fit.

It is assumed that there is no error on x, that all error can be projected onto the y ordinate.

Definition at line 136 of file StFastLineFitter.h.

4.12.2.2 int StFastLineFitter::rc () const `[inline]`

Return code of fit.

rc = 0: The fit was successful

rc = 1: There were too few points for the fit. Aborted.

rc = 2: There was a zero determinant. Aborted

Definition at line 127 of file StFastLineFitter.h.

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

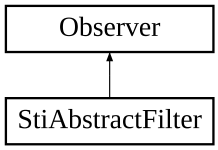
- Sti/**StFastLineFitter.h**

- Sti/**StFastLineFitter.cxx**

4.13 StiAbstractFilter Class Template Reference

`#include <StiAbstractFilter.h>`

Inheritance diagram for StiAbstractFilter::



Public Methods

- **StiAbstractFilter** (Subject ∗s, StiIOBroker ∗b, string name)
  We must pass in valid pointers to Subject and IOBroker.
- virtual ∼**StiAbstractFilter** ()
  We must detach from the subject at destruction time.
- virtual bool **operator()** (T) const=0
  Enforce the interface to the filtering of the object of type T.
- virtual void **getNewState** ()=0
  Enforce a polymorphic update of state when the IOBroker notifies.
- virtual void **print** () const=0
  Print contents of filter to an ostream.

Protected Attributes

- StiIOBroker ∗ **mBroker**
- string **mName**

4.13.1 Detailed Description

template<class T> class StiAbstractFilter< T >

StiAbstractFilter is a templated base class meant to provide a common interface for polymorphic filtering of objects. We inherit from Observer and store a pointer to StiIOBroker so that objects states are handled dynamically.

Author:
M.L. Miller (Yale Software)

Definition at line 25 of file StiAbstractFilter.h.

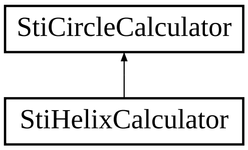
The documentation for this class was generated from the following file:

- Sti/**StiAbstractFilter.h**

4.14 StiCircleCalculator Class Reference

`#include <StiCircleCalculator.h>`

Inheritance diagram for StiCircleCalculator::



Public Methods

- **StiCircleCalculator** ()
- virtual ~**StiCircleCalculator** ()
- virtual void **calculate** (const StThreeVector< double > &pt1, const StThreeVector< double > &pt2, const StThreeVector< double > &pt3)
- double **radius** () const
- double **xCenter** () const
- double **yCenter** () const
- double **probableH** () const

Protected Methods

- void **calculateRadius** (const StThreeVector< double > &pt1, const StThreeVector< double > &pt2, const StThreeVector< double > &pt3)
- void **calculateCenter** (const StThreeVector< double > &pt1, const StThreeVector< double > &pt2, const StThreeVector< double > &pt3)
- void **calculateProbableH** (const StThreeVector< double > &pt1, const StThreeVector< double > &pt2, const StThreeVector< double > &pt3)

Protected Attributes

- **Messenger** & **mMessenger**
- double **mRadius**
- double **mXCenter**
- double **mYCenter**
- double **mProbableH**

4.14.1 Detailed Description

StiCircleCalculator calculates the radius and center of a circle. It is assumed that the circle is in the x-y plane.

Author:

Thomas Ullrich , M.L. Miller (Yale Software)

Note:

StiCircleCalculator is adapted from the class EtCircleCalculator from the Yale Elastic Tracking prototype. Many thanks to Thomas and Brian for the math.

Definition at line 24 of file StiCircleCalculator.h.

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

- Sti/**StiCircleCalculator.h**
- Sti/**StiCircleCalculator.cxx**

4.15 StiCompositeLeafIterator Class Template Reference

`#include <StiCompositeLeafIterator.h>`

Public Types

- typedef **StiCompositeTreeNode**< T > **tnode_t**

  For internal convenience.

- typedef vector< **tnode_t** *> **tnode_vec**

  For internal convenience.

Public Methods

- **StiCompositeLeafIterator** (**tnode_t** *node)

  Only the daughters of **node** *will automatically be found.*

- virtual ~**StiCompositeLeafIterator** ()

  Defualt destructor.

- void **reset** ()

  Reset iterator to point to first leaf.

- **tnode_t** * **operator** * () const

  Dereference iterator, just as an STL iterator.

- void **operator++** ()

  Define only prefix of ++ (only a forward iterator).

- bool **operator!=** (const tnode_vec::const_iterator &)

  Define !=.

- unsigned int **getLeafCount** () const

  Returns the number of leaves found for the tree node passed in constructor.

- tnode_vec::iterator **begin** ()

  Return an iterator marking the beginning of the leaf vector.

- tnode_vec::iterator **end** ()

Return an iterator marking the end of the leaf vector.

- tnode_vec::const_iterator **const_begin** () const

  Return a const_iterator marking the beginning of the leaf vector.

- tnode_vec::const_iterator **const_end** () const

  Return a const_iterator marking the end of the leaf vector.

Protected Methods

- **StiCompositeLeafIterator** ()

  This is not implemented. One must pass a node to the constructor in order for the leaves to be found.

- void **findLeaves** ()

  Internal function used to find leaves. It is called in the constructor.

Protected Attributes

- **tnode_t * mcurrentnode**

  We store a pointer to the root of the tree for internal convenience.

- tnode_vec::const_iterator **mcurrentleaf**

  We have to store an interator into the leaf vector for traversal.

- **tnode_vec mleaves**

  The vector of leaves.

4.15.1 Detailed Description

template<class T> class StiCompositeLeafIterator< T >

StiCompositeLeafIterator is a templated iterator class that is complimenatary to **StiCompositeTreeNode** (p. 47). Given a pointer to a **StiCompositeTreeNode** (p. 47) in its constructor, StiCompositeLeafIterator provides access to all of the leaves of that node. Leaves are defined as nodes that have no daughters, and are actaully found using the templated helper class LeafFinder. StiCompositeLeafIterator stores pointers to the leaves in an internal container and provides limited access to the leaves. Ultimately, StiCompositeLeafIterator should conform to the requirements of (at least) an STL forward iterator, **however it does not yet**. As such, it will currently work when passed to some, but not all, STL algorithms. However, let it be noted that it provides access to const_iterators into the vector of leaves. These are true STL iterators and can be passed to any algorithm that takes const_iterators.

Note that StiCompositeLeafIterator is a "meta" iterator. That is, it is a combination of an iterator and a container. As such, it must provide functionality for both. This is reflected in its providal of operators, e.g., **operator++**() (p. 46), and memberfunctions that denote the container characteristics, specifically **begin**() (p. 42) and **end**() (p. 42). These names have been changed to **const_begin**() (p. 43) and **const_end**() (p. 43) to reflect that they return const_iterators.

Author:
M.L. Miller (Yale Software)

Examples:
StiCompositeLeafIterator_ex.cxx.

Definition at line 45 of file StiCompositeLeafIterator.h.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 template<class T> StiCompositeLeafIterator< T >::StiCompositeLeafIterator (tnode_t ∗ *node*)

Only the daughters of **node** will automatically be found.

The created instance of StiCompositeLeafIterator will find only the leaves that belong to the argument to the constructor call, **node**. That is, the iterator will treat node as if it is the root of a tree. Suppose that node is itself has a parent, and that parent has leaves that exist on a branch other than node. In such a case, these leaves will be ignored by StiCompositeLeafIterator. Therefore, if one wants to find all possible leaves of a given tree, one must be sure that **node** corresponds to the true root of the tree.

Definition at line 126 of file StiCompositeLeafIterator.h.

References findLeaves().

4.15.3 Member Function Documentation

4.15.3.1 template<class T> void StiCompositeLeafIterator< T >::findLeaves () `[protected]`

Internal function used to find leaves. It is called in the constructor.

We provide safe forward access to the vector of leaves. This is a real STL iterator (std::vector::const_iterator) and can be used accordingly. Howver, it **cannot** be used to modify the container of leaves that the iterator points into.

const_end() (p. 43) marks an **invalid** iterator, and it points to one entry **past** the last valid entry in the leaf vector.

Definition at line 269 of file StiCompositeLeafIterator.h.

References mcurrentnode, mleaves, and reset().

Referenced by StiCompositeLeafIterator().

4.15.3.2 template<class T> unsigned int StiCompositeLeafIterator< T >::getLeafCount () const `[inline]`

Returns the number of leaves found for the tree node passed in constructor.

This returns the number of leaves that can be found by following all possible paths downward from the node passed to the constructor.

Definition at line 196 of file StiCompositeLeafIterator.h.

References mleaves.

4.15.3.3 template<class T> StiCompositeLeafIterator< T >::tnode_t * StiCompositeLeafIterator< T >::operator * () const `[inline]`

Dereference iterator, just as an STL iterator.

Suppose that you had declared the following typedefs:

`typedef` **StiCompositeTreeNode** (p. 47)`<Foo> tnode_t;`

`typedef StiCompositeLeafIterator<Foo> tleafit_t;`

And you added the following code:

`tnode_t root;`

`... code to hang other nodes on the root ...`

Then you dereference the leaf iterator as follows:

```
for (tleafit_t it(root); it!=it.const_end(); ++it) {

Foo* myFoo = *it;

}
```

Definition at line 165 of file StiCompositeLeafIterator.h.

4.15.3.4 template<class T> void StiCompositeLeafIterator< T >::operator++ () [inline]

Define only prefix of ++ (only a forward iterator).

This simply increments the iterator to point to the next leaf in the leave vector.

Definition at line 174 of file StiCompositeLeafIterator.h.

References mcurrentleaf.

4.15.3.5 template<class T> void StiCompositeLeafIterator< T >::reset () [inline]

Reset iterator to point to first leaf.

A call to reset does not invalidate the leaves that this iterator corresponds to. Instead, it simple resets the iterator to point to the first leaf in the leaf vector. Therefore, one may call **reset**() (p. 46) with impuninity. However, this also means that a StiCompositeLeafIterator can never be assigned to point to a different collection of leaves. This can only be accomplished by constructing a new instance of StiCompositeLeafIterator that points to a different node.

Definition at line 146 of file StiCompositeLeafIterator.h.

References mcurrentleaf, and mleaves.

Referenced by findLeaves().

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

- Sti/**StiCompositeLeafIterator.h**

4.16 StiCompositeTreeNode Class Template Reference

`#include <StiCompositeTreeNode.h>`

Public Types

- typedef vector< StiCompositeTreeNode *> **vec_type**

  For internal convenience.

- typedef vector< StiCompositeTreeNode *> **StiCompositeTreeNode-Vector**

  For internal convenience.

Public Methods

- **StiCompositeTreeNode** ()

  We provide only a default constructor.

- virtual ~**StiCompositeTreeNode** ()

  Default Destructor.

- void **setName** (const string &)

  Set the name of the node.

- void **setOrderKey** (const **StiOrderKey** &)

  Set the order-key for the node.

- void **setData** (T *)

  Set the data to be hung on the node.

- const string & **getName** () const

  Return the name of the node.

- unsigned int **getChildCount** () const

  Return the number of children that belong to this node.

- StiCompositeTreeNode * **getParent** () const

  Return a (non-const!) pointer to the parent of this node.

- const **StiOrderKey** & **getOrderKey** () const
  Return a reference to the the orderkey of this node.
- T ∗ **getData** () const
  Return a (non-const!) pointer to the data hung on this node.
- virtual void **add** (StiCompositeTreeNode ∗)
  Add a child to this node.
- vec_type::iterator **whereInParent** ()
  Provide the iterator into the parent that can be dereferenced to get this node.
- vec_type::iterator **begin** ()
  Provide random access iterator to the beginning of the vector of children.
- vec_type::iterator **end** ()
  Provide random access iterator to the end of the vector of children.
- vec_type::const_iterator **begin** () const
  Provide const_iterator to the beginning of the vector of children.
- vec_type::const_iterator **end** () const
  Provied const_iterator to the end of the vector of children.
- vec_type::reverse_iterator **rbegin** ()
  Provide reverse iterator to the beginning of the vector of children.
- vec_type::reverse_iterator **rend** ()
  Provide reverse iterator to the end of the vector of children.
- vec_type::const_reverse_iterator **rbegin** () const
  Provide const_reverse_iterator tot he beginning of the vector of children.
- vec_type::const_reverse_iterator **rend** () const
  Provide const_reverse_iterator to the end of the vector of children.

4.16.1 Detailed Description

template<class T> class StiCompositeTreeNode< T >

StiCompositeTreeNode is a templated class that can be used to represent objects in a tree structure. The objects to be organized are stored as T∗ pointers by StiCompositeTreeNode and are accessed via **getData**() (p. 48) and **setData**(T∗)

(p. 50) methods. Additionaly, a StiCompositeTreeNode can have 0 or 1 parent and 0-n daughters. A node with no parent is called a **root**. A node with no daughters is called a **leaf**. Internally, the daughters treated as daughters stored in a vector. As such, StiCompositeTreeNode provides access (via STL iterators) to the bounds of the vector. This has the consequence that traversal of the tree can be performed via recursive calls to the STL algorithms. For such cases one merely needs to define funtors that either perform some action given a node (e.g., stream the node to screen), or evaluate whether a given node (or a comparison between two nodes) satisfies some logical condition. For more, see example in StlUtilities.h.

StiCompositeTreeNode is a special tree-node class in that it was designed with the ability to store daughters in a sorted order, which is especially useful for efficient traversal through the tree. One could manually sort the tree by the data stored, or one can use the **StiOrderKey** (p. 147) typedef. Currently this typedef is set to a double, and one can eager cache the sort-key to avoid calls into the data structure. Because StiCompositeTreeNode provides random-access iterators into the daughters, STL algorithms can easily be used (recursively) to perform tasks on an entire tree (e.g., sort, find, for_each). Many of these are already implemented in StlUtilities.h.

see also: **StiCompositeLeafIterator** (p. 42)

Author:

M.L. Miller (Yale Software)

Note:

A node with no childeren is called a 'leaf', or sometimes a 'data node'. In reality, all StiCompositeTreeNodes **can** hold data. In practice, however, most uses of StiCompositeTreeNode will store valid data only on leaves.

Warning:

StiCompositeTreeNode stores data by pointer, so only objects created by new should be passed as data to the node.

Warning:

It is assumed that StiCompositeTreeNode owns no objects! That means all nodes must be deleted manually deleted by user. This is not a thread safe class.

Examples:

StiCompositeLeafIterator_ex.cxx, and **StiDetectorTreeBuilder_ex.cxx**.

Definition at line 94 of file StiCompositeTreeNode.h.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 template<class T> StiCompositeTreeNode< T >::StiCompositeTreeNode ()

We provide only a default constructor.

When the node is created all members are initialized to deafult values (namely 0 or null) and one must then set all information by hand.

Definition at line 203 of file StiCompositeTreeNode.h.

References StiOrderKey::index, and StiOrderKey::key.

4.16.3 Member Function Documentation

4.16.3.1 template<class T> void StiCompositeTreeNode< T >::add (StiCompositeTreeNode< T > * *newChild*) `[virtual]`

Add a child to this node.

For the sake of consistency in the parent-child relationship of one node to another, you will find no public method setParent(). Instead, this task is performed internally by a call to **add**() (p. 50). Thus, once a node2 is added to node1 via node1->add(node2), node1 is automatically set as the parent of node2, and no intervention by the user is necessary.

Additionally, node2 cannot be added as a daughter to node1 if node2 is already a daughter of node1. In such a case, an internal check in the **add**() (p. 50) method will recognize the situation and return with no action taken.

Definition at line 278 of file StiCompositeTreeNode.h.

References end(), and setParent().

4.16.3.2 template<class T> void StiCompositeTreeNode< T >::setData (T * *val*) `[inline]`

Set the data to be hung on the node.

The pointer to data (T* mData) defaults to 'null' on creation. If no call to **setData**() (p. 50) is made, then mData remains set to 'null'.

Definition at line 231 of file StiCompositeTreeNode.h.

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

- Sti/**StiCompositeTreeNode.h**

4.17 StiDedxCalculator Class Reference

`#include <StiDedxCalculator.h>`

Public Types

- typedef vector< **StiKalmanTrackNode** *> **StiKalmanTrackNodeVec**

  For internal convenience.

- typedef vector< double > **DoubleVec**

  For internal convenience.

Public Methods

- **StiDedxCalculator** ()

  Default Constructor.

- virtual ~**StiDedxCalculator** ()

  Default Destructor.

- void **setFractionUsed** (double)

  Set the truncation fraction.

- void **setDetectorFilter** (StDetectorId detector)
- virtual void **getDedx** (const **StiKalmanTrack** *track, double &dEdx, double &dEdxE, double &nPointsUsed)

  Calculate Dedx for the track.

- StDetectorId **whichDetId** ()
- double **whatUseFraction** ()

4.17.1 Detailed Description

StiDedxCalculator is a simple utility class that, given an **StiTrack** (p. 166), calculates the ionization (dedx) via a truncated mean measurement. The truncated mean method uses the following algorithm:

1) The ionization samples (hits) are retrieved from the track.

2) For each hit the charge (de) must be associated with the pathlength (dx) of the particle in the active volume of the detector. From this association we define de/dx_i, where i represents an index corresponding to a given hit.

3) The hits (dedx_i) are placed in a vector and sorted in ascending order.

4) A fraction 'f' of the hits are kept, and 1-f is truncated from the vector.

5) Of the remaining samples, we calculate the mean value <dedx>.

Author:
C. Pruneau , M.L. Miller (Yale Software) , B. Norman (Kent State)

Definition at line 39 of file StiDedxCalculator.h.

4.17.2 Member Function Documentation

4.17.2.1 void StiDedxCalculator::getDedx (const StiKalmanTrack * *track*, double & *dEdx*, double & *dEdxE*, double & *nPointsUsed*) `[virtual]`

Calculate Dedx for the track.

Dedx represents the mean dedx of those hits that are specified by the truncation fraction (see **setFractionUsed**(double) (p. 52)). Dedx has units of KeV/cm.

Definition at line 32 of file StiDedxCalculator.cxx.

References StiKalmanTrack::getNodes().

4.17.2.2 void StiDedxCalculator::setFractionUsed (double *val*) `[inline]`

Set the truncation fraction.

The truncation fraction is a number between 0. and 1. If a track has n samples associated with it (n hits) then the truncation fraction represents the fraction of n **that is used** to calculate the ionization. For example, if a track has n=10 hits, then setting the truncation fraction to .7 corresoponds to using 7 measurements with the lowest dedx per hit.

Definition at line 102 of file StiDedxCalculator.h.

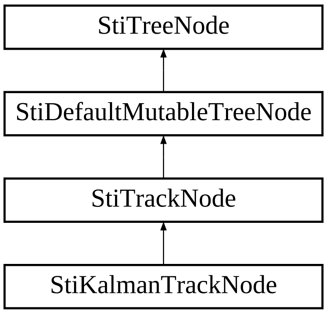
The documentation for this class was generated from the following files:

- Sti/**StiDedxCalculator.h**
- Sti/**StiDedxCalculator.cxx**

4.18 StiDefaultMutableTreeNode Class Reference

#include <StiDefaultMutableTreeNode.h>

Inheritance diagram for StiDefaultMutableTreeNode::



Public Methods

- virtual ~**StiDefaultMutableTreeNode** ()
- **StiDefaultMutableTreeNode** ()

  Creates a tree node that has no parent and no children, but which allows children.

- void **reset** ()
- void **setAsCopyOf** (const StiDefaultMutableTreeNode ∗node)
- void **insert** (StiTreeNode ∗newChild, int childIndex)

  Removes `newChild` *from its present parent (if it has a parent), sets the child's parent to this node, and then adds the child to this node's child array at index* `childIndex`*.* `newChild` *must not be null and must not be an ancestor of this node.*

  Parameters:

  newChild *the StiTreeNode ∗ to insert under this node*

  childIndex *the index in this node's child array where this node is to be inserted*

  See also:

  isNodeDescendant.

- void **remove** (int childIndex)

  Removes the child at the specified index from this node's children and sets that node's parent to 0. The child node to remove must be a `StiTreeNode` *∗.*

  Parameters:

  childIndex *the index in this node's child array of the child to remove*

Exceptions:
  ArrayIndexOutOfBoundsException *if* `childIndex` *is out of bounds.*

- void **setParent** (StiTreeNode *newParent)

  Sets this node's parent to `newParent` *but does not change the parent's child array. This method is called from* **insert()** (p. 53) *and* **remove()** (p. 53) *to reassign a child's parent, it should not be messaged from anywhere else.*
  Parameters:
  newParent *this node's new parent.*

- StiTreeNode * **getParent** ()

  Returns this node's parent or 0 if this node has no parent.
  Returns:
  this node's parent TreeNode, or 0 if this node has no parent.

- StiTreeNode * **getChildAt** (int index) const

  Returns the child at the specified index in this node's child array.
  Parameters:
  index *an index into this node's child array is out of bounds*
  Returns:
  the StiTreeNode in this node's child array at the specified index.

- int **getChildCount** () const

  Returns the number of children of this node.
  Returns:
  an int giving the number of children of this node.

- int **getIndex** (StiTreeNode *aChild)

  Returns the index of the specified child in this node's child array. If the specified node is not a child of this node, returns `-1`*. This method performs a linear search and is O(n) where n is the number of children.*
  Parameters:
  aChild *the StiTreeNode to search for among this node's children*
  Returns:
  an int giving the index of the node in this node's child array, or `-1` *if the specified node is a not a child of this node.*

- bool **getAllowsChildren** ()
- void **removeFromParent** ()
- void **remove** (StiTreeNode *aChild)
- void **removeAllChildren** ()
- void **removeAllChildrenBut** (StiTreeNode *aChild)
- void **add** (StiTreeNode *newChild)
- bool **isNodeAncestor** (StiTreeNode *anotherNode)
- bool **isNodeDescendant** (StiDefaultMutableTreeNode *anotherNode)

- StiTreeNode * **getSharedAncestor** (StiDefaultMutableTreeNode *aNode)
- bool **isNodeRelated** (StiDefaultMutableTreeNode *aNode)
- int **getLevel** ()
- StiTreeNode * **getRoot** ()
- bool **isRoot** ()
- StiDefaultMutableTreeNode * **getNextNode** ()
- StiDefaultMutableTreeNode * **getPreviousNode** ()
- bool **isNodeChild** (StiTreeNode *aNode)
- StiTreeNode * **getFirstChild** ()
- StiTreeNode * **getLastChild** ()
- StiTreeNode * **getChildAfter** (StiTreeNode *aChild)
- StiTreeNode * **getChildBefore** (StiTreeNode *aChild)
- bool **isNodeSibling** (StiTreeNode *anotherNode)
- int **getSiblingCount** ()
- StiDefaultMutableTreeNode * **getNextSibling** ()
- StiDefaultMutableTreeNode * **getPreviousSibling** ()
- bool **isLeaf** ()
- StiDefaultMutableTreeNode * **getFirstLeaf** ()
- StiDefaultMutableTreeNode * **getLastLeaf** ()
- StiDefaultMutableTreeNode * **getNextLeaf** ()
- StiDefaultMutableTreeNode * **getPreviousLeaf** ()
- StiDefaultMutableTreeNodeVector * **breadthFirstEnumeration** ()
- void **appendChildrenToVector** (StiDefaultMutableTreeNode *node, StiDefaultMutableTreeNodeVector *v)

Protected Attributes

- StiTreeNode * **parent**
- StiDefaultMutableTreeNodeVector **children**

4.18.1 Detailed Description

A `StiDefaultMutableTreeNode` is a general-purpose node in a tree data structure. A tree node may have at most one parent and 0 or more children. `StiDefaultMutableTreeNode` provides operations for examining and modifying a node's parent and children and also operations for examining the tree that the node is a part of. A node's tree is the set of all nodes that can be reached by starting at the node and following all the possible links to parents and children. A node with no parent is the root of its tree; a node with no children is a leaf. A tree may consist of many subtrees, each node acting as the root for its own subtree.

This class provides enumerations for efficiently traversing a tree or subtree in various orders or for following the path between two nodes.

This is not a thread safe class. If you intend to use a StiDefaultMutableTreeNode (or a tree of TreeNodes) in more than one thread, you need to do your own synchronizing. A good convention to adopt is synchronizing on the root node of a tree.

While StiDefaultMutableTreeNode implements the MutableTreeNode interface and will allow you to add in any implementation of MutableTreeNode not all of the methods in StiDefaultMutableTreeNode will be applicable to all StiTreeNodes implementations. Especially with some of the enumerations that are provided, using some of these methods assumes the StiDefaultMutableTreeNode contains only StiDefaultMutableNode instances. All of the TreeNode/MutableTreeNode methods will behave as defined no matter what implementations are added.

See also:
StiTreeNode

Author:
Claude Pruneau

Definition at line 56 of file StiDefaultMutableTreeNode.h.

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

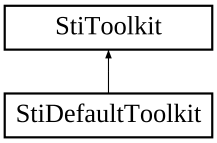
- Sti/**StiDefaultMutableTreeNode.h**
- Sti/**StiDefaultMutableTreeNode.cxx**

4.19 StiDefaultToolkit Class Reference

Definition of toolkit.

`#include <StiDefaultToolkit.h>`

Inheritance diagram for StiDefaultToolkit::



Public Methods

- virtual StiObjectFactoryInterface< **StiHit** > ∗ **getHitFactory** ()
- virtual StiObjectFactoryInterface< **StiKalmanTrack** > ∗ **getTrackFactory** ()
- virtual StiObjectFactoryInterface< StiMcTrack > ∗ **getMcTrackFactory** ()
- virtual StiObjectFactoryInterface< StiDetector > ∗ **getDetectorFactory** ()
- virtual StiObjectFactoryInterface< StiDetectorNode > ∗ **getDetectorNodeFactory** ()
- virtual StiObjectFactoryInterface< **StiKalmanTrackNode** > ∗ **getTrackNodeFactory** ()
- virtual StiObjectFactoryInterface< Parameter > ∗ **getParameterFactory** ()
- virtual StiObjectFactoryInterface< **StiTrackFilter** > ∗ **getTrackFilterFactory** ()
- virtual **StiDetectorContainer** ∗ **getDetectorContainer** ()
- virtual **StiHitContainer** ∗ **getHitContainer** ()
- virtual **StiTrackContainer** ∗ **getTrackContainer** ()
- virtual **StiTrackContainer** ∗ **getMcTrackContainer** ()
- virtual StiGeometryTransform ∗ **getGeometryTransform** ()
- virtual StiCoordinateTransform ∗ **getCoordinateTransform** ()
- virtual StiDetectorFinder ∗ **getDetectorFinder** ()
- virtual StiSeedFinder ∗ **getTrackSeedFinder** ()
- virtual **StiTrackFinder** ∗ **getTrackFinder** ()
- virtual StiTrackFitter ∗ **getTrackFitter** ()
- virtual **StiTrackMerger** ∗ **getTrackMerger** ()
- virtual StiDisplayManager ∗ **getDisplayManager** ()

- virtual StiIOBroker ∗ **getIOBroker** ()
- virtual **StiHitFiller** ∗ **getHitFiller** ()
- virtual StAssociationMaker ∗ **getAssociationMaker** ()
- virtual void **setAssociationMaker** (StAssociationMaker ∗a)
- virtual **StiHitErrorCalculator** ∗ **getHitErrorCalculator** ()

Protected Methods

- **StiDefaultToolkit** ()
- virtual ∼**StiDefaultToolkit** ()

Protected Attributes

- StiObjectFactoryInterface< **StiTrackFilter** > ∗ **trackFilterFactory**
- StiObjectFactoryInterface< Parameter > ∗ **parameterFactory**
- StiObjectFactoryInterface< **StiHit** > ∗ **hitFactory**
- StiObjectFactoryInterface< **StiKalmanTrack** > ∗ **trackFactory**
- StiObjectFactoryInterface< StiMcTrack > ∗ **mcTrackFactory**
- StiObjectFactoryInterface< StiDetector > ∗ **detectorFactory**
- StiObjectFactoryInterface< StiDetectorNode > ∗ **detectorNodeFactory**
- StiObjectFactoryInterface< **StiKalmanTrackNode** > ∗ **trackNodeFactory**
- **StiDetectorContainer** ∗ **detectorContainer**
- **StiHitContainer** ∗ **hitContainer**
- **StiTrackContainer** ∗ **trackContainer**
- **StiTrackContainer** ∗ **mcTrackContainer**
- StiGeometryTransform ∗ **geometryTransform**
- StiCoordinateTransform ∗ **coordinateTransform**
- StiDetectorFinder ∗ **detectorFinder**
- StiSeedFinder ∗ **trackSeedFinder**
- **StiTrackFinder** ∗ **trackFinder**
- StiTrackFitter ∗ **trackFitter**
- **StiTrackMerger** ∗ **trackMerger**
- StiDisplayManager ∗ **displayManager**
- StiIOBroker ∗ **ioBroker**
- **StiHitFiller** ∗ **hitFiller**
- StAssociationMaker ∗ **associationMaker**
- **StiHitErrorCalculator** ∗ **hitErrorCalculator**

Friends

- class **StiToolkit**

4.19.1 Detailed Description

Definition of toolkit.

Definition at line 24 of file StiDefaultToolkit.h.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 StiDefaultToolkit::StiDefaultToolkit () `[protected]`

Protected ctor

Definition at line 54 of file StiDefaultToolkit.cxx.

4.19.2.2 StiDefaultToolkit::~StiDefaultToolkit () `[protected, virtual]`

Dtor. Must delete all objects currently allocated.

Definition at line 81 of file StiDefaultToolkit.cxx.

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

- StiMaker/**StiDefaultToolkit.h**
- StiMaker/**StiDefaultToolkit.cxx**

4.20 StiDetectorContainer Class Reference

`#include <StiDetectorContainer.h>`

Public Methods

- virtual void **buildDetectors** (StiObjectFactoryInterface< StiDetectorNode > ∗nodefactory, StiObjectFactoryInterface< StiDetector > ∗detfactory)

  This calls builds the detector tree given a pointer to the necessaryfactories.

- const data_node ∗ **root** () const
- void **reset** ()

  This performs a full internal reset of interator structure.

- StiDetector ∗ **operator** ∗ () const

  Dereference current iterator and return a pointer to current StiDetector.

- bool **moveOut** ()

  Step out radially in STAR TPC global coordinates.

- bool **moveIn** ()

  Step in radially in STAR TPC global coordinates.

- void **movePlusPhi** ()

  Step around in increasing phi.

- void **moveMinusPhi** ()

  Step around in decreasing phi.

- void **setToDetector** (const StiDetector ∗layer)

  Set iterators to the detector nearest to the passed StiDetector pointer.

- void **setToDetector** (double position)

  Set iterators to the first detector in the radial layer closest to thespecified position.

- void **setToDetector** (double position, double angle)

  Set iterators to the detector closest to the given position and angle.

- void **print** () const

  This is not currently implemented.

- virtual ~**StiDetectorContainer** ()

  Private destructor: implementation of singleton pattern.

- **StiDetectorContainer** ()

  Private destructor: implementation of singleton pattern.

Static Public Methods

- StiDetectorContainer * **instance** ()

  Access to singleton instance.

- void **kill** ()

  Kill the current instance. Use this wisely!

Friends

- class **nobody**

  We inclue this to avoid compiler warnings generated because of the singleton design pattern.

4.20.1 Detailed Description

StiDetectorContainer is an interface to the representation of the STAR detector material. It is an implementation of the 'facade' pattern. That is, it is meant to provide an unchanging interface to the detector model while the actual underlying reperesntation of the detector itself can change. In reality, the underlying model has undergone at least five significant changes, while the public interface of StiDetectorContainer has remained constant.

Because there is only one STAR detector, there is also only one instance of StiDetectorContainer. This is guarunteed by implementing StiDetectorContainer via the singelton design pattern. See the example below for more information on singleton access.

StiDetectorContainer behaves as an iterator. That is, once built it always points to a valid StiDetector object, which can be accessed via: `*(`**StiDetectorContainer::instance**`()` (p. 61)`)`. One can set the location of the current detector position via the **setToDetector**() (p. 65) methods.

Internally, the STAR detector is modeld as a sorted tree structurere implemented via **StiCompositeTreeNode** (p. 47) objects. Additionally, StiDetectorContainer uses an instance of **StiCompositeLeafIterator** (p. 42) to implement the **setToDetector**() (p. 65) methods. However, the navigation methods (e.g., **moveIn**() (p. 63)) are implemented by using the sorted nature of the tree structure. As such, **moveIn**() (p. 63), **moveOut**() (p. 63), **movePlusPhi**() (p. 64), and **moveMinusPhi**() (p. 63) require no searching or expensive computation. Instead, they are implemented via simple increment (++) or decrement (–) of STL random access iterators provided by **StiCompositeTreeNode** (p. 47). Therefore, once StiDetectorContainer is initialized for propogation via a **setToDetector**() (p. 65) call, navigation should be extremely efficient.

Author:

M.L. Miller (Yale Software)

Warning:

You do not have to call **StiDetectorContainer::kill**() (p. 61) to avoid a memory leak. When you call **kill**() (p. 61), you invalidate all pre-existing pointers to **instance**() (p. 61). Because termination of program execution will automatically clean up the heap, it **is** generally good practice not to call **kill**() (p. 61).

Examples:

StiDetectorContainer_ex.cxx.

Definition at line 79 of file StiDetectorContainer.h.

4.20.2 Member Function Documentation

4.20.2.1 void StiDetectorContainer::buildDetectors (StiObjectFactoryInterface< StiDetectorNode > * *nodefactory*, StiObjectFactoryInterface< StiDetector > * *detfactory*) `[virtual]`

This calls builds the detector tree given a pointer to the necessaryfactories.

Recursively load all detector definition files from the given directory. There is internal protection to avoid building the detector representation more than once.

Definition at line 254 of file StiDetectorContainer.cxx.

References StiCompositeTreeNode< data_t >::begin(), StiDetectorTreeBuilder::build(), StiCompositeTreeNode< data_t >::end(), and reset().

4.20.2.2 bool StiDetectorContainer::moveIn ()

Step in radially in STAR TPC global coordinates.

A call to **moveIn**() (p. 63) may not always alter the StiDetector to which the container points. Notably, if there is nowhere else to 'move in to', then **moveIn**() (p. 63) will have no action. So, to see if the action succeeded, one must store a pointer to the StiDetector represented by the current state of the container, call **moveIn**() (p. 63), and then check that the pointer to the StiDetector represented by the new state of the container is different than that of the previous state.

Additionally, when a call to **moveIn**() (p. 63) is made, the container 'selects' the StiDetector object that is closest in phi to the StiDetector object that is being 'movedIn' from. Therefore, a call to **moveIn**() (p. 63) usually need not be followed by a call to **movePlusPhi**() (p. 64) or **moveMinusPhi**() (p. 63), except in cases of extreme assymetry, such as navigation through the Silicon Vertex Tracker.

Definition at line 147 of file StiDetectorContainer.cxx.

References StiCompositeTreeNode< data_t >::begin(), StiCompositeTreeNode::getChildCount(), StiCompositeTreeNode< data_t >::getOrderKey(), StiCompositeTreeNode< data_t >::getParent(), and StiOrderKey::index.

4.20.2.3 void StiDetectorContainer::moveMinusPhi ()

Step around in decreasing phi.

Minus phi is defined as counter-clockwise if viewing sectors 1-12 from the membrane, decreasing phi in STAR TPC global coordinates. A call to **moveMinusPhi**() (p. 63) will always have a valid action. That is, the call will wrap around past 2pi.

Definition at line 241 of file StiDetectorContainer.cxx.

4.20.2.4 bool StiDetectorContainer::moveOut ()

Step out radially in STAR TPC global coordinates.

A call to **moveOut**() (p. 63) may not always alter the StiDetector to which the container points. Notably, if there is nowhere else to 'move out to', then **moveOut**() (p. 63) will have no action. So, to see if the action succeeded, one must store a pointer to the StiDetector represented by the current state of the container, call **moveOut**() (p. 63), and then check that the pointer to the StiDetector represented by the new state of the container is different than that of the previous state.

Additionally, when a call to **moveOut**() (p. 63) is made, the container 'selects' the StiDetector object that is closest in phi to the StiDetector object that is being 'movedOut' from. Therefore, a call to **moveIn**() (p. 63) usually need not be followed by a call to **movePlusPhi**() (p. 64) or **moveMinusPhi**() (p. 63), except in cases of extreme assymetry, such as navigation through the Silicon Vertex Tracker.

Definition at line 199 of file StiDetectorContainer.cxx.

References StiCompositeTreeNode< data_t >::begin(), StiCompositeTreeNode< data_t >::end(), StiCompositeTreeNode::getChildCount(), StiCompositeTreeNode< data_t >::getOrderKey(), StiCompositeTreeNode< data_t >::getParent(), and StiOrderKey::index.

4.20.2.5 void StiDetectorContainer::movePlusPhi ()

Step around in increasing phi.

Plus phi is defined as clockwise if viewing sectors 1-12 from the membrane, increasing phi in STAR TPC global coordinates. A call to **movePlusPhi**() (p. 64) will always have a valid action. That is, the call will wrap around past 2pi.

Definition at line 228 of file StiDetectorContainer.cxx.

4.20.2.6 void StiDetectorContainer::reset ()

This performs a full internal reset of interator structure.

A call to reset simply sets the pointer to the default StiDetector object. It does not alter the state of the detector model.

Definition at line 122 of file StiDetectorContainer.cxx.

References StiCompositeTreeNode< data_t >::begin().

Referenced by buildDetectors().

4.20.2.7 void StiDetectorContainer::setToDetector (double *radius*, double *angle*)

Set iterators to the detector closest to the given position and angle.

If no detector exists at this position, the iterator is set to the innermost layer closest to phi=0, s.t. phi>0.

Definition at line 84 of file StiDetectorContainer.cxx.

References StiOrderKey::key, and setToDetector().

4.20.2.8 void StiDetectorContainer::setToDetector (const StiDetector ∗ *layer*)

Set iterators to the detector nearest to the passed StiDetector pointer.

This is used, e.g., to set the iterators to a certain point in preparation for propogation of a new track. If no StiDetector pointer is found that is equal to **layer**, then an error message is streamed to the screen and **reset**() (p. 64) is called.

Definition at line 108 of file StiDetectorContainer.cxx.

Referenced by StiEvaluableTrackSeedFinder::makeTrack(), and setToDetector().

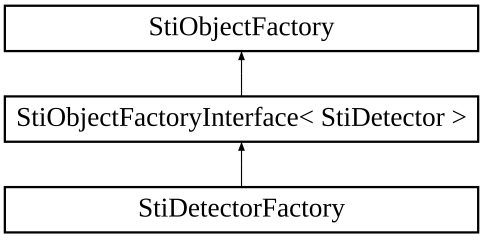
The documentation for this class was generated from the following files:

- Sti/**StiDetectorContainer.h**
- Sti/**StiDetectorContainer.cxx**

4.21 StiDetectorFactory Class Reference

`#include <StiFactoryTypes.h>`

Inheritance diagram for StiDetectorFactory::



Public Methods

- **StiDetectorFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

  This is the only constructor available.

- virtual ~**StiDetectorFactory** ()

  Default destructor.

Protected Methods

- virtual void ∗ **makeNewObject** () const

  Return a pointer to a new StiDetector object on the heap.

4.21.1 Detailed Description

StiDetectorFactory is derived from the abstract factory interface class StiObjectFactoryInterface<StiDetector>. This class has the sole purpose of creating an StiDetector object on the heap for each call to **makeNewObject**() (p. 66).

Author:
M.L. Miller (Yale Software)

Examples:
StiDetectorContainer_ex.cxx.

Definition at line 108 of file StiFactoryTypes.h.

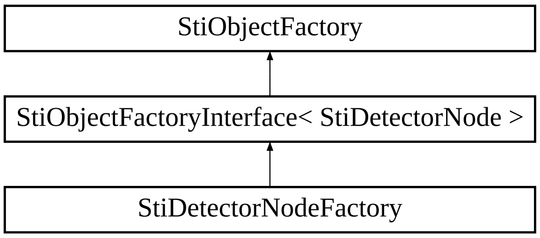
The documentation for this class was generated from the following files:

- Sti/**StiFactoryTypes.h**
- Sti/**StiFactoryTypes.cxx**

4.22 StiDetectorNodeFactory Class Reference

#include <StiFactoryTypes.h>

Inheritance diagram for StiDetectorNodeFactory::



Public Methods

- **StiDetectorNodeFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

  This is the only constructor available.

- virtual ~**StiDetectorNodeFactory** ()

  Default destructor.

Protected Methods

- virtual void ∗ **makeNewObject** () const

  Return a pointer to a new **StiCompositeTreeNode** (p. 47)*<StiDetector> on the heap.*

4.22.1 Detailed Description

StiHitFactory (p. 101) is derived from the abstract factory interface class StiObjectFactoryInterface<**StiCompositeTreeNode** (p. 47)<StiDetector>>. This class has the sole purpose of creating an **StiCompositeTreeNode** (p. 47)<StiDetector> object on the heap for each call to **makeNewObject**() (p. 68). It also contains some typedefs that are hidden from CINT (via preprocessor catch).

Author:

M.L. Miller (Yale Software)

Examples:
StiDetectorContainer_ex.cxx, and **StiDetectorTreeBuilder_ex.cxx**.

Definition at line 176 of file StiFactoryTypes.h.

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

- Sti/**StiFactoryTypes.h**
- Sti/**StiFactoryTypes.cxx**

4.23 StiDetectorTreeBuilder Class Reference

`#include <StiDetectorTreeBuilder.h>`

Public Methods

- **StiDetectorTreeBuilder** ()

  Default Contstructor.

- virtual ~**StiDetectorTreeBuilder** ()

  Default Destructor.

- data_node ∗ **build** (StiObjectFactoryInterface< StiDetectorNode > ∗nodefactory, StiObjectFactoryInterface< StiDetector > ∗detfactory)

  Build the Detector model.

4.23.1 Detailed Description

StiDetectorTreeBuilder is a utility class that uses objects it gets from two factories to build a full model of the STAR detector material. StiDetectorTreeBuilder is the class responsible for actually organizing the StiDetector objects into a tree structure. As such, it uses the utility class StiDetectorBuilder to generate StiDetector objects, and then these objects are organized as belonging to an **StiCompositeTreeNode** (p. 47)<StiDetector> object. This is all accomplished via the call to **build**() (p. 71).

The general flow of execution is as follows. First, in the constructor of StiDetectorTreeBuilder, the member mDetectorBuilder is set to point to an instance of StiCodedDetectorBuilder created on the heap. Once a call to **build**() (p. 71) is made, the tree is assembled by looping on detectors that are generated by mDetectorBuilder. Each detector object returned by the mDetectorBuilder is then hung on the tree by a call to addToTree which calls hangWhere(). By using mDetectorBuilder polymorphically, StiDetectorTreeBuilder becomes extremely flexible. That is, it does not care how the StiDetector objects are created (e.g., from root macro, data base, or geant). However, to really take advantage of this flexibility one should remove ownership of mDetectorBuilder from StiDetectorTreeBuilder and instead set the polymorphic pointer by hand before a call to **build**() (p. 71). This is work to be done.

Author:

M.L. Miller (Yale Software)

Warning:

There is **some** internal protection against build being called more then once. See the documenation for the **build**() (p. 71) method.
Member of type StiDetectoBuilder∗ defaults to StiCodedDetectorBuilder.

Examples:

StiDetectorTreeBuilder_ex.cxx.

Definition at line 59 of file StiDetectorTreeBuilder.h.

4.23.2 Member Function Documentation

4.23.2.1 data_node ∗ StiDetectorTreeBuilder::build (StiObjectFactoryInterface< StiDetectorNode > ∗ *nodefactory*, StiObjectFactoryInterface< StiDetector > ∗ *detfactory*)

Build the Detector model.

There is **some** internal protection against building more than one instance of the detector model. That is, StiDetectorTreeBuilder stores a pointer to the root of the tree that it builds. **build**() (p. 71) first checks that this pointer is null. If not, the call to **build**() (p. 71) will return 0. However, once the StiDetectorTreeBuilder instance that originally built the tree goes out of scope, so does the stored pointer to the root, and thus the protection becomes impossible. Additionaly, the root of the tree is assumed to be owned by the nodefactory.

Definition at line 36 of file StiDetectorTreeBuilder.cxx.

Referenced by StiDetectorContainer::buildDetectors().

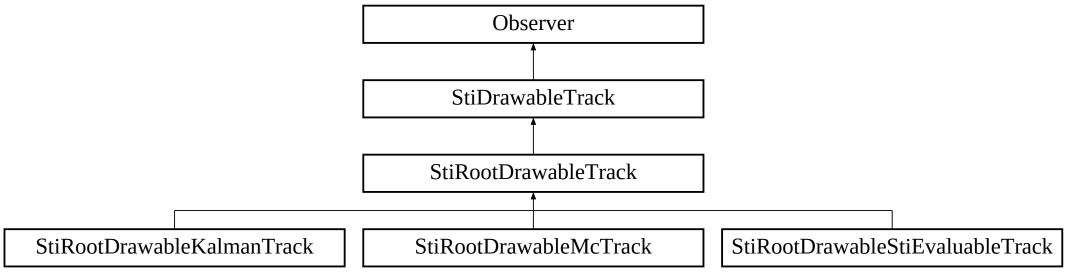
The documentation for this class was generated from the following files:

- Sti/**StiDetectorTreeBuilder.h**
- Sti/**StiDetectorTreeBuilder.cxx**

4.24 StiDrawableTrack Class Reference

`#include <StiDrawableTrack.h>`

Inheritance diagram for StiDrawableTrack::



Public Methods

- **StiDrawableTrack** ()
- virtual ~**StiDrawableTrack** ()
- virtual void **getNewState** ()=0
- virtual void **fillHitsForDrawing** ()=0
- virtual void **update** ()
- virtual void **reset** ()=0

4.24.1 Detailed Description

Abstract base class used to define the interface to drawable tracks

Author:
Claude A Pruneau

Definition at line 11 of file StiDrawableTrack.h.

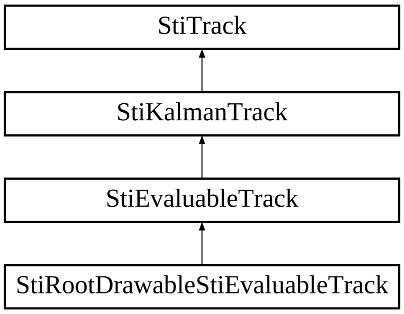
The documentation for this class was generated from the following files:

- Sti/**StiDrawableTrack.h**
- Sti/**StiDrawableTrack.cxx**

4.25 StiEvaluableTrack Class Reference

#include <StiEvaluableTrack.h>

Inheritance diagram for StiEvaluableTrack::



Public Methods

- **StiEvaluableTrack** ()
- virtual ~**StiEvaluableTrack** ()
- void **setStTrackPairInfo** (StTrackPairInfo ∗)

  Set StTrackPairInfo pointer member.

- StTrackPairInfo ∗ **stTrackPairInfo** () const

  Get a pointer to StTrackPairInfo object.

- virtual void **reset** ()

  Reset the class members to default state.

Protected Attributes

- StTrackPairInfo ∗ **mPair**

  A pointer to the union of a monte-carlo track and a StGlobalTrack.

4.25.1 Detailed Description

StiEvaluableTrack is a class that is used to evaluate the tracker when run on simulated data. StiEvaluableTrack is a **StiKalmanTrack** (p. 105). Additionally it encapsulates the relationship between the found **StiKalmanTrack** (p. 105), the StMcTrack (monte-carlo) from which it came, and the StGlobalTrack (found by existing STAR tracking) that was deembed to the best match to the monte-carlo track. The best match is decided using the existing STAR package St-AssociationMaker, and the lower bound requirement for the match to be deemed successful are determined by **StiEvaluableTrackSeedFinder** (p. 78).

Author:
M.L. Miller (Yale Software)

Examples:
StiEvaluableTrack_ex.cxx.

Definition at line 28 of file StiEvaluableTrack.h.

4.25.2 Member Function Documentation

4.25.2.1 void StiEvaluableTrack::reset () `[virtual]`

Reset the class members to default state.

This is used so that StiEvaluableTrack objects can be owned and served by Sti-ObjectFactory. It is guarunteed that a call to **reset**() (p. 74) fully propogates up the inheritance tree.

Reimplemented from **StiKalmanTrack** (p. 125).

Reimplemented in **StiRootDrawableStiEvaluableTrack** (p. 156).

Definition at line 26 of file StiEvaluableTrack.cxx.

References mPair, and StiKalmanTrack::reset().

Referenced by StiEvaluableTrackSeedFinder::makeTrack(), and StiRoot-DrawableStiEvaluableTrack::reset().

4.25.2.2 StTrackPairInfo ∗ StiEvaluableTrack::stTrackPairInfo () const `[inline]`

Get a pointer to StTrackPairInfo object.

StTrackPairInfo represents the union of a monte-carlo track and a StGlobalTrac. This association has been made via StAssociationMaker.

Definition at line 62 of file StiEvaluableTrack.h.

References mPair.

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

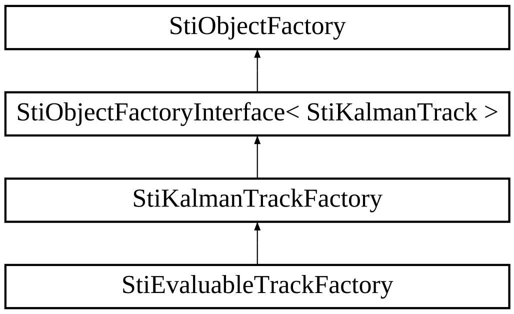
- Sti/**StiEvaluableTrack.h**

- Sti/**StiEvaluableTrack.cxx**

4.26 StiEvaluableTrackFactory Class Reference

`#include <StiFactoryTypes.h>`

Inheritance diagram for StiEvaluableTrackFactory::



Public Methods

- **StiEvaluableTrackFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

  Thi is the only constructor available.

- virtual ~**StiEvaluableTrackFactory** ()

  Default destructor.

Protected Methods

- virtual void * **makeNewObject** () const

  Return a pointer to a new StiEvaluable track on the heap.

4.26.1 Detailed Description

StiEvaluableTrackFactory is derived from the abstract factory interface class StiObjectFactoryInterface<**StiKalmanTrack** (p. 105)>. This class has the sole purpose of creating an **StiEvaluableTrack** (p. 73) object on the heap for each call to **makeNewObject**() (p. 76).

Author:

M.L. Miller (Yale Software)

Definition at line 81 of file StiFactoryTypes.h.

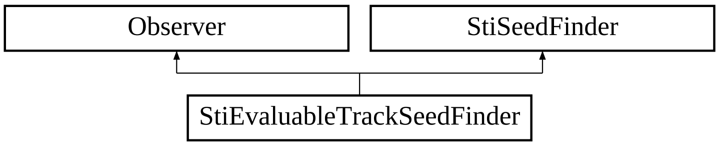
The documentation for this class was generated from the following files:

- Sti/**StiFactoryTypes.h**
- Sti/**StiFactoryTypes.cxx**

4.27 StiEvaluableTrackSeedFinder Class Reference

`#include <StiEvaluableTrackSeedFinder.h>`

Inheritance diagram for StiEvaluableTrackSeedFinder::



Public Methods

- **StiEvaluableTrackSeedFinder** (StAssociationMaker ∗, **StiHitContainer** ∗)

  This is the only constructor available.

- virtual ∼**StiEvaluableTrackSeedFinder** ()

  Default destructor.

- void **setEvent** (StMcEvent ∗mcevt=0)
- virtual bool **hasMore** ()
- virtual **StiKalmanTrack** ∗ **next** ()
- virtual void **reset** ()
- void **getNewState** ()

Protected Methods

- **StiEvaluableTrack** ∗ **makeTrack** (StMcTrack ∗)

  Construct an evaluable track from a m.c. track.

4.27.1 Detailed Description

StiEvaluableTrackSeedFinder is used to provide candidate seeds for the track finder when running on simulated data. These seeds are of type **StiEvaluableTrack** (p. 73), and encapsulate information regarding the monte-carlo track and the StGlobalTrack found by existing STAR tracking. The association is performed via StAssociationMaker. The best match is chosen from all possible matches, and this match must pass a cut on the number of matched points. This cut is explained in **makeTrack**() (p. 80).

StiEvaluableTrackSeedFinder is a StiSeedFinder, so it supports the enforced user interface, namely **hasMore**() (p. 79) and **next**() (p. 80). Additionaly, one can control the nature of the generated seeds. StiEvaluableTrackSeedFinder is, like all StiSeedFinder objects, dynamically buildable. Currently this is done by parsing a text file. However, it is understood that the information necessary for a dynamic build must eventually come from a data base. For more information, see the documentation for setBuildPath() and build() methods.

StiEvaluableTrackSeedFinder requires a valid pointer to an instance of type StAssociationMaker in the constructor call. Contsturctors of other types are explicitly prohibited.

Author:
M.L. Miller (Yale Software)

Examples:
StiEvaluableTrack_ex.cxx.

Definition at line 83 of file StiEvaluableTrackSeedFinder.h.

4.27.2 Constructor & Destructor Documentation

4.27.2.1 StiEvaluableTrackSeedFinder::StiEvaluableTrackSeedFinder (StAssociationMaker ∗ *assoc*, StiHitContainer ∗ *hc*)

This is the only constructor available.

We require a valid pointer to an StAssociationMaker object. All other constructor types are excplicity prohibited. It is assumed, however, that the StAssociationMaker object is owned by some other scope.

Definition at line 45 of file StiEvaluableTrackSeedFinder.cxx.

4.27.3 Member Function Documentation

4.27.3.1 bool StiEvaluableTrackSeedFinder::hasMore () `[virtual]`

A call to **hasMore**() (p. 79) simply checks if there are more seeds to be generated. It does not implement any increment or decrement calls, and thus may be called without ever changing the internal state of the seed finder.

Definition at line 122 of file StiEvaluableTrackSeedFinder.cxx.

Referenced by next().

4.27.3.2 StiEvaluableTrack * StiEvaluableTrackSeedFinder::makeTrack (StMcTrack * *mcTrack*) [protected]

Construct an evaluable track from a m.c. track.

This function is the heart of the seed-finder. It finds the best associated match from the StAssociationMaker instance and initializes the **StiKalmanTrack** (p. 105) state with the parameters from the m.c. track and the hits from the StGlobalTrack.

Definition at line 144 of file StiEvaluableTrackSeedFinder.cxx.

References StiKalmanTrack::getInnerMostNode(), StiDetectorContainer::instance(), StiEvaluableTrack::reset(), StiEvaluableTrack::setStTrackPairInfo(), and StiDetectorContainer::setToDetector().

Referenced by next().

4.27.3.3 StiKalmanTrack * StiEvaluableTrackSeedFinder::next () [virtual]

A call to **next**() (p. 80) constructs a seed from the current m.c. track and increments a pointer to the next available m.c. track. The generation of the seed itself is performed by the private **makeTrack**() (p. 80) method.

Definition at line 131 of file StiEvaluableTrackSeedFinder.cxx.

References hasMore(), and makeTrack().

4.27.3.4 void StiEvaluableTrackSeedFinder::reset () [virtual]

This call is inherited from StiSeedFinder but does not make much sense in the context of evaluable seeds. That is, the internal state cannot be reset without a call to **setEvent**() (p. 80).

Definition at line 80 of file StiEvaluableTrackSeedFinder.cxx.

4.27.3.5 void StiEvaluableTrackSeedFinder::setEvent (StMcEvent * *mcevt* = 0)

This should be called once per event. The call to setEVent internally initializes the seed finder for the event. Without this call, the behavior of **hasMore**() (p. 79) and **next**() (p. 80) is undefined.

Definition at line 88 of file StiEvaluableTrackSeedFinder.cxx.

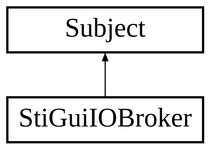
The documentation for this class was generated from the following files:

- Sti/**StiEvaluableTrackSeedFinder.h**
- Sti/**StiEvaluableTrackSeedFinder.cxx**

4.28 StiGuiIOBroker Class Reference

`#include <StiGuiIOBroker.h>`

Inheritance diagram for StiGuiIOBroker::



Public Methods

- void **setUnMarkedHitSize** (double val)

  UnMarked Hits are hits not assigned to tracks.

- double **unMarkedHitSize** () const
- void **setUnMarkedHitColor** (unsigned int val)
- unsigned int **unMarkedHitColor** () const
- void **setUnMarkedHitStyle** (unsigned int)
- unsigned int **unMarkedHitStyle** () const
- void **setMarkedHitSize** (double val)
- double **markedHitSize** () const
- void **setMarkedHitColor** (unsigned int val)
- unsigned int **markedHitColor** () const
- void **setMarkedHitStyle** (unsigned int)
- unsigned int **markedHitStyle** () const
- void **setUpdateEachTrack** (bool)
- bool **updateEachTrack** () const

Static Public Methods

- StiGuiIOBroker $*$ **instance** ()

Friends

- class **nobody**

4.28.1 Detailed Description

StiGuiIOBroker is a simple class that defines a central point for user access to get/set methods for RootDrawable objects. It is available at the cint level, so it provides a dynamic gateway to perform methods such as setting the point size, track line width, etc.

Author:

M.L. Miller (Yale Software)

Note:

Implemented as a singleton.

Definition at line 26 of file StiGuiIOBroker.h.

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

- StiGui/**StiGuiIOBroker.h**
- StiGui/**StiGuiIOBroker.cxx**

4.29 StiHelixFitter Class Reference

`#include <StiHelixFitter.h>`

Public Types

- typedef vector< **StiHit** $*$> **StiHitVector**

  For internal convenience.

Public Methods

- **StiHelixFitter** ()
- virtual ~**StiHelixFitter** ()
- bool **valid** () const

  Return a boolean that signifies whether the information is valid.

- double **xCenter** () const

  Return the xCenter in global coordinates.

- double **yCenter** () const

  Return the yCenter in global coordinates.

- double **z0** () const

  Return the z reference point in global coordinates.

- double **curvature** () const

  *Return the signed curvature. (k*h).*

- double **tanLambda** () const

  Return the value of tanLambda.

- void **reset** ()

  Full internal clear.

- bool **fit** (const **StiHitVector** &)

4.29.1 Detailed Description

StiHelixFitter is a class that performs a fast and approximate fit to a collection of **StiHit** (p. 86) object to return the necessary parameters to initialize an **StiKalmanTrack** (p. 105). These parameters are: curvature, tanLambda, xCenter, yCenter, and z0. For more on these parameters, please see documentation for **StiKalmanTrack::initialize**() (p. 122).

StiHelixFitter uses an instance of StFastCircleFitter and **StFastLineFitter** (p. 35). A circle fit is first performed in the transverse plane, and then a line is performed in the two dimensional plane z vs s_xy, where s_xy is the pathlength along the circle, starting at the innermost point moving outwards.

Author:
M.L. Miller (Yale Software)

Definition at line 40 of file StiHelixFitter.h.

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

- Sti/**StiHelixFitter.h**
- Sti/**StiHelixFitter.cxx**

4.30 StiHit Class Reference

`#include <StiHit.h>`

Public Methods

- **StiHit** ()

  Default constructor.

- ~**StiHit** ()

  Default destructor.

- double **x** () const

  Return the local x value.

- double **y** () const

  Return the local y value.

- double **z** () const

  Return the global z value.

- double **sxx** () const

  Return the (x,x) component of the error matrix.

- double **syy** () const

  Return the (y,y) component of the error matrix.

- double **szz** () const

  Return the (z,z) component of the error matrix.

- double **sxy** () const

  Return the (x,y) component of the error matrix.

- double **sxz** () const

  Return the (x,z) component of the error matrix.

- double **syz** () const

  Return the (y,z) component of the error matrix.

- double **getEloss** ()
- double **refangle** () const

  Return the refAngle of the detector plane from which the hit arose.

- double **position** () const

  Return the position of the detector plane from whcih the hit arose.

- const StiDetector ∗ **detector** () const

  Return a const pointer to the StiDetector object from which the hitarose.

- StiDetector ∗ **detector** ()

  Return a pointer to the StiDetector object from which the hit arose.

- const StMeasuredPoint ∗ **stHit** () const

  Return a const pointer to the StHit object corresponding to this StiHitinstance.

- unsigned int **timesUsed** () const

  Return the number of times this hit was assigned to a track.

- const StThreeVectorF & **globalPosition** () const

  Return a const reference to a StThreeVectorF that denotes the positionof the hit in global STAR coordinates.

- void **set** (double refAngle, double position, double x, double y, double z, double sxx, double sxy, double sxz, double syy, double syz, double szz)

  Set the position and error in one function call.

- void **set** (double refAngle, double position, double x, double y, double z)

  Set the position in one function call.

- void **setX** (double)

  Set the local x value.

- void **setY** (double)

  Set the local y value.

- void **setZ** (double)

  Set the global z value.

- void **setRefangle** (double)

  Set the refAngle of the detector plane from the hit arose.

- void **setPosition** (double)

  Set the position of the detector plane from the hit arose.

- void **setSxx** (double)

  Set the error-matrix components one by one:.

- void **setSyy** (double)
- void **setSzz** (double)
- void **setSxy** (double)
- void **setSxz** (double)
- void **setSyz** (double)
- void **setError** (const StMatrixF &)

  Set the position error matrix for the measurement from an StMatrixFobject.

- void **scaleError** (double)

  Scale the error arbitrarily.

- void **setDetector** (StiDetector ∗)

  Set the pointer to the StiDetector from which the hit arose.

- void **setStHit** (StMeasuredPoint ∗)

  Set the pointer to the corresponding StHit object.

- void **setTimesUsed** (unsigned int)

  Set the number of times used.

- void **reset** ()

4.30.1 Detailed Description

StiHit is a simple class that encapsulates a three dimensional position measurement in the STAR detector. The measurement is represented in a frame that is 'local' to the detector plane from which it arose.

It is assumed that all hits come from a detector that can be composed of discrete planes. Each plane is characterized by two values: position and refAngle. In the mid-rapidity region of STAR (SVT, TPC, EMC, etc) the position corresponds to the magnitude of a vector pointing from the origin to the center of the plane. The refAngle is the azimuthal defined by the aforemention vector and the STAR global x-axis. All hits store the position and refAngle of the detector plane from which they came.

Within each detector plane, the hit is characterized by two more numbers: y and z. The y value corresponds to the distance along the plane (e.g., padrow) w.r.t. the center of the plane. The z value corresponds to the STAR global coordinate.

While it only takes two values (y and z) two specify the hit location within a plane (once the location of the plane is known) StiHit stores one more value: x. This value of x corresponds to the magnitude of a vector pointing from the origin to the detector plane **perpendicular to the plane**. So, for planes with no tilt w.r.t. the origin (e.g., TPC pad-planes), x will be identical to position. Actually, this is not even quite true for tpc hits, as distortion corrections can make x slightly different than position. However, by storing both x and position, we allow for the separation of information that depends only on material location (track fitting) from that which depends only on hit location (track finding).

StiHit stores information that represents a full error matrix. For efficiency purposes this is stored as a collection of discreet doubles instead of a matrix. Because the error matrix is always symmetric, we must store only six values. These are denoted by s_ij, where s_ij corresponds to the (i,j) component of the matrix.

StiHit also stores a pointer to an StHit that it corresponds to. Additionally, StiHit stores a pointer to the StDetector object from which its measurement arose.

Author:

M.L. Miller (Yale Software)

Definition at line 62 of file StiHit.h.

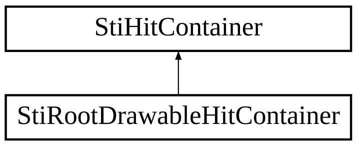
The documentation for this class was generated from the following files:

- Sti/**StiHit.h**
- Sti/**StiHit.cxx**

4.31 StiHitContainer Class Reference

#include <StiHitContainer.h>

Inheritance diagram for StiHitContainer::



Public Methods

- **StiHitContainer** ()

  Implementation of the singleton design pattern.

- virtual ∼**StiHitContainer** ()

  Implementation of the singleton design pattern.

- void **setDeltaD** (double)

  Set the half-width of the search window in distance along the pad.

- void **setDeltaZ** (double)

  Set the half-width of the search window in distance along global z.

- double **deltaD** () const

  Return the value of deltaD (cm).

- double **deltaZ** () const

  Return the value of deltaZ (cm).

- virtual void **update** ()

  Provide for drawable derived class(es?).

- virtual void **push_back** (**StiHit** ∗)

  Add a hit to the container.

- virtual unsigned int **size** () const

  Return the total number of hits in the container.

- virtual void **clear** ()

Clear all hits from the container.

- virtual void **sortHits** ()

  Sort all of the hits in the container.

- void **partitionUsedHits** ()

  Ignore hits marked as used (std::stable_partition).

- const hitvector & **hits** (double refangle, double position)

  Return a const reference to a vector of hits.

- hitvector & **hits** (const StiDetector ∗)

  Return a reference to the vectore of hits, or iterators marking it's bounds.

- hitvector::iterator **hitsBegin** (const StiDetector ∗)
- hitvector::iterator **hitsEnd** (const StiDetector ∗)
- const hitmap & **hits** () const

  Return a const reference ot the hit-vector map.

- void **setRefPoint** (**StiHit** ∗ref)

  Set the reference point to define sub-volume of hits to be accessed.

- void **setRefPoint** (double position, double refAngle, double y, double z)

  Set the reference point to define sub-volume of hits to be accessed.

- bool **hasMore** () const

  Return a boolean that reflects whether there are more hits available in the specified sub-volume.

- **StiHit** ∗ **getCurrentHit** ()

  Return a pointer to the **StiHit** (p. 86) *object currently pointed to from the specified sub-volume.*

- **StiHit** ∗ **getHit** ()

  Return a pointer to the **StiHit** (p. 86) *object currently pointed to from the specified sub-volume.*

- void **addVertex** (**StiHit** ∗)

  Add a vertex to the hit-container.

- unsigned int **numberOfVertices** () const

  Return the number of vertices stored in the container.

- const hitvector & **vertices** () const

  Return a const reference to the a vector of vertices.

Protected Attributes

- **Messenger** & **mMessenger**

Friends

- ostream & **operator**<< (ostream &, const StiHitContainer &)

  Container for primary vertices.

4.31.1 Detailed Description

StiHitContainer is exactly that–a container for StiHits! Because of the sheer number of hits in a STAR event, StiHitContainer is designed to provide efficient access to a subset of hits that are within a user specified volume. This is accomplished by mapping between a two dimensional key and an STL container of hits. This mapping will be discussed in further detail below. There is a natural connection between the hits and the detector from which came. However, for implementation purposes, it is convenient to keep these two entities (HitContainer and DetectorContainer) separate, but keep a well defined method of communication between the two. In such a way one can maintain a HitContainer and DetectorContainer that can exist in different representations. That is, one can have a hit container that is well behaved in local coordinates which map very naturally to the detector model **as well as** a HitContainer that can behave naturally in global coordinates, which do not map naturally to the detector model.

The coordinates of the hits stored are described in **StiHit** (p. 86). It should be noted that the ITTF project treats the STAR TPC as if it were 12 sectors, each extending to +-200 cm. That is, we map hits from sectors 13-24 to a coordinate system defined by sectors 1-12. That way we do not need to make any distinction between hits that come from different sides of the TPC central membrane. This is motivated by the fact that the east and west sectors mark a clear distinction in data taking, but that distinction is unimportant in pattern recognition. For more on the nature of the mapping see StiGeometryTransform.

First we describe the storage of the hits. StiHitContainer treats the hits from a common detector plane (e.g., TPC padrow 13, sector 12) as a sorted std::vector<**StiHit** (p. 86)∗>. The hits are sorted via the functor StizHitLessThan. This functor has a binary predicate that orders hits in a strict less than ordering based upon global z value (see above). Next, StiHitContainer stores these hit-vectors in a std::map<HitMapKey, std::vector<**StiHit** (p. 86)∗>, MapKeyLessThan >. Class HitMapKey is a simple struct that stores two values: refAngle and position, and MapKeyLessThan is a simple struct that defines a stric less-than ordering for HitMapKey objects. These values are described in **StiHit** (p. 86). By specifying a HitMapKey, then, one can achieve **extremely** efficient retrieval of the hit-vector for a given detector plane.

Next we will discuss the retrieval of hits from the container. As stated above, one can gain access to a hit-vector for a given detector plane by specifying the position and refAngle of a detector (see method **hits**(double,double) (p. 95). Additionally, one can access the hit-map itself (or at least a const reference to it!) via the method **hits**() (p. 91). However, as stated before, StiHitContainer is capable of efficient retrievl of a subset of the hits in an event. This subset can be defined as a subset of the hits from a given detector plane. Perhaps it is easier to elucidate via an example. Suppose one is interested in the hits corresponding to TPC sector 12, padrow 13. Then, this sector/padrow combination can be easily mapped to a position and refAngle (see StiGeometryTransform). In this detector one can always specify a 'local' coordinate system where any hit is then fully described by two numbers: local y and z (see **StiHit** (p. 86) for more inforamtion), where y is the distance along the plane (padrow) and z is the global z. Now, suppose one is interested in hits that are within some volume centered at (y0,z0) and bounded by +-deltaD in y and +-deltaZ in z (deltaD is now poorly named–it was meant to represent distance D along a plane). Then, to retrieve the hits from this volume one must call the **setDeltaD**() (p. 96) and **setDeltaZ**() (p. 96) methods to establish the bounds. Then one must call one of the **setRefPoint**() (p. 97) methods. After this, the container has selected the hits within the specified volume, and they can be retreived via the iterator like interface specified by **hasMore**() (p. 91) and **getHit**() (p. 91).

Additionally, StiHitContainer has been modified to provide a similair interface to the primary vertices in a STAR event. Access to the verices is via the methods **addVertex**() (p. 94) and **vertices**() (p. 98). Each vertex is mapped to an **StiHit** (p. 86) object (see StiGeometryTransform) and stored in a hit-vector.

StiHitContainer must be cleared, filled, and sorted for each event. A manual call to **sortHits**() (p. 98) is necessary to achieve the most efficient container implementation. The filling is performed via **StiHitFiller** (p. 103).

Finally, StiHitContainer is implemented via the singelton pattern. This provides a guaruntee that there is only one instance of StiHitContainer. Additionally, it provides for safe global access. Other classes may be derived from StiHitContainer. For more on how to handle this situation, please see the documenation for the instance() method.

Author:

M.L. Miller (Yale Software)

Note:

StiHitContainer does not own the hits that it stores.

Note:

See the documentation of the methods **push_back**() (p. 95) **sortHits**() (p. 98) and **setRefPoint**() (p. 97) for performance guaruntees.

Warning:

struct MapKeyLessThan is used for ordering the HitMapKey objects. For a map, this means that this less-than operation is also used to defined equality. Because this involves operations on doubles that are not guarunteed to be **exactly** identical (the values can come from hits as well as detectors), struct MapKeyLessThan has a built in tolerance that allows for the situation when two HitMapKey objects are actually equal but have very slightly differing values for the doubles they store. These tolerances are currently set in the definition of the struct HitMapKey.

Warning:

You do **not** have to call kill() to avoid a memory leak. A call to kill() will invalidate all existing pointers to StiHitContainer::instance().

Definition at line 148 of file StiHitContainer.h.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 StiHitContainer::StiHitContainer ()

Implementation of the singleton design pattern.

Use this wisely, if at all. See the above warning regarding use of kill().

Definition at line 79 of file StiHitContainer.cxx.

4.31.3 Member Function Documentation

4.31.3.1 void StiHitContainer::addVertex (StiHit * *val*) `[inline]`

Add a vertex to the hit-container.

Each vertex can be mapped to a **StiHit** (p. 86) object (see the documentation for StiGeometryTransform).

Definition at line 342 of file StiHitContainer.h.

4.31.3.2 void StiHitContainer::clear () [virtual]

Clear all hits from the container.

A call to **clear**() (p. 95) must call std::vector<**StiHit** (p. 86)*>clear() for all vectors stored in the map. Thus a call to clear is of O(N) *M where N is the number of keys in the map (see documenation of **hits**() (p. 91) for an estimate of N) and M is the time required to clear each vector.

Definition at line 144 of file StiHitContainer.cxx.

4.31.3.3 const hitvector & StiHitContainer::hits (double *refangle*, double *position*)

Return a const reference to a vector of hits.

The values of refangle and position are used to fill an existing HitMapKey object. This object is used to key the retrieval of a vector of hits associated with the specified position. For more on the definition of refangle and position, please see **StiHit** (p. 86) documentation.

A call to **hits**(double,double) (p. 95) corresponds to the retrieval of an object from an STL map based on a key. Such a retrieval is guarunteed to be of O(logN) time complexity, where N is the number of keys in the map. For our practices, N is roughly equal to (TPC) 12*45 + (SVT layer 1) 2*4 + (SVT layer 2) 2*6 + (SVT layer 3) 2*8 + (SSD) 20.

Definition at line 181 of file StiHitContainer.cxx.

4.31.3.4 void StiHitContainer::push_back (StiHit * *hit*) [virtual]

Add a hit to the container.

The time complexity of push_back has two components:

1) The correct hit-vector must be retrieved (or inserted if it doesn't exist) from the map. This portion is O(logN) where N is the number of hit-vectors in the map. See the documentation of the **hits**() (p. 91) method for an estimate of N.

2) The point must be added to the hit-vector. This is guarunteed to be a constant time process, where the constant is determined by the vendor STL implementation.

Warning:

A call to **push_back**() (p. 95) invalidates the sorted state of the container. Thus, once all hits have been added to the container, then one must call **sortHits**() (p. 98).

Definition at line 119 of file StiHitContainer.cxx.

References StiHit::detector().

4.31.3.5 void StiHitContainer::setDeltaD (double *val*) [inline]

Set the half-width of the search window in distance along the pad.

The distance is specified in STAR units (cm). deltaD corresponds to the distance along the detector plane, e.g., distance along a TPC padrow. If one is interested in points that are within +- 7.2 cm from a given value of local y (see **StiHit** (p. 86)), one would call setDetlaD(7.2). A call to **setDeltaD**() (p. 96) sets the value of deltaD until either:

1) another call to **setDeltaD**() (p. 96) is made or

2) the StiHitContainer instance is deleted.

Definition at line 291 of file StiHitContainer.h.

4.31.3.6 void StiHitContainer::setDeltaZ (double *val*) [inline]

Set the half-width of the search window in distance along global z.

The distance is specified in STAR units (cm). deltaZ corresponds to the distance along z in global STAR coordinations. If one is interested in points that are withing +- 7.2 cm from a given value of global z (see **StiHit** (p. 86)), one would call setDetlaZ(7.2). A call to **setDeltaZ**() (p. 96) sets the value of deltaD until either:

1) another call to **setDeltaZ**() (p. 96) is made or

2) the StiHitContainer instance is deleted.

Definition at line 305 of file StiHitContainer.h.

4.31.3.7 void StiHitContainer::setRefPoint (double *position*, double *refAngle*, double *y*, double *z*)

Set the reference point to define sub-volume of hits to be accessed.

This form of setRefPoint packs the information passed as arguments into a member **StiHit** (p. 86) object, and passes this to the method **setRefPoint**(StiHit*) (p. 97).

Definition at line 223 of file StiHitContainer.cxx.

References StiHit::reset(), StiHit::setPosition(), StiHit::setRefangle(), setRefPoint(), StiHit::setY(), and StiHit::setZ().

4.31.3.8 void StiHitContainer::setRefPoint (StiHit ∗ *ref*)

Set the reference point to define sub-volume of hits to be accessed.

The sub-volume is identified by the following algorithm:

1) Identify the detector plane of intereset via the position and refAngle of the **StiHit** (p. 86) pointer passed.

2) Find those hits that satisfy abs(hit->z-z_i)<deltaZ. This is accomplished via a call to the STL algorithms lower_bound and upper_bound.

3) Find thos hits that satisfy abs(hit->y-y_i)<deltaD. This can only be accomplished via a linear search over those hits satisfying condition

Once **setRefPoint**() (p. 97) has been called, one use the iterator like interfaces **hasMore**() (p. 91), **getHit**() (p. 91) and **getCurrentHit**() (p. 91) to access the hits that satisfied the search criterion.

The time complexity of setRefPoint has several components:

1) The correct hit-vector must be retrieved from the map. This is of O(logN) where N is the number of keys in the map (see documentation of **hits**() (p. 91) for an estimate of the size of N).

2) The calls to lower_bound and upper_bound are each of O(logM) where M is the number of hits in the sorted vector.

3) The linear search over those hits that satisfy criterion 2. This is of O(P) where P is the number of points that passed criterion 2.

This algorithm is easily modifiable to perform the search in the opposite order (search in y, then z instead of z, then y). See the source code for the necessary conversion actions.

Definition at line 264 of file StiHitContainer.cxx.

References StiHit::position(), StiHit::refangle(), StiHit::setZ(), StiHit::y(), and StiHit::z().

Referenced by setRefPoint().

4.31.3.9 unsigned int StiHitContainer::size () const `[virtual]`

Return the total number of hits in the container.

The time complexity of size is of O(logN) where N is the number of keys in the map. For an estimate of N please see the documnation for **hits**() (p. 91) method.

Definition at line 160 of file StiHitContainer.cxx.

4.31.3.10 void StiHitContainer::sortHits () [virtual]

Sort all of the hits in the container.

This function calls the STL sort algorithm for each hit-vector in the map. The **StiHit** (p. 86) objects are ordered via the struct StizHitLessThan.

Note:

A call to **push_back**(StiHit∗) (p. 95) invalidates the sorted state of the container. Thus any number of x calls to **push_back**(StiHit∗) (p. 95) must be followed by a call to **sortHits**() (p. 98).

The time complexity of **sortHits**() (p. 98) has two components:

1) The time to access each vector in the map. This is of O(N) where N is the number of keys in the map (see documentation of **hits**() (p. 91) for an estimate of N).

2) The time to sort an individual vector. This is of O(M logM) where M is the number of hits in stored in the vector.

Definition at line 325 of file StiHitContainer.cxx.

4.31.3.11 void StiHitContainer::update () [virtual]

Provide for drawable derived class(es?).

Null implementation. We provide this virtual function for the situation when StiHitContainer::instance() behaves polymorphically, e.g., when it actually points to a StiRootDrawableDetector object. In that situation, a call to **update**() (p. 98) will propogate to the most derived class, allowing that class to perform necessary tasks (e.g., append hits to display).

Definition at line 103 of file StiHitContainer.cxx.

4.31.3.12 const hitvector & StiHitContainer::vertices () const [inline]

Return a const reference to the a vector of vertices.

The vertices are stored in a single std::vector<**StiHit** (p. 86)∗> object.

Definition at line 354 of file StiHitContainer.h.

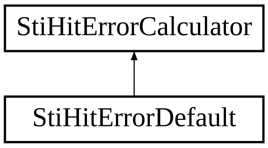
The documentation for this class was generated from the following files:

- Sti/**StiHitContainer.h**
- Sti/**StiHitContainer.cxx**

4.32 StiHitErrorCalculator Class Reference

`#include` <`StiHitErrorCalculator.h`>

Inheritance diagram for StiHitErrorCalculator::



Public Methods

- **StiHitErrorCalculator** ()
- virtual ~**StiHitErrorCalculator** ()
- virtual pair< double, double > **getHitError** (**StiHit** ∗, double, double)=0

4.32.1 Detailed Description

Real errors are generated by fitting and parameterizing a histogram of hit residuals in 3D (radius, dip angle, and drift time). The original parameterization method used in Tpt is determined by StRedsidualMaker and implemented in tpt_hit_uncert, both by Mike Lisa. The method used here closely follows his code.

For details of Mike's analysis of the hit errors, see his web page: `http://www.star.bnl.gov/~lisa/HitErrors` (Mar 14, 2001)

The default error function returns a zero error over all space. This should allow easier error checking for pathalogic and error behavior.

Author:
A. A. Rose (Wayne State)

Date:
2.20.2002

Definition at line 18 of file StiHitErrorCalculator.h.

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

- Sti/**StiHitErrorCalculator.cxx**
- Sti/**StiHitErrorCalculator.h**

4.33 StiHitErrorMaker Class Reference

4.33.1 Detailed Description

The hit errors are ideally derived purely from the geometry of the detector; error(x)=(cell size length in x)/sqrt(12). However, realistic hit errors are often calculated from the hit residuals.

The ITTF hit errors can be assigned using either the default geometric calculation or the real calculated errors. The state of assigned errors is determined from the boolean errorDefault; use of default errors will also generate an error message.

Real errors are generated by fitting and parameterizing a histogram of hit residuals in 3D (radius, dip angle, and drift time). In the future, the user can either pass the hist directly, or pass a fit from such a histogram. The original parameterization method used in Tpt is determined by StRedsidualMaker and implemented in tpt_hit_uncert, both by Mike Lisa. The method used here closely follows his code.

For details of Mike's analysis of the hit errors, see his web page: `http://www.star.bnl.gov/~lisa/HitErrors` (Mar 14, 2001)

Author:
A. A. Rose (Wayne State)

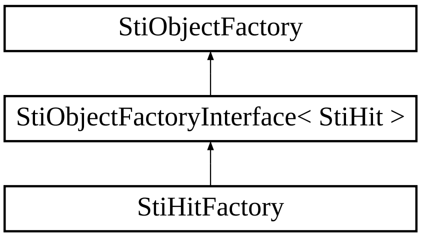
The documentation for this class was generated from the following file:

- Sti/**StiHitError.cxx**

4.34 StiHitFactory Class Reference

`#include <StiFactoryTypes.h>`

Inheritance diagram for StiHitFactory::



Public Methods

- **StiHitFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

  This is the only constructor available.

- virtual ~**StiHitFactory** ()

  Default destructor.

Protected Methods

- virtual void * **makeNewObject** () const

  Return a pointer to a new **StiHit** (p. 86) *object on the heap.*

4.34.1 Detailed Description

StiHitFactory is derived from the abstract factory interface class StiObjectFactoryInterface<**StiHit** (p. 86)>. This class has the sole purpose of creating an **StiHit** (p. 86) object on the heap for each call to **makeNewObject**() (p. 101).

Author:
M.L. Miller (Yale Software)

Definition at line 23 of file StiFactoryTypes.h.

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

- Sti/**StiFactoryTypes.h**
- Sti/**StiFactoryTypes.cxx**

4.35 StiHitFiller Class Reference

`#include <StiHitFiller.h>`

Public Types

- typedef vector< StDetectorId > **det_id_vector**

  Define a collection of enumerated constants.

Public Methods

- **StiHitFiller** ()

  Default constructor.

- virtual ~**StiHitFiller** ()

  Default destructor.

- void **addDetector** (StDetectorId)

  Add detectors from which hits will be used.

- void **setEvent** (StEvent ∗)

  Set a pointer to the next StEvent object.

- void **fillHits** (**StiHitContainer** ∗, StiObjectFactoryInterface< **StiHit** > ∗)

  Fill the hits for a given event.

Friends

- ostream & **operator**<< (ostream &, const StiHitFiller &)

  Friend out a streamer for the class.

4.35.1 Detailed Description

StiHitFiller is a utility class meant to interface between StEvent and **StiHitContainer** (p. 90). StiHitFiller has the sole purpose of transfering the hit collection from StEvent into the **StiHitContainer** (p. 90). This process consists of loopin on the StEvent hit collection, transforming each StHit to an **StiHit** (p. 86), and then adding this hit to the **StiHitContainer** (p. 90).

Author:
M.L. Miller (Yale Software)

Note:
Only hits that come from a detector whose enumeration has been added to StiHitFiller (via **addDetector**(StDetectorId) (p. 104)) will be added to the **StiHitContainer** (p. 90) instance.

Definition at line 45 of file StiHitFiller.h.

4.35.2 Member Function Documentation

4.35.2.1 void StiHitFiller::addDetector (StDetectorId *det*)

Add detectors from which hits will be used.

The enumeration StDetectorId is defined in /pams/global/inc/StDetectorId.h". Only hits that come from a detector whose enumearation has been added via a call to **addDetector**() (p. 104) will be added to the **StiHitContainer** (p. 90).

Definition at line 46 of file StiHitFiller.cxx.

4.35.2.2 void StiHitFiller::setEvent (StEvent ∗ *val*) `[inline]`

Set a pointer to the next StEvent object.

Note:
setEvent must be called before **fillHits**() (p. 103) for every event.

Definition at line 90 of file StiHitFiller.h.

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

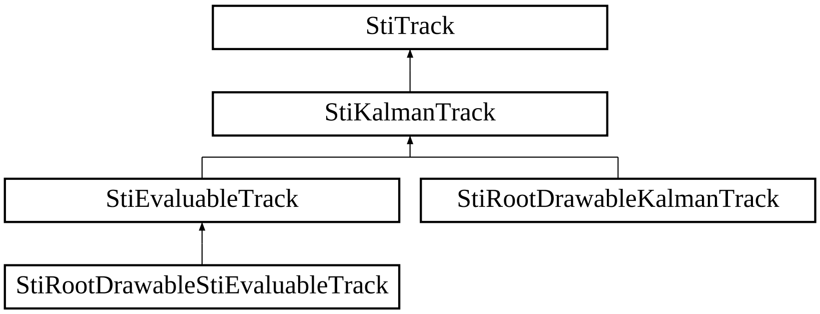
- Sti/**StiHitFiller.h**
- Sti/**StiHitFiller.cxx**

4.36 StiKalmanTrack Class Reference

Definition of Kalman Track.

#include <StiKalmanTrack.h>

Inheritance diagram for StiKalmanTrack::



Public Methods

- **StiKalmanTrack** ()
- virtual ~**StiKalmanTrack** ()
- void **reset** ()
- void **getMomentum** (double p[3], double e[6]) const

  Calculates and returns the momentum and error of the track.

- double **getP** () const

  Calculates and returns the momentum of the track at the inner most node.

- double **getPt** () const

  Calculates and returns the transverse momentum of the track at the inner most node.

- double **getCurvature** () const

  Return the curvature of the track at its inner most point.

- double **getRapidity** () const

  Return the rapidity of the track if the mass is known.

- double **getPseudoRapidity** () const

  Return the pseudorapidity of the track.

- double **getPhi** () const

Return azimuthal angle at inner most point of the track.

- double **getTanL** () const

  Returns the tangent of the dip angle of the track determined at the inner most point of the track.

- double **getDca** (**StiHit** *h=0) const
- double **getDca** (**StiTrack** *t) const
- double **getPrimaryDca** () const
- int **getPointCount** () const

  Return the number of hits associated with this track.

- int **getFitPointCount** () const

  Returns the number of hits associated and used in the fit of this track.

- int **getGapCount** () const

  Return the number of gaps on this track.

- double **getTrackLength** () const
- int **getMaxPointCount** () const
- int **getSeedHitCount** () const

  number of hits used to seed the track.

- void **setSeedHitCount** (int c)
- bool **isPrimary** () const
- double **calculateTrackLength** () const
- double **calculateTrackSegmentLength** (const **StiKalmanTrackNode** &p1, const **StiKalmanTrackNode** &p2) const
- int **calculatePointCount** () const
- int **calculateMaxPointCount** () const
- double **getTpcDedx** () const
- double **getSvtDedx** () const
- StiKTNBidirectionalIterator **begin** () const
- StiKTNBidirectionalIterator **end** () const
- **StiKalmanTrackNode** * **getOuterMostNode** () const

  Accessor method returns the outer most node associated with the track.

- **StiKalmanTrackNode** * **getInnerMostNode** () const

  Accessor method returns the inner most node associated with the track.

- **StiKalmanTrackNode** * **getOuterMostHitNode** () const

  Accessor method returns the outer most hit node associated with the track.

- **StiKalmanTrackNode** ∗ **getInnerMostHitNode** () const

  Accessor method returns the inner most hit node associated with the track.

- **StiKalmanTrackNode** ∗ **getFirstNode** () const

  Accessor method returns the first node associated with the track.

- **StiKalmanTrackNode** ∗ **getLastNode** () const

  Accessor method returns the last node associated with the track.

- void **setLastNode** (**StiKalmanTrackNode** ∗n)
- StiDirection **getTrackingDirection** () const

  Returns the direction (kInsideOut, kOutsideIn) used in the reconstruction of this track.

- StiDirection **getFittingDirection** () const

  Returns the direction (kInsideOut, kOutsideIn) used in the fit of this track.

- void **setTrackingDirection** (StiDirection direction)

  Sets the direction (kInsideOut, kOutsideIn) used in the reconstruction of this track.

- void **setFittingDirection** (StiDirection direction)

  Sets the direction (kInsideOut, kOutsideIn) used in the fit of this track.

- **StiKalmanTrackNode** ∗ **addHit** (**StiHit** ∗h)

  Method used to add a hit to this track.

- **StiKalmanTrackNode** ∗ **insertHit** (**StiHit** ∗hInserted, **StiHit** ∗targetParent)

  Method to insert a hit in this track.

- void **removeHit** (**StiHit** ∗h)

  Remove given hit from this track.

- void **removeAllHits** ()

  Remove all hits and nodes currently associated with this track.

- **StiKalmanTrackNode** ∗ **findHit** (**StiHit** ∗h)

  Work method used to find the node containing the given hit.

- void **initialize** (double curvature, double tanl, const StThreeVectorD &origin, const hitvector &)

  Convenience method to initialize a track based on seed information.

- **StiKalmanTrackNode** ∗ **getNodeNear** (double x) const

  Work method returns the node closest to the given position.

- StThreeVector< double > **getPointNear** (double x) const
- StThreeVector< double > **getGlobalPointNear** (double x) const
- StThreeVector< double > **getGlobalPointAt** (double x) const
- StThreeVector< double > **getMomentumAtOrigin** () const
- StThreeVector< double > **getMomentumNear** (double x)
- StThreeVector< double > **getHitPositionNear** (double x) const
- virtual vector< StMeasuredPoint ∗> **stHits** () const

  return hits;.

- virtual vector< **StiKalmanTrackNode** ∗> **getNodes** (StDetectorId det) const

  return vector of nodes with hits.

- void **swap** ()
- double **getMass** () const
- int **getCharge** () const
- double **getChi2** () const
- double **getDca2** (**StiTrack** ∗t) const
- double **getDca3** (**StiTrack** ∗t) const
- bool **find** (int direction=kOutsideIn)
- void **prune** ()
- void **reserveHits** ()
- bool **extendToVertex** (**StiHit** ∗vertex)
- void **setFlag** (long v)
- long **getFlag** () const

Static Public Methods

- void **setKalmanTrackNodeFactory** (StiObjectFactoryInterface< **StiKalmanTrackNode** > ∗)

  Set the factory used for the creation of kalman track nodes.

- void **setParameters** (StiKalmanTrackFinderParameters ∗p)

Protected Attributes

- StiDirection **trackingDirection**
- StiDirection **fittingDirection**
- **StiKalmanTrackNode** ∗ **firstNode**
- **StiKalmanTrackNode** ∗ **lastNode**
- int **mSeedHitCount**
- long **mFlag**
- double **m**

Static Protected Attributes

- StiKalmanTrackFinderParameters ∗ **pars** = 0
- StiObjectFactoryInterface< **StiKalmanTrackNode** > ∗ **trackNodeFactory** = 0

4.36.1 Detailed Description

Definition of Kalman Track.

A concrete subclass of **StiTrack** (p. 166) defining a Kalman track to be used by the Kalman Track Finder.

The track reconstruction is driven by an instance of class StiKalmanTrackFinder while the Kalman state of the track at any given location is held by instances of the **StiKalmanTrackNode** (p. 129) class. The use of nodes allows, in principle, to have, during the track search, and reconstruction, tracks that behave as trees rather than simple linear or sequential structures.

Users should not invoke the ctor of this class directly but should instead call the "getObject" method of the StiKalmantrackFactory class to get instances of this class. The **StiKalmanTrackFactory** (p. 127) holds (and owns, i.e. has responsibility for memory management) of a large array of re-usable track objects. Instances of this class should only be obtained from the factory as this eliminates (or at the very least minimizes the risk) of memory leaks.

This class holds a static pointer to a track node factory. The factory is invoked whenever instances of StiTrackNode are needed. The class holds pointers to the fisrt and last node associated with a track. Given that the reconstruction proceeds primarily outside-in, the first node is the outer most point associated with this track. The last node is the inner most point associated with the track.

This class includes a host of convenience methods to calculate track properties such as the number of hits on the track (getPointCount), the track length (getTrackLength), etc. Many of those properties are not stored internally but rather calculated on the fly from the appropriate nodes on the tracks. This offers the advantage that it is not necessary to recalculate these various properties systematically each time a fit or re-fit is performed but once when the information is actually needed.

See also:
StiKalmanTrackFactory (p. 127) , **StiKalmanTrackNode** (p. 129) , StiKalmanTrackFinder

Author:
Claude A Pruneau (Wayne State University)

Definition at line 88 of file StiKalmanTrack.h.

4.36.2 Constructor & Destructor Documentation

4.36.2.1 StiKalmanTrack::StiKalmanTrack () `[inline]`

Constructor Delegates the initialization of the object to the reset method. Note that users should not call this ctor directly but should instead invoke to the "getObject" method of the StiKalmantrackFactory class to get instances of this class. The **StiKalmanTrackFactory** (p. 127) holds (and owns, i.e. has responsibility for memory management) of a large array of re-usable track objects. Instances of this class should only be obtained from the factory as this eliminates (or at the very least minimizes the risk) of memory leaks.

Definition at line 100 of file StiKalmanTrack.h.

4.36.2.2 virtual StiKalmanTrack::∼StiKalmanTrack () `[inline, virtual]`

Destructor Nothing to be done as instances of this class do not "own" the objects (i.e. nodes) its members point to.

Definition at line 115 of file StiKalmanTrack.h.

4.36.3 Member Function Documentation

4.36.3.1 StiKalmanTrackNode ∗ StiKalmanTrack::addHit (StiHit ∗ *h*)

Method used to add a hit to this track.

Method used to add an arbitrary hit to a track. A track node is first obtained from the KalmantrackNode factory. The hit is added to the node and the node is added to the track as a child to the last node of the track.

Definition at line 72 of file StiKalmanTrack.cxx.

References StiKalmanTrackNode::add(), StiKalmanTrackNode::fX, StiKalmanTrackNode::reset(), and StiHit::x().

4.36.3.2 StiKTNBidirectionalIterator StiKalmanTrack::begin () const [inline]

Convenience method used to return a track node iterator initialized to the track first node.

Returns:
Bidirectional Itertator of KalmanTrackNodes

Exceptions:
runtime_error

Definition at line 533 of file StiKalmanTrack.h.

Referenced by getMaxPointCount(), and getPointCount().

4.36.3.3 StiKTNBidirectionalIterator StiKalmanTrack::end () const [inline]

Convenience method used to return a track node iterator initialized to the track last node.

Returns:
Bidirectional Itertator of KalmanTrackNodes

Exceptions:
runtime_error

Definition at line 544 of file StiKalmanTrack.h.

Referenced by getMaxPointCount(), getNodes(), getPointCount(), and stHits().

4.36.3.4 bool StiKalmanTrack::extendToVertex (StiHit ∗ *vertex*)

Extend track to the given vertex.

Attempt an extension of the track the given vertex.

1. Get node from node factory.

2. Reset node.
3. Propagate the node from given parent node "sNode", to the given vertex using a call to "propagate".
4. Evaluate the chi2 of the extrapolated if the vertex is added to the track. Done using a call to "evaluateChi2".
5. If chi2 is less than max allowed "maxChi2ForSelection", update track parameters using the vertex as a measurement and add the vertex to the track as the last node.

Notes

- Throws logic_error if no node can be obtained from the node factory.
- The methods "propagate", "evaluateChi2", and "updateNode" may throw runtime_error exceptions which are NOT caught here...

Definition at line 963 of file StiKalmanTrack.cxx.

References StiKalmanTrackNode::add(), StiKalmanTrackNode::evaluateChi2(), StiKalmanTrackNode::propagate(), StiKalmanTrackNode::reset(), and StiKalmanTrackNode::updateNode().

4.36.3.5 StiKalmanTrackNode ∗ StiKalmanTrack::findHit (StiHit ∗ *h*)

Work method used to find the node containing the given hit.

Current implementation only considers the first child of each node and must therefore be revised.

Definition at line 188 of file StiKalmanTrack.cxx.

References StiDefaultMutableTreeNode::getChildCount(), and StiDefaultMutableTreeNode::getFirstChild().

Referenced by insertHit(), and removeHit().

4.36.3.6 int StiKalmanTrack::getCharge () const `[virtual]`

Return the track sign

Notes

1. Use the last node and the field.

Reimplemented from **StiTrack** (p. 167).

Definition at line 566 of file StiKalmanTrack.cxx.

References StiKalmanTrackNode::fP3.

4.36.3.7 double StiKalmanTrack::getChi2 () const `[virtual]`

Return the track chi2

Notes

1. Use the chi2 held by the last hit node used in the fit.

Reimplemented from **StiTrack** (p. 167).

Definition at line 577 of file StiKalmanTrack.cxx.

References StiKalmanTrackNode::fChi2.

4.36.3.8 double StiKalmanTrack::getCurvature () const `[inline, virtual]`

Return the curvature of the track at its inner most point.

Calculates and returns the track curvature at the inner most node held by this track.

Obtains the curvature from the inner most hit node associated with this track.

Reimplemented from **StiTrack** (p. 166).

Definition at line 401 of file StiKalmanTrack.h.

References StiKalmanTrackNode::fP3, and getInnerMostHitNode().

4.36.3.9 double StiKalmanTrack::getDca (StiTrack * *t*) const `[inline]`

Returns the distance of closest approach of this track to the give track.

Returns:
dca in cm.

Definition at line 499 of file StiKalmanTrack.h.

4.36.3.10 double StiKalmanTrack::getDca (StiHit * $h = 0$) const [inline, virtual]

Returns the distance of closest approach of this track to the given hit.

See also:
StiHit (p. 86)

Returns:
dca in cm.

Reimplemented from **StiTrack** (p. 166).

Definition at line 487 of file StiKalmanTrack.h.

4.36.3.11 double StiKalmanTrack::getDca2 (StiTrack * t) const [inline, virtual]

Calculate and return the distance of closest approach to given track - 2D calc

Notes

1. No implementation.
2. Returns 0

Reimplemented from **StiTrack** (p. 166).

Definition at line 511 of file StiKalmanTrack.h.

4.36.3.12 double StiKalmanTrack::getDca3 (StiTrack * t) const [inline, virtual]

Calculate and return the distance of closest approach to given track - 3D calc

Notes

1. No implementation.
2. Returns 0

Reimplemented from **StiTrack** (p. 166).

Definition at line 523 of file StiKalmanTrack.h.

4.36.3.13 int StiKalmanTrack::getFitPointCount () const [virtual]

Returns the number of hits associated and used in the fit of this track.

Return the number of hits (points) used in the fit of this track.

Notes

1. Currently no difference is made between points on the track and fit points on the track.
2. Call "**getPointCount**() (p. 119)" to get the count.

Returns:
number of hits on this track.

Reimplemented from **StiTrack** (p. 166).

Definition at line 701 of file StiKalmanTrack.cxx.

4.36.3.14 int StiKalmanTrack::getGapCount () const [virtual]

Return the number of gaps on this track.

Return the number of gaps (active layers with no hits) along this track.

Notes

1. A gap consists of one or multiple contiguous active layers through which this track passes.
2. There can be gaps on the inside or the outside of the track if no hits are found there.

Returns:
number of gaps.

Reimplemented from **StiTrack** (p. 166).

Definition at line 661 of file StiKalmanTrack.cxx.

4.36.3.15 StiKalmanTrackNode * StiKalmanTrack::getInnerMostHitNode () const

Accessor method returns the inner most hit node associated with the track.

Return the inner most hit associated with this track.

Notes

1. Throws logic_error exception if firstNode or lastNode are not defined, or if track has no hit.
2. Loop through all nodes from **begin**() (p. 111) to **end**() (p. 111) (or vice versa if tracking direction is outside-in) and search for node with hit. Return first hit found.

Returns:
outer most hit node on this track

Definition at line 773 of file StiKalmanTrack.cxx.

Referenced by getCurvature(), getMomentum(), getP(), getPseudoRapidity(), getPt(), and getTrackLength().

4.36.3.16 StiKalmanTrackNode ∗ StiKalmanTrack::getInnerMostNode () const `[inline]`

Accessor method returns the inner most node associated with the track.

Accessor method returns the inner most node associated with the track.

Notes

1. Node returned depends on the direction of tracking.
2. Return firstNode if tracking was done inside-out, lastNode otherwise.
3. No check done to determine whether returned value is non null.

Returns:
outer most node on this track

Definition at line 585 of file StiKalmanTrack.h.

Referenced by StiEvaluableTrackSeedFinder::makeTrack().

4.36.3.17 double StiKalmanTrack::getMass () const `[inline, virtual]`

Return the mass hypothesis used in the resconstruction of this track.

Reimplemented from **StiTrack** (p. 167).

Definition at line 329 of file StiKalmanTrack.h.

4.36.3.18 int StiKalmanTrack::getMaxPointCount () const [virtual]

Returns the maximum number of points that can possibly be on the track given its track parameters, i.e. its position in the detector. The calculation accounts for sublayers that are not active, and nominally active volumes that were turned off or had no data for some reason.

Reimplemented from **StiTrack** (p. 166).

Definition at line 630 of file StiKalmanTrack.cxx.

References begin(), and end().

4.36.3.19 void StiKalmanTrack::getMomentum (double *p*[3], double *e*[6]) const [inline, virtual]

Calculates and returns the momentum and error of the track.

Calculates and returns the momentum and error of the track

This method calculates and returns in the two arrays provided as arguments the 3-momentum and error of the track in Star global coordinates. The 3-momentum is calculated at the inner most point associated with the track. The inner-most point may or may not be the main vertex of the event. Care should thus be exercised while using this method.

The error is calculated (and returned) only if a non null array is passed as a second argument. It is thus possible to get the momentum without a lengthy calculation of the error matrix. The error error matrix corresponds to a full covariance matrix. The definition of the error matrix is described in the introduction of this class definition. Note that the actual calculation of the momentum and associated error is delegated to the track node class and uses the inner most node of the track.

Reimplemented from **StiTrack** (p. 166).

Definition at line 370 of file StiKalmanTrack.h.

References getInnerMostHitNode(), and StiKalmanTrackNode::getMomentum().

4.36.3.20 StiKalmanTrackNode * StiKalmanTrack::getNodeNear (double *x*) const

Work method returns the node closest to the given position.

Work method returns the node closest to the given position. The given position is a radial distance calculated in the local reference frame of the detector.

Definition at line 425 of file StiKalmanTrack.cxx.

References StiDefaultMutableTreeNode::breadthFirstEnumeration(), and StiKalmanTrackNode::fX.

Referenced by getPointNear().

4.36.3.21 StiKalmanTrackNode ∗ StiKalmanTrack::getOuterMostHitNode () const

Accessor method returns the outer most hit node associated with the track.

Return the inner most hit associated with this track.

Notes

1. Throws logic_error exception if firstNode or lastNode are not defined, or if track has no hit.
2. Loop through all nodes from **end**() (p. 111) to **begin**() (p. 111) (or vice versa if tracking direction is outside-in) and search for node with hit. Return first hit found.

Returns:
inner most hit node on this track

Exceptions:
logic_error

Definition at line 738 of file StiKalmanTrack.cxx.

Referenced by getTrackLength().

4.36.3.22 StiKalmanTrackNode ∗ StiKalmanTrack::getOuterMostNode () const `[inline]`

Accessor method returns the outer most node associated with the track.

Accessor method returns the outer most node associated with the track.

Notes

1. Node returned depends on the direction of tracking.
2. Return firstNode if tracking was done outside-in, lastNode otherwise.
3. No check done to determine whether returned value is non null.

Returns:
outer most node on this track

Definition at line 571 of file StiKalmanTrack.h.

4.36.3.23 double StiKalmanTrack::getP () const `[inline, virtual]`

Calculates and returns the momentum of the track at the inner most node.

Calculates and returns the momentum of the track at the inner most node held by this track which may or (or not) be the primary vertex.

Reimplemented from **StiTrack** (p. 166).

Definition at line 382 of file StiKalmanTrack.h.

References getInnerMostHitNode(), and StiKalmanTrackNode::getP().

4.36.3.24 double StiKalmanTrack::getPhi () const `[inline, virtual]`

Return azimuthal angle at inner most point of the track.

Returns the azimuthal angle of the track determined at the inner most point of the track hich may or may not be a vertex.

Returns:
phi in radian

Reimplemented from **StiTrack** (p. 166).

Definition at line 454 of file StiKalmanTrack.h.

4.36.3.25 int StiKalmanTrack::getPointCount () const `[virtual]`

Return the number of hits associated with this track.

Calculate and return the number of hits on this track.

Notes

1. Iterate through all nodes of this track.
2. Count number of hits.

Returns:
number of hits.

Reimplemented from **StiTrack** (p. 166).

Definition at line 605 of file StiKalmanTrack.cxx.

References begin(), and end().

4.36.3.26 StThreeVector< double > StiKalmanTrack::getPointNear (double *x*) const

Convenience method returns a point corresponding to the node of this track which is the closest to the given position.

Definition at line 468 of file StiKalmanTrack.cxx.

References StiKalmanTrackNode::fP0, StiKalmanTrackNode::fP1, StiKalmanTrackNode::fX, and getNodeNear().

4.36.3.27 double StiKalmanTrack::getPrimaryDca () const `[inline]`

Returns the distance of closest approach of this track to the primary vertex

Returns:
dca

Definition at line 557 of file StiKalmanTrack.h.

4.36.3.28 double StiKalmanTrack::getPseudoRapidity () const `[inline, virtual]`

Return the pseudorapidity of the track.

Returns the pseudo-rapidity of the track.

1. Obtains the helix pitch angle from the inner most hit node associated with the track.
2. Calculate/return the pseudo-rapidity using the pitch angle.

Returns:
pseudo-rapidity

Reimplemented from **StiTrack** (p. 166).

Definition at line 442 of file StiKalmanTrack.h.

References getInnerMostHitNode(), and getTanL().

4.36.3.29 double StiKalmanTrack::getPt () const `[inline, virtual]`

Calculates and returns the transverse momentum of the track at the inner most node.

Calculates and returns the transverse momentum of the track at the inner most node held by this track which may or (or not) be the primary vertex.

Reimplemented from **StiTrack** (p. 166).

Definition at line 391 of file StiKalmanTrack.h.

References getInnerMostHitNode(), and StiKalmanTrackNode::getPt().

4.36.3.30 double StiKalmanTrack::getRapidity () const `[inline, virtual]`

Return the rapidity of the track if the mass is known.

Returns the rapidity of the track if the mass is known.

1. Obtains the momentum from the inner most hit node associated with the track.
2. Obtains the mass of this track using the **getMass**() (p. 116) method. If the mass returned is negative, throws a runtime_error exception.

Exceptions:

runtime_error

Returns:

rapidity

Reimplemented from **StiTrack** (p. 166).

Definition at line 417 of file StiKalmanTrack.h.

References StiKalmanTrackNode::getMomentum().

4.36.3.31 double StiKalmanTrack::getTanL () const `[inline, virtual]`

Returns the tangent of the dip angle of the track determined at the inner most point of the track.

Return tan(lambda) of the particle at the inner most node held by this track which may (or not) be the primary vertex.

Returns:
tan(lambda)

Reimplemented from **StiTrack** (p. 166).

Definition at line 466 of file StiKalmanTrack.h.

Referenced by getPseudoRapidity().

4.36.3.32 double StiKalmanTrack::getTrackLength () const [virtual]

Returns the track length (in centimeters) from the first to the last point on track. The main vertex is included in the calculation if associated with the track.

Reimplemented from **StiTrack** (p. 166).

Definition at line 715 of file StiKalmanTrack.cxx.

References StiKalmanTrackNode::getCurvature(), getInnerMostHitNode(), and getOuterMostHitNode().

4.36.3.33 void StiKalmanTrack::initialize (double *curvature*, double *tanl*, const StThreeVectorD & *origin*, const hitvector & *v*)

Convenience method to initialize a track based on seed information.

Initialization of this kalman track from external parameters.

This track object is initialized on the basis of parameters determined externally. The parameters consist of the track curvature, the tangent of pitch angle, the origin of the helix, and a vector of hits already associated with the track.

Arguments:

curvature	1/radius of the tack.
tanl	tan(pitch angle)
origin	origin of the track in global coordinates.
v	vector of hits associated with this track.

Internal Track Representation:

x	fX	independent variable
state[0]	fP0	y; ordinate at "x"
state[1]	fP1	z; position along beam axis at "x"
state[2]	fP2	eta=C*x0; C == curvature, x0==position of helix center.
state[3]	fP3	C (local) curvature of the track
state[4]	fP4	tan(l)
error[0]	fC00	Error Matrix - Symmetric e.g. fC20=fC02
error[1]	fC10	
error[2]	fC11	
error[3]	fC20	
error[4]	fC21	
error[5]	fC22	
error[6]	fC30	
error[7]	fC31	
error[8]	fC32	
error[9]	fC33	
error[10]	fC40	
error[11]	fC41	
error[12]	fC42	
error[13]	fC43	
error[14]	fC44	

Algorithm:

1. Verify that a valid node factory exists.
2. Use local arrays state and error to add and set all nodes of this track.
3. Use the same curvature, and tanl for all nodes as supplied in argument list.
4. Use Unit matrix for error matrix.
5. Loop over all hits of the input hit vector and create a track node for each.
6. Paramters of the track node are set according to the y,z of the hits added.
7. Hits given are transformed in the local coordinates of their detector.

Notes:

1. Throws a logic_error exception if no track node factory is available.
2. Throws a logic_error exception if the factory is not a castable to a factory of **StiKalmanTrackNode** (p. 129).
3. Throws a logic error exception if hits do not have a valid pointer to a detector object.

Definition at line 364 of file StiKalmanTrack.cxx.

References StiKalmanTrackNode::add(), StiKalmanTrackNode::fAlpha, StiKalmanTrackNode::reset(), and StiKalmanTrackNode::set().

4.36.3.34 bool StiKalmanTrack::isPrimary () const

Identifies the track as a primary or secondary track. The track is defined as primary if it contains a primary vertex i.e. if the vertex was included as a point to the track because it had low enough a incremental chi2.

Definition at line 806 of file StiKalmanTrack.cxx.

4.36.3.35 void StiKalmanTrack::prune ()

Prune the track to select the best branch of the tree identified by given leaf node.

The best brach is assumed to be the one given by the leaf "node". All siblings of the given node, are removed, and iteratively all siblings of its parent are removed from the parent of the parent, etc.

Definition at line 917 of file StiKalmanTrack.cxx.

References StiDefaultMutableTreeNode::getParent(), and StiDefaultMutableTreeNode::removeAllChildrenBut().

4.36.3.36 void StiKalmanTrack::removeAllHits ()

Remove all hits and nodes currently associated with this track.

Remove all references to hits from this track be setting the firstNode and lastNode pointers to "0".

Note

1. No need to destroy any object since the memory for the nodes and hits is owned by the factory that supply these.

Definition at line 216 of file StiKalmanTrack.cxx.

4.36.3.37 void StiKalmanTrack::reserveHits ()

Declare hits associated with given track as used.

Declare hits on the track ending at "node" as used. This method starts with the last node and seeks the parent of each node recursively. The hit associated with each node (when there is a hit) is set to "used".

Definition at line 936 of file StiKalmanTrack.cxx.

4.36.3.38 void StiKalmanTrack::reset () `[virtual]`

Reset the class members to their default state. This method is called by the ctor of the class to initialize the members of the class to an "empty" or null track state. The method must also be called everytime an instance of this class is retrieved from its factory in order to set the first and last nodes to "null" thus guaranteeing that the track object is empty i.e. does not represent any track and is thus ready for a new search and reconstruction.

Reimplemented from **StiTrack** (p. 166).

Reimplemented in **StiEvaluableTrack** (p. 74), **StiRootDrawableKalmanTrack** (p. 151), and **StiRootDrawableStiEvaluableTrack** (p. 156).

Definition at line 35 of file StiKalmanTrack.cxx.

Referenced by StiRootDrawableKalmanTrack::reset(), StiEvaluableTrack::reset(), and StiLocalTrackMerger::StiLocalTrackMerger().

4.36.3.39 void StiKalmanTrack::setKalmanTrackNodeFactory (StiObjectFactoryInterface< StiKalmanTrackNode > * *val*) `[static]`

Set the factory used for the creation of kalman track nodes.

Set the factory used for the creation of kalman track nodes.

See also:
StiKalmanTrackNodeFactory (p. 139)

Definition at line 52 of file StiKalmanTrack.cxx.

4.36.3.40 void StiKalmanTrack::swap ()

Swap the track node sequence inside-out

Algorithm

1. Loop through the node sequence starting with the firstNode and invert the parent child relationships.
2. Include removal of all children for each node.
3. Include change of parent
4. Set parent of last node as "0" to complete swap.
5. Change the "trackingDirection" flag to reflect the swap.

Definition at line 823 of file StiKalmanTrack.cxx.

References StiDefaultMutableTreeNode::getChildCount(), StiDefaultMutableTreeNode::getFirstChild(), StiDefaultMutableTreeNode::removeAllChildren(), and StiDefaultMutableTreeNode::setParent().

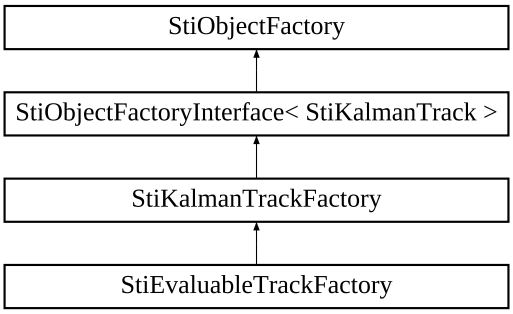
The documentation for this class was generated from the following files:

- Sti/**StiKalmanTrack.h**
- Sti/**StiKalmanTrack.cxx**

4.37 StiKalmanTrackFactory Class Reference

`#include <StiFactoryTypes.h>`

Inheritance diagram for StiKalmanTrackFactory::



Public Methods

- **StiKalmanTrackFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

  This is the only constructor available.

- virtual ~**StiKalmanTrackFactory** ()

  Default destructor.

Protected Methods

- virtual void * **makeNewObject** () const

  Return a pointer to a new **StiKalmanTrack** (p. 105) *object on the heap.*

4.37.1 Detailed Description

StiKalmanTrackFactory is derived from the abstract factory interface class StiObjectFactoryInterface<**StiKalmanTrack** (p. 105)>. This class has the sole purpose of creating an **StiKalmanTrack** (p. 105) object on the heap for each call to **makeNewObject**() (p. 127).

Author:
M.L. Miller (Yale Software)

Definition at line 50 of file StiFactoryTypes.h.

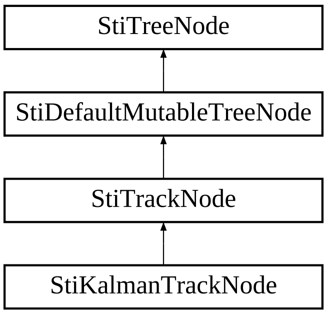
The documentation for this class was generated from the following files:

- Sti/**StiFactoryTypes.h**
- Sti/**StiFactoryTypes.cxx**

4.38 StiKalmanTrackNode Class Reference

#include <StiKalmanTrackNode.h>

Inheritance diagram for StiKalmanTrackNode::



Public Methods

- void **reset** ()

  Resets the node to a "null" un-used state.

- void **set** (**StiHit** ∗hit, const double alpha, const double xRef, const double xx[5], const double cc[15], const double dEdx, const double chi2)

  Sets the various attributes of this node based on the argument list.

- void **setState** (const StiKalmanTrackNode ∗node)

  Sets the Kalman state of this node equal to that of the given node.

- void **get** (double &alpha, double &xRef, double x[5], double cc[15], double &dEdx, double &chi2)

  Extract state information from this node.

- double **getCharge** () const

  Get the charge (sign) of the track at this node.

- StThreeVectorF **getMomentumF** () const

  Convenience Method that returns the track momentum at this node.

- StThreeVectorF **getGlobalMomentumF** () const

  Convenience Method that returns the track momentum at this node in global coordinates.

- StThreeVector< double > **getMomentum** () const
- StThreeVector< double > **getGlobalMomentum** () const
- void **getMomentum** (double p[3], double e[6]=0) const

  Calculates and returns the momentum and error of the track at this node. The momentum is in the local reference frame of this node.

- double **getCurvature** () const

  Calculates and returns the tangent of the track pitch angle at this node.

- double **getDipAngle** () const
- double **getTanL** () const
- double **getP** () const

  Calculates and returns the momentum of the track at this node.

- double **getPt** () const

  Calculates and returns the transverse momentum of the track at this node.

- void **getGlobalMomentum** (double p[3], double e[6]=0) const

  Calculates and returns the momentum and error of the track at this node in global coordinates.

- void **setAsCopyOf** (const StiKalmanTrackNode *node)

  Set the attributes of this node as a copy of the given node.

- int **propagate** (StiKalmanTrackNode *p, const StiDetector *tDet)

  Propagates a track encapsulated by the given node "p" to the given detector "tDet".

- void **propagate** (const StiKalmanTrackNode *p, const **StiHit** *vertex)

  Propagates a track encapsulated by the given node "p" to the given vertex.

- double **evaluateDedx** ()

  Evaluates, stores and returns the dedx associated with this node. Possible returned values are: > 0 : value of dedx -1 : pathlength was invalid or less than "0" -2 : no hit is associated with the node. -3 : invalid eloss data for this node.

- void **propagate** (double x)
- void **propagateCylinder** (double x)
- void **propagateError** ()
- void **propagateMCS** (double density, double radThickness, double massHypo)
- StThreeVector< double > **getPointAt** (double xk) const

Extrapolate the track parameters to radial position "x" and return a point global coordinates along the track at that point.

- double **evaluateChi2** ()
- void **updateNode** ()
- void **extendToVertex** ()
- void **rotate** (double alpha)
- void **add** (StiKalmanTrackNode ∗newChild)
- double **getWindowY** () const
- double **getWindowZ** () const
- double **pitchAngle** () const
- double **crossAngle** () const
- StThreeVector< double > **getHelixCenter** () const

  Return center of helix circle in global coordinates.

- void **setError** (pair< double, double > p)

Static Public Methods

- void **setParameters** (StiKalmanTrackFinderParameters ∗parameters)
- double **getFieldConstant** ()

Public Attributes

- double **fAlpha**

  rotation angle of local coordinates wrt global coordinates.

- double **fX**

  local X-coordinate of this track (reference plane).

- double **fP0**

  local Y-coordinate of this track (reference plane).

- double **fP1**

  local Z-coordinate of this track (reference plane).

- double **fP2**

  (signed curvature)∗(local X-coordinate of helix axis).

- double **fP3**

  signed curvature [sign = sign(-qB)].

- double **fP4**

  tangent of the track momentum dip angle.

- double **fC00**

  covariance matrix of the track parameters.

- double **fC10**
- double **fC11**
- double **fC20**
- double **fC21**
- double **fC22**
- double **fC30**
- double **fC31**
- double **fC32**
- double **fC33**
- double **fC40**
- double **fC41**
- double **fC42**
- double **fC43**
- double **fC44**
- double **fChi2**
- float **fdEdx**
- float **pathLength**
- float **eyy**
- float **ezz**
- int **hitCount**
- int **nullCount**
- int **contiguousHitCount**
- int **contiguousNullCount**

Static Protected Attributes

- bool **recurse** = false
- StiKalmanTrackFinderParameters * **pars** = 0
- int **shapeCode** = 0
- const StiDetector * **det** = 0
- const StiPlanarShape * **planarShape** = 0
- const StiCylindricalShape * **cylinderShape** = 0
- StiMaterial * **gas** = 0
- StiMaterial * **prevGas** = 0
- StiMaterial * **mat** = 0
- StiMaterial * **prevMat** = 0

- double **x1** = 0
- double **x2** = 0
- double **y1** = 0
- double **z1** = 0
- double **dx** = 0
- double **r1** = 0
- double **r2** = 0
- double **c1** = 0
- double **c2** = 0
- double **c1sq** = 0
- double **c2sq** = 0
- double **x0** = 0
- double **y0** = 0
- double **radThickness** = 0
- double **density** = 0
- double **gasDensity** = 0
- double **matDensity** = 0
- double **gasRL** = 0
- double **matRL** = 0
- bool **useCalculatedHitError** = true

Friends

- ostream & **operator**<< (ostream &os, const StiKalmanTrackNode &n)

4.38.1 Detailed Description

Work class used to handle Kalman filter information while constructing track nodes. A node may or may not own a hit depending whether it lies on a measurement layer where a hit was found. A node can have 0, 1, or many children. The canonical ordering of the nodes is outside-in. Children node should be at smaller radii.

Author:
Claude A Pruneau

Definition at line 30 of file StiKalmanTrackNode.h.

4.38.2 Member Function Documentation

4.38.2.1 double StiKalmanTrackNode::evaluateChi2 ()

Calculate the increment of chi2 caused by the addition of this node to the track.

Uses the track extrapolation to "fX", and hit position to evaluate an update to the track chi2. The new chi2 is stored internally in this node. The increment is returned as a convenience for the caller method.

Notes

1. Use full error matrices.

2. Return increment in chi2 implied by the node/hit assocition.

3. Throws an exception if numerical problems arise.

Definition at line 787 of file StiKalmanTrackNode.cxx.

References fC00, fP0, and fP1.

Referenced by StiKalmanTrack::extendToVertex().

4.38.2.2 double StiKalmanTrackNode::evaluateDedx ()

Evaluates, stores and returns the dedx associated with this node. Possible returned values are: > 0 : value of dedx -1 : pathlength was invalid or less than "0" -2 : no hit is associated with the node. -3 : invalid eloss data for this node.

Calculates the differential energy loss per unit length in the track segment associated with this node.

Definition at line 552 of file StiKalmanTrackNode.cxx.

4.38.2.3 void StiKalmanTrackNode::getMomentum (double p[3], double e[6] = 0) const

Calculates and returns the momentum and error of the track at this node. The momentum is in the local reference frame of this node.

Calculate/return track 3-momentum and error.

Calculate the 3-momentum of the track in the local reference frame.

Momentum Representation

p[0]	px	outward
p[1]	py	along detector plane
p[2]	pz	along beam direction

Notes:

1. Throws runtime_error exception if |sin(phi)^2|>1.
2. Bypasses error calculation if error array "e" is a null pointer.

Definition at line 229 of file StiKalmanTrackNode.cxx.

References fP2, fP3, fP4, fX, and getPt().

Referenced by getGlobalMomentum().

4.38.2.4 double StiKalmanTrackNode::getP () const `[inline]`

Calculates and returns the momentum of the track at this node.

Calculate/return the track momentum

Calculate the track momentum in GeV/c based on this node's track parameters.

The momentum is calculated based on the track curvature held by this node. A minimum curvature of 1e-12 is allowed.

Definition at line 296 of file StiKalmanTrackNode.h.

References fP3, and fP4.

Referenced by StiKalmanTrack::getP().

4.38.2.5 double StiKalmanTrackNode::getPt () const `[inline]`

Calculates and returns the transverse momentum of the track at this node.

Calculate/return the track transverse momentum

Calculate the track transverse momentum in GeV/c based on this node's track parameters.

The momentum is calculated based on the track curvature held by this node. A minimum curvature of 1e-12 is allowed.

Definition at line 279 of file StiKalmanTrackNode.h.

References fP3.

Referenced by getMomentum(), getMomentumF(), and StiKalmanTrack::getPt().

4.38.2.6 void StiKalmanTrackNode::propagate (double *xk*)

Work method used to perform the tranport of "this" node from its current "fX" position to the given position "xk". Throws a runtime_error exception if the propagation cannot be carried out.

Definition at line 581 of file StiKalmanTrackNode.cxx.

References fP0, fP1, fP2, fP3, fP4, and fX.

4.38.2.7 void StiKalmanTrackNode::propagate (const StiKalmanTrackNode ∗ *pNode*, const StiHit ∗ *vertex*)

Propagates a track encapsulated by the given node "p" to the given vertex.

Propagate the track encapsulated by pNode to the given vertex. Use this node to represent the track parameters at the vertex.

This method propagates the track from the given parent node "pNode" to the given vertex effectively calculating the location (x,y,z) of the track near the given vertex. It use "this" node to represent/hold the track parameters at the vertex.

Definition at line 539 of file StiKalmanTrackNode.cxx.

References fAlpha, propagate(), propagateError(), setState(), StiHit::x(), and StiHit::y().

4.38.2.8 int StiKalmanTrackNode::propagate (StiKalmanTrackNode ∗ *pNode*, const StiDetector ∗ *tDet*)

Propagates a track encapsulated by the given node "p" to the given detector "tDet".

Steering routine that propagates the track encapsulated by the given node "pNode" to the given detector "tDet". The propagation involves the following steps.

1. Extrapolation of the existing track to the next layer, by "transporting" the track a smaller radius.
2. Determine if the extrapolation actually intersects an existing volume.
3. Exit with status code if no intersection is found.
4. Transport the error matrix to the new radius.

5. If mcsCalculated==true, proceed to calculate MCS effects on the error matrix.
6. if elossCalculated==true, proceed to calculate Eloss effects on the track parameters.

Currently, propagate can handle kPlaner and kCylindrical geometries only. An exception is thrown if other geometry shape are used.

Definition at line 397 of file StiKalmanTrackNode.cxx.

References fAlpha, fP0, fP1, fP3, fP4, propagateCylinder(), propagateError(), rotate(), and setState().

Referenced by StiKalmanTrack::extendToVertex(), and propagate().

4.38.2.9 void StiKalmanTrackNode::propagateCylinder (double *L*)

Work method used to perform the transport of "this" node from its current "fX" position to the a cylindtical surface of the given radius.

Definition at line 610 of file StiKalmanTrackNode.cxx.

References fP0, fP1, fP2, fP3, fP4, and fX.

Referenced by propagate().

4.38.2.10 void StiKalmanTrackNode::propagateError ()

Propagate the track error matrix

Note: This method must be called ONLY after a call to the propagate method.

Definition at line 670 of file StiKalmanTrackNode.cxx.

References fC00, fP0, fP1, fP2, fP3, fP4, and fX.

Referenced by propagate().

4.38.2.11 void StiKalmanTrackNode::rotate (double *alpha*)

Rotate this node track representation azymuthally by given angle.

This method rotates by an angle alpha the track representation held by this node.

Notes

1. The rotation is bound between -M_PI and M_PI.

2. Throws runtime_error if "(fP0-y0)*fP3>=0" in order to avoid math exception.
3. Avoid undue rotations as they are CPU intensive...

Definition at line 898 of file StiKalmanTrackNode.cxx.

References fC00.

Referenced by propagate().

4.38.2.12 void StiKalmanTrackNode::updateNode ()

Update the track parameters using this node.

This method uses the hit contained by node to update the track parameters contained by this node and thus complete the propagation of this track to the location x=fX.

1. Throw a runtime_error exception if no hit is actually associated with this node.
2. Compute the measurement error matrix "r". Invert it.
3. Update the measurement matrix "k" and calculate updated curvature, eta, and pitch.
4. Update track error matrix.

Notes

1. Throw logic_error if no hit is associated with this node.
2. Throw runtime_error if determinent of "r" matrix is null.

Definition at line 835 of file StiKalmanTrackNode.cxx.

Referenced by StiKalmanTrack::extendToVertex().

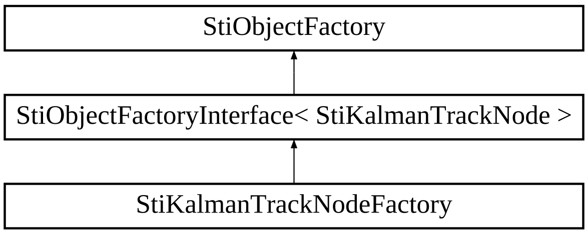
The documentation for this class was generated from the following files:

- Sti/**StiKalmanTrackNode.h**
- Sti/**StiKalmanTrackNode.cxx**

4.39 StiKalmanTrackNodeFactory Class Reference

`#include <StiFactoryTypes.h>`

Inheritance diagram for StiKalmanTrackNodeFactory::



Public Methods

- **StiKalmanTrackNodeFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

  This is the only constructor available.

- virtual ~**StiKalmanTrackNodeFactory** ()

  Default destructor.

Protected Methods

- virtual void ∗ **makeNewObject** () const

  Return a pointer to a new **StiKalmanTrackNode** (p. 129) *object on the heap.*

4.39.1 Detailed Description

StiKalmanTrackNodeFactory is derived from the abstract factory interface class StiObjectFactoryInterface<**StiKalmanTrackNode** (p. 129)>. This class has the sole purpose of creating an **StiKalmanTrackNode** (p. 129) object on the heap for each call to **makeNewObject**() (p. 139).

Author:

M.L. Miller (Yale Software)

Definition at line 138 of file StiFactoryTypes.h.

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

- Sti/**StiFactoryTypes.h**
- Sti/**StiFactoryTypes.cxx**

4.40 StiKTNIterator Class Reference

`#include <StiKTNIterator.h>`

4.40.1 Detailed Description

This class is an STL compliant forward iterator that will traverse from the leaf of a tree upward to a root.

Author:
M.L. Miller (Yale Software)

Note:
We use the defualt copy/assignment generated by compiler.
Singularity (i.e., 'end') is represented by setting mNode=0.
StiKTNIterator is a non-virtual class.

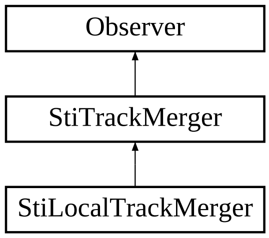
The documentation for this class was generated from the following file:

- Sti/**StiKTNIterator.h**

4.41 StiLocalTrackMerger Class Reference

#include <StiLocalTrackMerger.h>

Inheritance diagram for StiLocalTrackMerger::



Public Methods

- **StiLocalTrackMerger** (**StiTrackContainer** ∗)

  One must provide a valid pointer to the track container.

- virtual ∼**StiLocalTrackMerger** ()
- void **setDeltaR** (double)

  Set the search window in radius.

- virtual void **mergeTracks** ()

  Merge the tracks in the track container.

Protected Methods

- virtual void **getNewState** ()

4.41.1 Detailed Description

StiLocalTrackMerger is the most naive implementation to merge split tracks. It implements the algorithm previously used in the TPT module, identifying split tracks via successive one dimensional tests in the 5 helix dimensions. This comparison is based on the charge of the particle, the radius of curvature, the dip angle, and the center of the circle as projected onto the transverse (x-y) plane. Additionally, a test is perfomed to calculate the DCA of the two tracks to a common point.

Author:

M.L. Miller (Yale Software)

Definition at line 26 of file StiLocalTrackMerger.h.

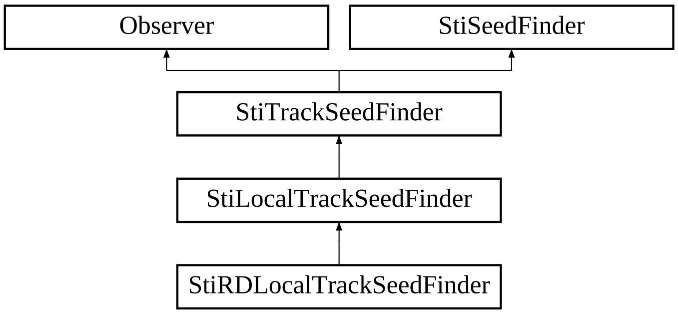
The documentation for this class was generated from the following files:

- Sti/**StiLocalTrackMerger.h**
- Sti/**StiLocalTrackMerger.cxx**

4.42 StiLocalTrackSeedFinder Class Reference

`#include <StiLocalTrackSeedFinder.h>`

Inheritance diagram for StiLocalTrackSeedFinder::



Public Methods

- **StiLocalTrackSeedFinder** (**StiDetectorContainer** ∗, **StiHitContainer** ∗)
- virtual ~**StiLocalTrackSeedFinder** ()
- virtual void **getNewState** ()
- virtual bool **hasMore** ()
- virtual **StiKalmanTrack** ∗ **next** ()
- virtual void **reset** ()
- virtual void **addLayer** (StiDetector ∗)
- virtual void **print** () const

Protected Types

- typedef vector< StiDetector ∗> **DetVec**
- typedef vector< **StiHit** ∗> **HitVec**

Protected Methods

- virtual **StiKalmanTrack** ∗ **makeTrack** (**StiHit** ∗)

Protected Attributes

- DetVec **mDetVec**
- DetVec::iterator **mCurrentDet**

- double **mCurrentRadius**
- HitVec::iterator **mHitsBegin**
- HitVec::iterator **mHitsEnd**
- HitVec::iterator **mCurrentHit**
- double **mDeltaY**
- double **mDeltaZ**
- unsigned int **mSeedLength**
- double **mExtrapDeltaY**
- double **mExtrapDeltaZ**
- unsigned int **mSkipped**
- unsigned int **mMaxSkipped**
- unsigned int **mExtrapMinLength**
- unsigned int **mExtrapMaxLength**
- bool **mUseOrigin**
- HitVec **mSeedHitVec**
- bool **mDoHelixFit**
- StiHelixCalculator **mHelixCalculator**
- **StiHelixFitter mHelixFitter**

4.42.1 Detailed Description

StiLocalTrackSEedFinder is a concrete implementation of StiTrackSeedFinder. It is built from a collection of StiDetectorObjects, which it stores in a vector and orders properly, then uses each detector as a layer from which a one-point seed can be generated. It then proceeds to step inwards, iteratively making a local decision at each step.

Author:
M.L. Miller

Definition at line 33 of file StiLocalTrackSeedFinder.h.

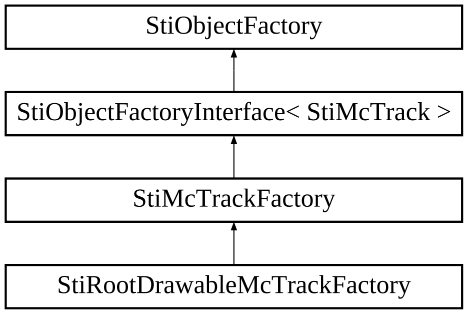
The documentation for this class was generated from the following files:

- Sti/**StiLocalTrackSeedFinder.h**
- Sti/**StiLocalTrackSeedFinder.cxx**

4.43 StiMcTrackFactory Class Reference

`#include <StiMcTrack.h>`

Inheritance diagram for StiMcTrackFactory::



Public Methods

- **StiMcTrackFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

  This is the only constructor available.

- virtual ~**StiMcTrackFactory** ()

  Default destructor.

Protected Methods

- virtual void ∗ **makeNewObject** () const

  Return a pointer to a new StiMcTrack object on the heap.

4.43.1 Detailed Description

StiMcTrack factory

Definition at line 67 of file StiMcTrack.h.

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

- Sti/**StiMcTrack.h**
- Sti/**StiMcTrack.cxx**

4.44 StiOrderKey Struct Reference

`#include <StiCompositeTreeNode.h>`

Public Methods

- **StiOrderKey** ()
- **StiOrderKey** (double k, unsigned int i)

Public Attributes

- double **key**
- unsigned int **index**

4.44.1 Detailed Description

This is used to eager-cache information for sorting and/or traversal of the tree.

Definition at line 82 of file StiCompositeTreeNode.h.

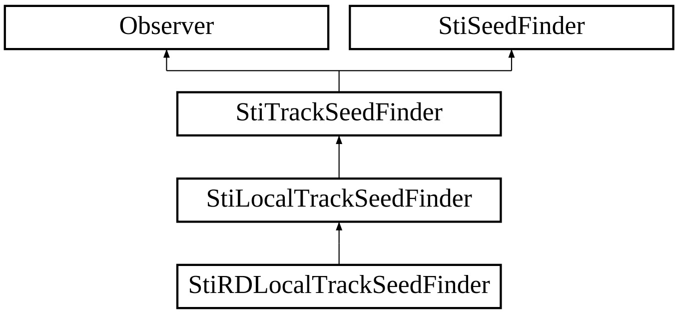
The documentation for this struct was generated from the following file:

- Sti/**StiCompositeTreeNode.h**

4.45 StiRDLocalTrackSeedFinder Class Reference

`#include <StiRDLocalTrackSeedFinder.h>`

Inheritance diagram for StiRDLocalTrackSeedFinder::



Public Methods

- **StiRDLocalTrackSeedFinder** (**StiDetectorContainer** ∗, **StiHitContainer** ∗)
- virtual ∼**StiRDLocalTrackSeedFinder** ()
- virtual void **getNewState** ()
- virtual void **reset** ()

Protected Methods

- virtual **StiKalmanTrack** ∗ **makeTrack** (**StiHit** ∗)

Protected Attributes

- StiRootDrawableHits ∗ **mdrawablehits**

4.45.1 Detailed Description

StiRDLocalTrackSeedFinder is a ROOT rendered visual version of class **StiLocalTrackSeedFinder** (p. 144). Essentially, it displays the set of points that are selected as the seed of a track and used to initialize the kalman track.

Author:
M.L. Miller (Yale Software)

Definition at line 24 of file StiRDLocalTrackSeedFinder.h.

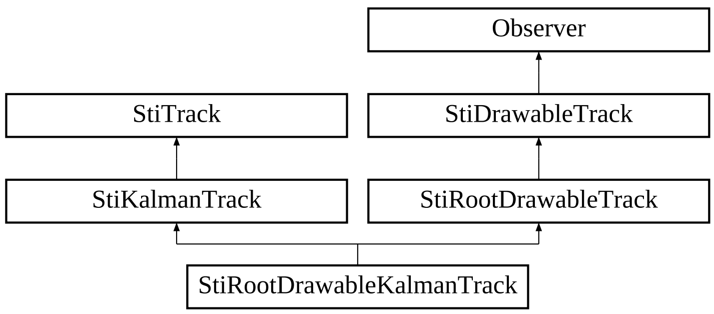
The documentation for this class was generated from the following files:

- StiGui/**StiRDLocalTrackSeedFinder.h**
- StiGui/**StiRDLocalTrackSeedFinder.cxx**

4.46 StiRootDrawableKalmanTrack Class Reference

`#include <StiRootDrawableKalmanTrack.h>`

Inheritance diagram for StiRootDrawableKalmanTrack::



Public Methods

- **StiRootDrawableKalmanTrack** ()
- virtual ~**StiRootDrawableKalmanTrack** ()
- virtual void **fillHitsForDrawing** ()
- virtual void **reset** ()

Protected Methods

- void **getNewState** ()

4.46.1 Detailed Description

Work class used to display Kalman tracks with ROOT

Author:
M.L. Miller (Yale Software)

Definition at line 10 of file StiRootDrawableKalmanTrack.h.

4.46.2 Member Function Documentation

4.46.2.1 void StiRootDrawableKalmanTrack::reset () `[virtual]`

Reset the class members to their default state. This method is called by the ctor of the class to initialize the members of the class to an "empty" or null track state. The method must also be called everytime an instance of this class is retrieved from its factory in order to set the first and last nodes to "null" thus guaranteeing that the track object is empty i.e. does not represent any track and is thus ready for a new search and reconstruction.

Reimplemented from **StiKalmanTrack** (p. 125).

Definition at line 124 of file StiRootDrawableKalmanTrack.cxx.

References StiKalmanTrack::reset().

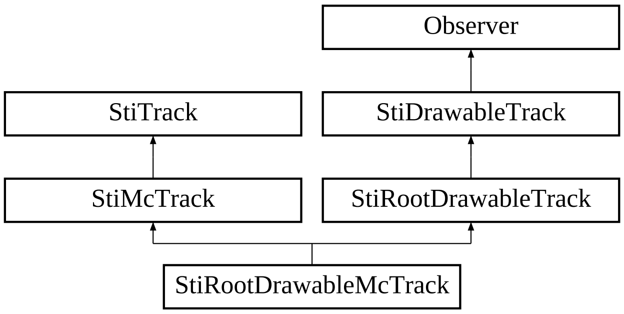
The documentation for this class was generated from the following files:

- StiGui/**StiRootDrawableKalmanTrack.h**
- StiGui/**StiRootDrawableKalmanTrack.cxx**

4.47 StiRootDrawableMcTrack Class Reference

`#include <StiRootDrawableMcTrack.h>`

Inheritance diagram for StiRootDrawableMcTrack::



Public Methods

- **StiRootDrawableMcTrack** ()
- virtual ~**StiRootDrawableMcTrack** ()
- virtual void **fillHitsForDrawing** ()
- virtual void **reset** ()

Protected Methods

- void **getNewState** ()

4.47.1 Detailed Description

Work class used to display Monte Carlo tracks with ROOT. Instances of this class hold a pointer to the actual track.

Author:
Claude A Pruneau

Definition at line 13 of file StiRootDrawableMcTrack.h.

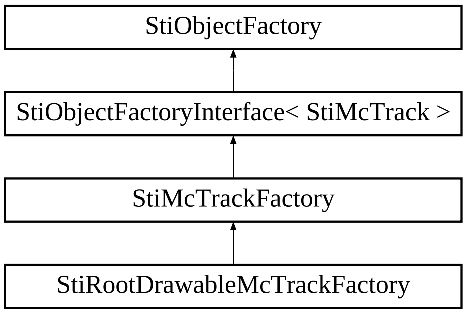
The documentation for this class was generated from the following files:

- StiGui/**StiRootDrawableMcTrack.h**
- StiGui/**StiRootDrawableMcTrack.cxx**

4.48 StiRootDrawableMcTrackFactory Class Reference

`#include` <StiRootDrawableMcTrack.h>

Inheritance diagram for StiRootDrawableMcTrackFactory::



Public Methods

- **StiRootDrawableMcTrackFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

  This is the only constructor available.

- virtual ~**StiRootDrawableMcTrackFactory** ()

  Default destructor.

Protected Methods

- virtual void ∗ **makeNewObject** () const

  Return a pointer to a new StiMcTrack object on the heap.

4.48.1 Detailed Description

StiRootDrawableMcTrack (p. 152) factory

Definition at line 26 of file StiRootDrawableMcTrack.h.

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

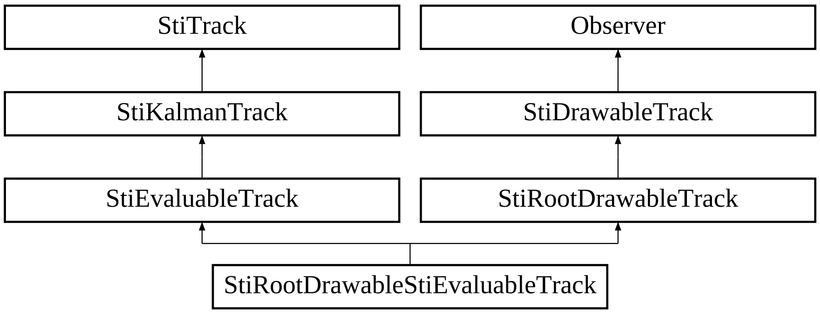
- StiGui/**StiRootDrawableMcTrack.h**

- StiGui/**StiRootDrawableMcTrack.cxx**

4.49 StiRootDrawableStiEvaluableTrack Class Reference

#include <StiRootDrawableStiEvaluableTrack.h>

Inheritance diagram for StiRootDrawableStiEvaluableTrack::



Public Methods

- **StiRootDrawableStiEvaluableTrack** ()
- virtual ~**StiRootDrawableStiEvaluableTrack** ()
- virtual void **fillHitsForDrawing** ()
- virtual void **reset** ()

    Reset the class members to default state.

Protected Methods

- void **getNewState** ()
- void **setLineInfo** ()

4.49.1 Detailed Description

Work class used to display StiEvaluable tracks with ROOT

Author:
M. Miller

Definition at line 9 of file StiRootDrawableStiEvaluableTrack.h.

4.49.2 Member Function Documentation

4.49.2.1 void StiRootDrawableStiEvaluableTrack::reset () [virtual]

Reset the class members to default state.

This is used so that **StiEvaluableTrack** (p. 73) objects can be owned and served by StiObjectFactory. It is guarunteed that a call to **reset**() (p. 156) fully propogates up the inheritance tree.

Reimplemented from **StiEvaluableTrack** (p. 74).

Definition at line 218 of file StiRootDrawableStiEvaluableTrack.cxx.

References StiEvaluableTrack::reset().

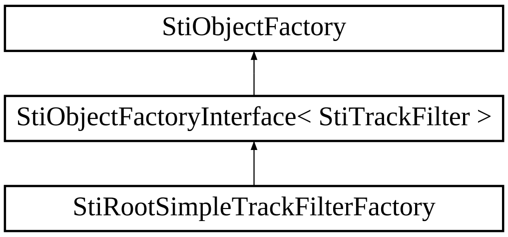
The documentation for this class was generated from the following files:

- StiGui/**StiRootDrawableStiEvaluableTrack.h**
- StiGui/**StiRootDrawableStiEvaluableTrack.cxx**

4.50 StiRootSimpleTrackFilterFactory Class Reference

`#include` <StiRootSimpleTrackFilter.h>

Inheritance diagram for StiRootSimpleTrackFilterFactory::



Public Methods

- **StiRootSimpleTrackFilterFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

  This is the only constructor available.

- virtual ~**StiRootSimpleTrackFilterFactory** ()

  Default destructor.

Protected Methods

- virtual void ∗ **makeNewObject** () const

  Return a pointer to a new StiRootSimpleTrackFilter object on the heap.

4.50.1 Detailed Description

StiRootSimpleTrackFilter factory

Definition at line 28 of file StiRootSimpleTrackFilter.h.

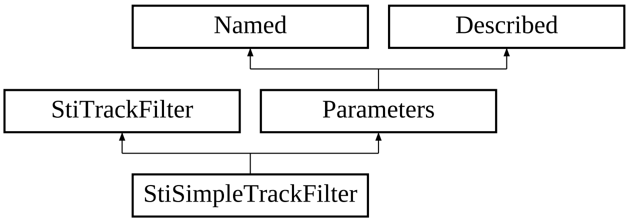
The documentation for this class was generated from the following files:

- StiMaker/**StiRootSimpleTrackFilter.h**
- StiMaker/**StiRootSimpleTrackFilter.cxx**

4.51 StiSimpleTrackFilter Class Reference

`#include <StiSimpleTrackFilter.h>`

Inheritance diagram for StiSimpleTrackFilter::



Public Types

- enum **StiTrackFilterIdentifier** { **kChi2** = 0, **kPhi**, **kPt**, **kP**, **k-PseudoRap**, **kNPts**, **kNGaps**, **kNToNmaxPts**, **kNTpcPts**, **kNSvt-Pts**, **kTpcDedx**, **kSvtDedx** }

Public Methods

- **StiSimpleTrackFilter** ()
- **StiSimpleTrackFilter** (const string &name, const string &description)
- virtual ~**StiSimpleTrackFilter** ()
- virtual void **initialize** ()
- bool **accept** (**StiTrack** ∗track) const

4.51.1 Detailed Description

StiSimpleTrackFilter. Instances of this class are track filter enabling filtering of tracks based on their pseudo-rapidity, pt, and other features that do not require the track particle species to be identified.

kChi2 : Chisquare of the track kPhi : Azimuthal angle of the track kPt : Transverse Momentum of the track (GeV/c) kPseudoRapidity : Pseudo-rapidity of the track kRapidity : Rapidity kNPts : Number of nodes (with hits) on the track kNFitPts : Number of nodes (with hits) used in the fit of the track k-NGaps : Number of active layers without hits on the track kFitToTotalPts : Ratio of fit to total points on the track kPrimaryDca : Distance to closest approach (DCA) to primary vertex kNTpcPts : Number of TPC hits kNSvtPts : Number of SVT hits kTpcDedx : Value of TPC truncated Dedx Mean kSvt-Dedx : Value of SVT truncated Dedx Mean kTrackType : Type of Track 0 -

All types selected - same as not using this filter 1 - Primary tracks only - track which include primary vertex 2 - Secondary tracks only - tracks which do not primary vertex kCharged : Charge of track -1 - Negative only 0 - Neutral only 1 - Positive only 2 - Negative or Positive only other - All accepted - same as not using this filter

Definition at line 40 of file StiSimpleTrackFilter.h.

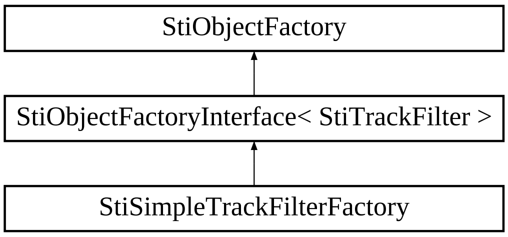
The documentation for this class was generated from the following files:

- Sti/**StiSimpleTrackFilter.h**
- Sti/**StiSimpleTrackFilter.cxx**

4.52 StiSimpleTrackFilterFactory Class Reference

`#include <StiSimpleTrackFilter.h>`

Inheritance diagram for StiSimpleTrackFilterFactory::



Public Methods

- **StiSimpleTrackFilterFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

  This is the only constructor available.

- virtual ~**StiSimpleTrackFilterFactory** ()

  Default destructor.

Protected Methods

- virtual void ∗ **makeNewObject** () const

  Return a pointer to a new **StiSimpleTrackFilter** (p. 158) *object on the heap.*

4.52.1 Detailed Description

StiSimpleTrackFilter (p. 158) factory

Definition at line 65 of file StiSimpleTrackFilter.h.

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

- Sti/**StiSimpleTrackFilter.h**
- Sti/**StiSimpleTrackFilter.cxx**

4.53 StiStEventFiller Class Reference

`#include <StiStEventFiller.h>`

Public Methods

- **StiStEventFiller** ()
- virtual ~**StiStEventFiller** ()
- StEvent * **fillEvent** (StEvent *, **StiTrackContainer** *)

  Fill the event from the track store.

- StEvent * **fillEventPrimaries** (StEvent *, **StiTrackContainer** *)
- void **fillDetectorInfo** (StTrackDetectorInfo *detInfo, const **StiTrack** *kTrack)
- void **fillGeometry** (StTrack *track, const **StiTrack** *kTrack, bool outer)
- void **fillFitTraits** (StTrack *track, const **StiTrack** *kTrack)
- void **fillPidTraits** (StTrack *track, const **StiTrack** *kTrack)
- void **filldEdxInfo** (**StiDedxCalculator** &, StTrack *track, const **StiTrack** *kTrack)
- void **fillTrack** (StTrack *track, const **StiTrack** *kTrack)
- unsigned short **encodedStEventFitPoints** (const **StiTrack** *kTrack)
- float **impactParameter** (const **StiTrack** *kTrack)

4.53.1 Detailed Description

StiStEventFiller is a utilitity class meant to properly convert **StiTrack** (p. 166) objects into StTrack (Global/Primary) objects and hang these on the StEvent Track-node.

Author:

Manuel Calderon de la Barca Sanchez (Yale Software)

Note:

Definition at line 77 of file StiStEventFiller.h.

4.53.2 Member Function Documentation

4.53.2.1 StEvent * StiStEventFiller::fillEvent (StEvent * *e*, StiTrackContainer * *t*)

Fill the event from the track store.

Algorithm: Loop over all tracks in the **StiTrackContainer** (p. 169), doing for each track:

- Create a new global track and associated information (see below) and set its data members according to the **StiTrack** (p. 166), can be done in a StGlobalTrack constructor
- Hang the new track to the StTrackNode container in StEvent, this creates a new entry in the container, the global track is now owned by it.

  In addition to the StGlobalTrack, we need to create the following objects (owned by it): StTrackTopologyMap StTrackFitTraits StTrackGeometry (2 are needed, one at first point, one at last point) (note: StHelixModel is implementation of the StTrackGeometry abstract class)

The track also owns a container of PidTraits, this algorithm will not fill this container.

And set up links to: StTrackDetectorInfo (owned by StEvent, StSPtrVecTrack-DetectorInfo) StTrackNode (owned by StEvent, StSPtrVecTrackNode) These links are track -> detector info track <-> track node

Skeleton of the algorithm:

`StSPtrVecTrackNode& trNodeVec = mEvent->trackNodes();`

`StSPtrVecTrackDetectorInfo& detInfoVec = mEvent->trackDetector-Info();`

`for (trackIterator trackIt = mTrackStore->begin(); trackIt != mTrackStore->end(); ++trackIt) {`

StiKalmanTrack (p. 105)`* kTrack = (*trackIt).second; // the container is a <map>, need second entry of <pair>`

`StTrackDetectorInfo* detInfo = new StTrackDetectorInfo();`

`fillDetectorInfo(detInfo,kTrack);`

`detInfoVec.push_back(detInfo);`

`StTrackNode* trackNode = new StTrackNode;`

`trNodeVec.push_back(trackNode);`

`StGlobalTrack* gTrack = new StGlobalTrack();`

`fillGlobalTrack(gTrack,kTrack);`

`set up relationships between objects`

`gTrack->setDetectorInfo(detInfo);`

`gTrack->setNode(trackNode);`

`trackNode->AddTrack(gTrack);`

}

The creation of the various objects needed by StGlobalTrack are taken care of in the methods: fillTopologyMap(), fillGeometry(), fillFitTraits(), which are called within fillGlobalTrack().

Definition at line 216 of file StiStEventFiller.cxx.

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

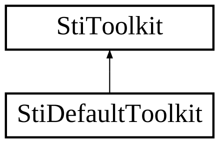
- StiMaker/**StiStEventFiller.h**
- StiMaker/**StiStEventFiller.cxx**

4.54 StiToolkit Class Reference

Definition of toolkit.

`#include <StiToolkit.h>`

Inheritance diagram for StiToolkit::



Public Methods

- virtual StiObjectFactoryInterface< **StiHit** > ∗ **getHitFactory** ()=0
- virtual StiObjectFactoryInterface< **StiKalmanTrack** > ∗ **getTrackFactory** ()=0
- virtual StiObjectFactoryInterface< StiMcTrack > ∗ **getMcTrackFactory** ()=0
- virtual StiObjectFactoryInterface< **StiKalmanTrackNode** > ∗ **getTrackNodeFactory** ()=0
- virtual StiObjectFactoryInterface< StiDetector > ∗ **getDetectorFactory** ()=0
- virtual StiObjectFactoryInterface< StiDetectorNode > ∗ **getDetectorNodeFactory** ()=0
- virtual StiObjectFactoryInterface< Parameter > ∗ **getParameterFactory** ()=0
- virtual StiObjectFactoryInterface< **StiTrackFilter** > ∗ **getTrackFilterFactory** ()=0
- virtual **StiDetectorContainer** ∗ **getDetectorContainer** ()=0
- virtual **StiHitContainer** ∗ **getHitContainer** ()=0
- virtual **StiTrackContainer** ∗ **getTrackContainer** ()=0
- virtual **StiTrackContainer** ∗ **getMcTrackContainer** ()=0
- virtual StiGeometryTransform ∗ **getGeometryTransform** ()=0
- virtual StiCoordinateTransform ∗ **getCoordinateTransform** ()=0
- virtual StiDetectorFinder ∗ **getDetectorFinder** ()=0
- virtual StiSeedFinder ∗ **getTrackSeedFinder** ()=0
- virtual **StiTrackFinder** ∗ **getTrackFinder** ()=0
- virtual StiTrackFitter ∗ **getTrackFitter** ()=0
- virtual **StiTrackMerger** ∗ **getTrackMerger** ()=0
- virtual StiDisplayManager ∗ **getDisplayManager** ()=0

- virtual StiIOBroker ∗ **getIOBroker** ()=0
- virtual StAssociationMaker ∗ **getAssociationMaker** ()=0
- virtual void **setAssociationMaker** (StAssociationMaker ∗a)=0
- virtual **StiHitFiller** ∗ **getHitFiller** ()=0
- virtual **StiHitErrorCalculator** ∗ **getHitErrorCalculator** ()=0

Static Public Methods

- StiToolkit ∗ **instance** ()
- void **kill** ()

Static Protected Attributes

- StiToolkit ∗ **sInstance** = 0

4.54.1 Detailed Description

Definition of toolkit.

Definition at line 53 of file StiToolkit.h.

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

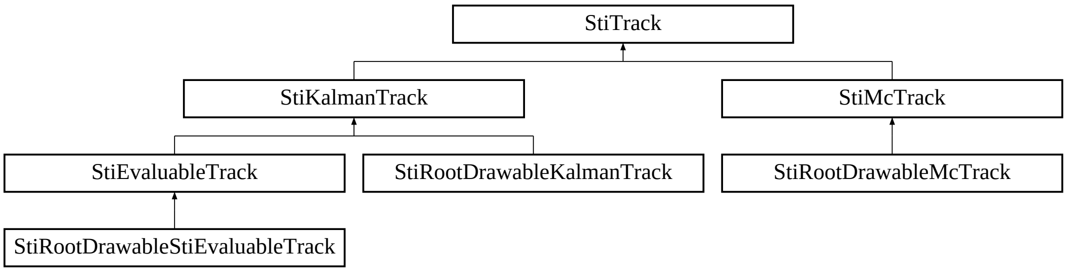
- Sti/**StiToolkit.h**
- Sti/**StiToolkit.cxx**

4.55 StiTrack Class Reference

Abstract definition of a Track.

`#include <StiTrack.h>`

Inheritance diagram for StiTrack::



Public Types

- enum **StiTrackProperty** { **kCharge** = 0, **kMass**, **kChi2**, **kDca2**, **kDca3**, **kFlag**, **kPrimaryDca**, **kPointCount**, **kFitPointCount**, **k-GapCount**, **kTrackLength**, **kMaxPointCount**, **kisPrimary**, **kTpc-Dedx**, **kSvtDedx**, **kCurvature**, **kP**, **kPt**, **kRapidity**, **kPseudo-Rapidity**, **kPhi**, **kTanL** }

Public Methods

- **StiTrack** ()
- virtual ~**StiTrack** ()
- virtual void **fit** (int direction=kOutsideIn)
- virtual bool **find** (int direction=kOutsideIn)
- virtual void **reset** ()=0
- virtual void **getMomentum** (double p[3], double e[6]) const=0
- virtual StThreeVector< double > **getMomentumAtOrigin** () const=0
- virtual StThreeVector< double > **getMomentumNear** (double x)=0
- virtual StThreeVector< double > **getHitPositionNear** (double x) const=0
- virtual double **getCurvature** () const=0
- virtual double **getP** () const=0
- virtual double **getPt** () const=0
- virtual double **getRapidity** () const=0
- virtual double **getPseudoRapidity** () const=0
- virtual double **getPhi** () const=0

- virtual double **getTanL** () const=0
- virtual double **getDca** (**StiHit** *h=0) const=0
- virtual double **getDca2** (StiTrack *t) const=0
- virtual double **getDca3** (StiTrack *t) const=0
- virtual int **getPointCount** () const=0
- virtual int **getFitPointCount** () const=0
- virtual int **getGapCount** () const=0
- virtual int **getMaxPointCount** () const=0
- virtual int **getSeedHitCount** () const=0

  number of hits used to seed the track.

- virtual void **setSeedHitCount** (int c)=0
- virtual double **getTrackLength** () const=0
- virtual vector< StMeasuredPoint *> **stHits** () const=0
- virtual double **getMass** () const=0

  Get mass of the particle that produced this track.

- virtual int **getCharge** () const=0

  Get charge of the particle that produced this track.

- virtual double **getChi2** () const=0

  Get chi2 of this track.

- virtual void **setFlag** (long v)=0
- virtual long **getFlag** () const=0
- virtual double **getValue** (int key)

Static Public Methods

- void **setTrackFinder** (**StiTrackFinder** *finder)
- void **setTrackFitter** (StiTrackFitter *fitter)
- **StiTrackFinder** * **getTrackFinder** ()
- StiTrackFitter * **getTrackFitter** ()

Static Protected Attributes

- **StiTrackFinder** * **trackFinder** = 0
- StiTrackFitter * **trackFitter** = 0

Friends

- ostream & **operator**<< (ostream &os, const StiTrack &track)

4.55.1 Detailed Description

Abstract definition of a Track.

Abstract definition of a track used in the Sti package.

Author:
Claude A Pruneau (Wayne State University)

Definition at line 59 of file StiTrack.h.

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

- Sti/**StiTrack.h**
- Sti/**StiTrack.cxx**

4.56 StiTrackContainer Class Reference

`#include <StiTrackContainer.h>`

Public Methods

- void **add** (**StiTrack** ∗track)

  Add given track to the container.

- void **push_back** (**StiTrack** ∗)

  Preserve simple interface to add tracks.

- **StiTrackContainer** ()
- virtual ∼**StiTrackContainer** ()

4.56.1 Detailed Description

StiTrackContainer is meant to provide the interface between the StiTracker and the persistency model. That is, StiTrackContainer will be the only thing that StiTracker will have to know about and StiTrackContainer will interface to (most likely) StEvent, unless we decide to make StiTrackContainer persistent.

StiTrackContainer is polymorphic and can hold all forms of **StiTrack** (p. 166) objects. That includes in particular **StiKalmanTrack** (p. 105) and StiMcTrack.

Author:

M.L. Miller (Yale Software)

Definition at line 36 of file StiTrackContainer.h.

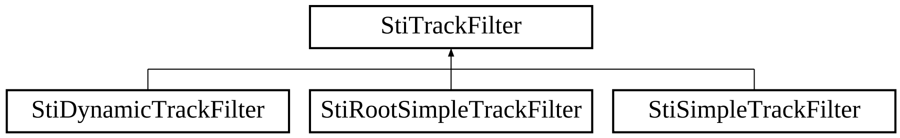
The documentation for this class was generated from the following files:

- Sti/**StiTrackContainer.h**
- Sti/**StiTrackContainer.cxx**

4.57 StiTrackFilter Class Reference

#include <StiTrackFilter.h>

Inheritance diagram for StiTrackFilter::



Public Methods

- **StiTrackFilter** ()
- virtual ∼**StiTrackFilter** ()
- virtual void **initialize** ()=0
- virtual bool **accept** (**StiTrack** ∗track) const=0
- bool **filter** (**StiTrack** ∗track)
- void **reset** ()

  Reset counters to zero.

- int **getAnalyzedTrackCount** () const

  Returns the number of tracks analyzed by this filter since last reset.

- int **getAcceptedTrackCount** () const

  Returns the number of tracks accpeted by this filter since last reset.

Protected Attributes

- int **analyzedTrackCount**

  Number of tracks analyzed since last reset.

- int **acceptedTrackCount**

  Number of tracks found acceptable since last reset.

4.57.1 Detailed Description

Base class defining a track filtering mechanism

Abstract base class defining a track filtering mechanism. This class cannot be instantiated given it features pure virtual methods. As such it provides the basic elements of a track filter without actually doing filtering itself - the method **bool accept(StiTrack * track)** which does the filtering is pure virtual, i.e. must be implemented in a derived class to satisfy the specific needs of an application.

The class features two protected data members **analyzedTrackCount** and **acceptedTrackCount** corresponding respectively to the number of tracks analyzed and accepted by this filter. The values can reset to zero by a call to the **void reset()** (p. 170) method. They can be accessed with the **int getAnalyzedTrackCount()** (p. 170) and **int getAcceptedTrackCount()** (p. 170) accessor methods respectively. Note that these two counters are not incremented by the **bool accept(StiTrack * track)** method. Users wishing to the use these variables for accounting should use the inlined method **filter(StiTrack * track)** (p. 171) which internally calls the "accept" method and also increments **analyzedTrackCount** for all tracks analyzed, and **acceptedTrackCount** for tracks for which "accept" return true.

Note: The **accept** method is declared "const" to emphasize that it does not increment counters but only returns a bool value representative of the acceptability of this track.

Definition at line 34 of file StiTrackFilter.h.

4.57.2 Member Function Documentation

4.57.2.1 bool StiTrackFilter::filter (StiTrack * *track*) `[inline]`

Filter the given track and return true it is found acceptable.

Determine whether the given track is found acceptable by a call to the **accept** method. Increments **analyzedTrackCount** by one for all tracks analyzed. Increments **acceptedTrackCount** by one if **accept** returns true.

Definition at line 67 of file StiTrackFilter.h.

References acceptedTrackCount, and analyzedTrackCount.

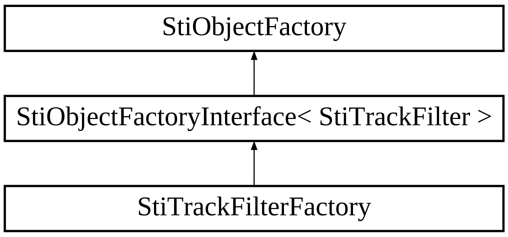
The documentation for this class was generated from the following files:

- Sti/**StiTrackFilter.h**
- Sti/**StiTrackFilter.cxx**

4.58 StiTrackFilterFactory Class Reference

`#include <StiTrackFilter.h>`

Inheritance diagram for StiTrackFilterFactory::



Public Methods

- **StiTrackFilterFactory** (const string &newName, int original=-1, int incremental=-1, int maxInc=-1)

    This is the only constructor available.

- virtual ~**StiTrackFilterFactory** ()

    Default destructor.

4.58.1 Detailed Description

StiTrackFilter (p. 170) factory

Definition at line 96 of file StiTrackFilter.h.

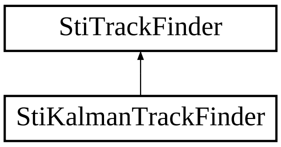
The documentation for this class was generated from the following files:

- Sti/**StiTrackFilter.h**
- Sti/**StiTrackFilter.cxx**

4.59 StiTrackFinder Class Reference

`#include <StiTrackFinder.h>`

Inheritance diagram for StiTrackFinder::



Public Methods

- virtual void **findTracks** ()=0
- virtual void **fitTracks** ()=0
- virtual void **extendTracksToVertex** (**StiHit** ∗vertex)=0
- virtual void **findNextTrack** ()=0
- virtual void **fitNextTrack** ()=0
- virtual void **reset** ()=0
- virtual bool **isValid** (bool debug=false) const=0
- virtual int **getTrackSeedFoundCount** () const=0
- virtual int **getTrackFoundCount** () const=0
- virtual int **getTrackFoundCount** (**StiTrackFilter** ∗filter) const=0
- virtual bool **find** (**StiTrack** ∗track, int direction)=0
- virtual **StiTrackFilter** ∗ **getTrackFilter** () const=0
- virtual **StiTrackFilter** ∗ **getGuiTrackFilter** () const=0
- virtual **StiTrackFilter** ∗ **getGuiMcTrackFilter** () const=0

4.59.1 Detailed Description

A purely abstract class defining the interface to a track finder.

Definition at line 11 of file StiTrackFinder.h.

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

- Sti/**StiTrackFinder.h**

4.60 StiTrackLessThan Struct Reference

Define the Less-Than operator for track ordering in the track container.

`#include <StiTrackContainer.h>`

Public Methods

- bool **operator()** (const **StiTrack** ∗lhs, const **StiTrack** ∗rhs) const

4.60.1 Detailed Description

Define the Less-Than operator for track ordering in the track container.

Definition at line 28 of file StiTrackContainer.h.

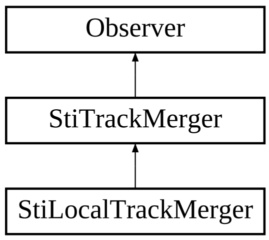
The documentation for this struct was generated from the following files:

- Sti/**StiTrackContainer.h**
- Sti/**StiTrackContainer.cxx**

4.61 StiTrackMerger Class Reference

`#include <StiTrackMerger.h>`

Inheritance diagram for StiTrackMerger::



Public Methods

- **StiTrackMerger** (**StiTrackContainer** ∗)

  One must provide a valid pointer to the track container.

- virtual ∼**StiTrackMerger** ()
- virtual void **mergeTracks** ()=0

  Merge the tracks in the track container.

Protected Methods

- **StiTrackMerger** ()
- virtual void **getNewState** ()=0

Protected Attributes

- **StiTrackContainer** ∗ **mTrackStore**

4.61.1 Detailed Description

StiTrackMerger is a pure virtual class that defines the interface for a class that encapsulates a track-merging algorithm. It's purpose is to combine track segments that belong to the same trajectory, but were initially reconstructed as seperate pieces.

Author:

M.L. Miller (Yale Software)

Note:

All information passed to and from an instance of StiTrackMerger is performed via the pointer to **StiTrackContainer** (p. 169).

Definition at line 19 of file StiTrackMerger.h.

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

- Sti/**StiTrackMerger.h**
- Sti/**StiTrackMerger.cxx**

4.62 StiTrackPairInfo Class Reference

`#include <StiTrackPairInfo.h>`

Public Methods

- **StiTrackPairInfo** ()
- **StiTrackPairInfo** (const **StiTrack** ∗, const trackPing &)
- virtual ∼**StiTrackPairInfo** ()
- const StMcTrack ∗ **partnerMcTrack** () const
- const **StiTrack** ∗ **partnerTrack** () const
- unsigned int **commonTpcHits** () const
- unsigned int **commonSvtHits** () const
- unsigned int **commonFtpcHits** () const
- void **setPartnerMcTrack** (const StMcTrack ∗)
- void **setPartnerTrack** (const **StiTrack** ∗)
- void **setCommonTpcHits** (unsigned int)
- void **setCommonSvtHits** (unsigned int)
- void **setCommonFtpcHits** (unsigned int)

4.62.1 Detailed Description

StiTrackPairInfo is a mirror of class StTrackPairInfo. Whereas StTrackPair-Info (used by StAssociationMaker) maps an StiMcTrack to a StGlobalTrack, StiTrackPairInfo (used by StiEventAssociator) maps an StiMcTrack to a **Sti-Track** (p. 166). That is the only difference.

Author:
M.L. Miller (Yale Software)

Definition at line 22 of file StiTrackPairInfo.h.

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

- StiEvaluator/**StiTrackPairInfo.h**
- StiEvaluator/**StiTrackPairInfo.cxx**

Chapter 5

ITTF File Documentation

5.1 Sti/StiCoordinateTransform.h File Reference

Definition of the StiLocal<->Global coordinate transforms.

`#include <vector>`

`#include "StarClassLibrary/StThreeVector.hh"`

Compounds

- class **StiCoordinateTransform**

5.1.1 Detailed Description

Definition of the StiLocal<->Global coordinate transforms.

@class StiCoordinateTransform

Encapsulates all coordinate transformations to and from the Sti Local system. The Sti local system is defined as followed:

- z points in the direction of global (star magnet iron) z
- x points radially outward from the center of the detector plane
- y follows the right hand rule.
- origin = global origin

Currently, all the transformations are "ideal", in that they assume all detectors are centered at (0, 0, 0) in global coords and have their z axis along global z.

This is done because the Sti track model assumes ideal rotations when going between detectors.

Author:
Ben Norman, Kent State University

Date:
March 2002

Definition in file **StiCoordinateTransform.h**.

5.2 Sti/StiIsActiveFunctor.h File Reference

function object for determine a detector's active regions.

Compounds

- struct **StiIsActiveFunctor**

5.2.1 Detailed Description

function object for determine a detector's active regions.

@class StiIsActiveFunctor

Returns whether or not a given detector is active (capable of providing hit information) as a function of local z and y. Local x is not required because the detector is considered a surface, not a solid.

Author:

Ben Norman, Kent State University

Date:

March 2002

Definition in file **StiIsActiveFunctor.h**.

5.3 Sti/StiKalmanTrack.h File Reference

Definition of Kalman Track.

```
#include "StThreeVector.hh"
#include "StThreeVectorF.hh"
#include "StThreeVectorD.hh"
#include <math.h>
#include <vector>
#include <stdexcept>
#include "StDetectorId.h"
#include "StiObjectFactoryInterface.h"
#include "StiKalmanTrackNode.h"
#include "StiHitContainer.h"
#include "StiKTNIterator.h"
#include "StiHit.h"
#include "StiTrack.h"
#include "StiKalmanTrackFinderParameters.h"
```

Compounds

- class **StiKalmanTrack**

  Definition of Kalman Track.

Defines

- #define **StiKalmanTrack_H** 1

5.3.1 Detailed Description

Definition of Kalman Track.

Subclass of **StiTrack** (p. 166) defining a Kalman track to be used by the Kalman Track Finder.

Author:

Claude A Pruneau, Wayne State University,

Date:

March 2001 \copyright 2001, STAR Experiment at BNL, All rights reserved.

Definition in file **StiKalmanTrack.h**.

5.4 Sti/StiLocalCoordinate.h File Reference

Represents coordinates in the Sti Local system.

`#include <iostream.h>`

`#include "StThreeVector.hh"`

Compounds

- class **StiLocalCoordinate**

Functions

- ostream & **operator**<< (ostream &, const StiLocalCoordinate &)

5.4.1 Detailed Description

Represents coordinates in the Sti Local system.

@class StiLocalCoordinate

The Sti local system is defined as followed:

- z points in the direction of global (star magnet iron) z
- x points radially outward from the center of the detector plane
- y follows the right hand rule.
- origin = global origin

Author:
Ben Norman, Kent State University

Date:
March 2002

Definition in file **StiLocalCoordinate.h**.

5.5 Sti/StiNeverActiveFunctor.h File Reference

function object which always returns false.

`#include "StiIsActiveFunctor.h"`

Compounds

- struct **StiNeverActiveFunctor**

5.5.1 Detailed Description

function object which always returns false.

@class StiNeverActiveFunctor

Author:
Ben Norman, Kent State University

Date:
March 2002

Definition in file **StiNeverActiveFunctor.h**.

5.6 Sti/StiSvtIsActiveFunctor.h File Reference

function object for determine a SVT ladder's active regions.

`#include "StiIsActiveFunctor.h"`

Compounds

- struct **StiSvtIsActiveFunctor**

5.6.1 Detailed Description

function object for determine a SVT ladder's active regions.

@class StiSvtIsActiveFunctor

Author:
Ben Norman, Kent State University

Date:
March 2002

Definition in file **StiSvtIsActiveFunctor.h**.

5.7 Sti/StiToolkit.h File Reference

Abstract interface for a STI toolkit.

`#include <string>`

`#include "StiFactoryTypes.h"`

Compounds

- class **StiToolkit**

  Definition of toolkit.

Defines

- #define **StiToolkit_H** 1

5.7.1 Detailed Description

Abstract interface for a STI toolkit.

Author:

Claude A Pruneau, Wayne State University,

Date:

March 2002 @copyright 2002, STAR Experiment at BNL, All rights reserved.

Definition in file **StiToolkit.h**.

5.8 Sti/StiTpcIsActiveFunctor.h File Reference

function object for determine a TPC padrow's active regions.

```
#include "StiIsActiveFunctor.h"
```

Compounds

- class **StiTpcIsActiveFunctor**

5.8.1 Detailed Description

function object for determine a TPC padrow's active regions.

@class StiTpcIsActiveFunctor

Author:
Ben Norman, Kent State University

Date:
March 2002

Definition in file **StiTpcIsActiveFunctor.h**.

5.9 StiMaker/StiDefaultToolkit.cxx File Reference

Default Implementation of the **StiToolkit** (p. 164) Abstract interface.

```
#include "StiDefaultToolkit.h"
#include "../Sti/StiFactoryTypes.h"
#include "../Sti/StiMcTrack.h"
#include "../StiGui/StiGuiFactoryTypes.h"
#include "../StiGui/StiRootDrawableHitContainer.h"
#include "../Sti/StiHitContainer.h"
#include "../Sti/StiDetectorContainer.h"
#include "../Sti/StiDetectorFinder.h"
#include "../Sti/StiTrackContainer.h"
#include "../Sti/StiGeometryTransform.h"
#include "../Sti/StiCoordinateTransform.h"
#include "../Sti/StiTrackSeedFinder.h"
#include "../Sti/StiTrackFinder.h"
#include "../Sti/StiTrackFitter.h"
#include "../Sti/StiSimpleTrackFilter.h"
#include "../Sti/StiKalmanTrackFitter.h"
#include "../Sti/StiKalmanTrackFinder.h"
#include "../Sti/StiTrackMerger.h"
#include "../Sti/StiCompositeSeedFinder.h"
#include "../Sti/StiLocalTrackMerger.h"
#include "../Sti/StiDisplayManager.h"
#include "../Sti/StiIOBroker.h"
#include "../Sti/StiDynamicTrackFilter.h"
#include "../Sti/StiHitFiller.h"
#include "../Sti/Parameter.h"
#include "../StiMaker/RootEditableParameter.h"
#include "../StiMaker/StiRootIOBroker.h"
```

```
#include "../StiMaker/StiRootSimpleTrackFilter.h"
#include "../StiGui/StiRootDisplayManager.h"
#include "../StiGui/StiRootDrawableMcTrack.h"
#include "../StiEvaluator/StiEvaluator.h"
#include "../StiEvaluator/StiEventAssociator.h"
#include "../Sti/StiEvaluableTrackSeedFinder.h"
#include "StAssociationMaker/StAssociationMaker.h"
#include "../Sti/StiHitErrorCalculator.h"
```

5.9.1 Detailed Description

Default Implementation of the **StiToolkit** (p. 164) Abstract interface.

Author:

Claude A Pruneau, Wayne State University,

Date:

March 2001 @copyright 2001, STAR Experiment at BNL, All rights reserved.

Definition in file **StiDefaultToolkit.cxx**.

5.10 StiMaker/StiDefaultToolkit.h File Reference

Default Implementation of the **StiToolkit** (p. 164) Abstract interface.

```
#include "../Sti/StiToolkit.h"
```

Compounds

- class **StiDefaultToolkit**
  Definition of toolkit.

Defines

- #define **StiDefaultToolkit_H** 1

5.10.1 Detailed Description

Default Implementation of the **StiToolkit** (p. 164) Abstract interface.

Author:
Claude A Pruneau, Wayne State University,

Date:
March 2001 @copyright 2001, STAR Experiment at BNL, All rights reserved.

Permission to use, copy, modify and distribute this software and its documentation strictly for non-commercial purposes is hereby granted without fee, provided that the above copyright notice appears in all copies and that both the copyright notice and this permission notice appear in the supporting documentation. The authors make no claims about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Definition in file **StiDefaultToolkit.h**.

Chapter 6

ITTF Example Documentation

6.1 CombinationIterator_ex.cxx

```
//CombinationIterator_ex.cxx

//This file is an example of how to use the combination iterator.

#include <iostream>
using std::cout;
using std::endl;

#include <vector>
using std::vector;

#include <algorithm>
using std::copy;

using std::ostream_iterator;

#include <time.h>

#include "CombinationIterator.h"

int main()
{
    cout <<"Hello World!"<<endl;

    vector<double> vec1;
    vector<double> vec2;
    vector<double> vec3;

    for (double i=1.; i<=3.; ++i) {
```

```
        vec1.push_back(i);
        vec2.push_back(10.*i);
        vec3.push_back(100.*i);
    }

    CombinationIterator<double> myIt;
    myIt.push_back(vec1.begin(), vec1.end());
    myIt.push_back(vec2.begin(), vec2.end());
    myIt.push_back(vec3.begin(), vec3.end());

    clock_t start = clock();
    for( ; myIt!=myIt.end(); ++myIt) {
        vector<double> combo = *myIt;
        cout <<"\n\t --- Next Combination --- "<<endl;
        copy((*myIt).begin(), (*myIt).end(),
             ostream_iterator<double>(cout, " "));
        cout <<endl;
    }

    clock_t stop = clock();
    double time = (stop-start)/CLOCKS_PER_SEC;

    cout <<"\n\nElapsed Time:\t"<<time<<" seconds"<<endl;

    return 1;
}
```

6.2 StFastLineFitter_ex.cxx

```
//StFastLineFitter_ex.cxx

#include "Sti/StiFastLineFitter.h"
#include <iostream>
#include <cmath>

//This is meant as an example of how to use StFastLineFitter_ex.cxx
int main()
{
    StFastLineFitter myFitter;
    myFitter.clear();

    double slope = 3.;
    double intercept = 7.;
    for (double x=1.; x<=10.; ++x) {
        double y=slope*x + intercept;
        double weight = sqrt(y);
        myFitter.addPoint(x, y, weight);
    }
    bool rc = myFitter.fit();
    if (myFitter.rc()!=0) {
        cout <<"ERROR:\tFit failed."<<endl;
    }
    myFitter.print();

    return 1;
}
```

6.3 StiCompositeLeafIterator_ex.cxx

```
void main()
{
    StiCompositeTreeNode<MyFoo> root;

    //... code to add daughters to root ...

    StiCompositeLeafIterator<MyFoo> leafIt(root);
    copy(leafIt.const_begin(), leafIt.const_end(),
         ostream_iterator(cout, " ")); //Stream nodes to screen
    //Now Stream first leaf:
    cout <<"First leaf: "<<*(leafIt.const_begin());

    return 1;
}
```

6.4 StiDedxCalculator_ex.cxx

6.5 StiDetectorContainer_ex.cxx

```
//StiDetectorContainer_ex.cxx

//This is an example of how to use StiDetectorContainer
int main()
{
    //The StiDetector factory
    StiObjectFactoryInterface<StiDetector>* detectorfactory;
    //Decide which type of derived factory to create
    if (UseGui==true)
        mdetectorfactory = new StiRDDetectorFactory("RDDetectorFactory");
    else
        mdetectorfactory = new StiDetectorFactory("DetectorFactory");
    detectorfactory->reset();

    //The DetectorNodeFactory
    StiObjectFactoryInterface<StiDetectorNode>* datanodefactory;
    datanodefactory = new StiDetectorNodeFactory("DetectorNodeFactory");
    datanodefactory->reset();

    //The Detector Tree
    StiDetectorContainer& store = *(StiDetectorContainer::instance());
    store.buildDetectors(datanodefactory, detectorfactory);
    store.reset();

    //Now navigate through the detector:
    store.setToPosition(1000000.); //get the outermost layer
    //Now more in until we can't go any further
    bool go=true;
    //This is how we get a StiDetector layer from an StiDetectorContainer instance
    StiDetector* layer= *store; //Dereference iterator
    while (go) {
        store.moveIn();
        if (layer!=*store) {
            //That means that we haven't been here yet.
            layer=*store; //Remember, continue
        }
        else {
            //That means that there is no place else to move into
            go=false;
        }
    }

    //Cleanup the objects owned by factories
    delete detectorfactory;
    delete datanodefactory;

    return 1;
    //Notice that we didin't call StiDetectorContainer::kill().
    //That is not a memory leak, it will be automatically destroyed when
    //we leave main
}
```

6.6 StiDetectorTreeBuilder_ex.cxx

```
//StiDetectorTreeBuilder_ex.cxx

//This function demonstrates how to use StiDetectorTreeBuilder.
//It is an excerpt from StiDetectorContainer.cxx
int main()
{
    //The StiDetector factory
    //We'll make a drawable represenation, to demonstrate the power of the factory
    //method
    StiObjectFactoryInterface<StiDetector>* detectorfactory;
    mdetectorfactory = new StiRDDetectorFactory("RDDetectorFactory");
    detectorfactory->reset();

    //The DetectorNodeFactory
    StiObjectFactoryInterface<StiDetectorNode>* datanodefactory;
    datanodefactory = new StiDetectorNodeFactory("DetectorNodeFactory");
    datanodefactory->reset();

    //This is just a useful typedef
    typedef StiCompositeTreeNode<StiDetector> data_node;

    //Instantiate the builder
    StiDetectorTreeBuilder mybuilder;
    //Actually build the 3d representation of the STAR detector in memory
    data_node* root = mybuilder.build(datanodefactory, detectorfactory);

    delete datanodefactory;
    delete detectorfactory;

    return 1;
}
```

6.7 StiEvaluableTrack_ex.cxx

```
//StiEvaluableTrack_ex.cxx

//This function is an example of how to use StiEvaluableTrack
//This is a code fragment (it is not meant to actually compile!)

int main()
{
    StAssociationMaker* mAssociationMaker=0;
    // ... code to make mAssociationMaker point to a valid instance

    //The track factory
    StiObjectFactoryInterface<StiKalmanTrack>* trackfactory =
        new StiRDEvaluableTrackFactory("StiRDEvaluableTrackFactory");

    StiEvaluableTrackSeedFinder* sf =
        new StiEvaluableTrackSeedFinder(mAssociationMaker);

    sf->setFactory(mtrackfactory);
    sf->setBuildPath("EvaluableSeedFinder.txt");
    sf->build();

    //now withing event loop
    for (int i=0; i<nEvents; ++i) {

        StMcEvent* mcEvent;

        //... code to point mcEvent to a valid StMcEvent ...

        sf->setEvent(mcEvent);

        while (sf->hasMore) {

            StiEvaluableTrack* track = sf->next();

            StTrackPairInfo* associatedPair = track->stTrackPairInfo();
            if (!associatedPair) {
                cout <<"StiEvaluator::evaluateForEvent(). ERROR:\t";
                cout <<"Associated Pair==0.  Abort"<<endl;
                return;
            }

            //Get the monte-carlo track
            StMcTrack = associatedPair->partnerMcTrack();

            //Get the StTrack
            StGlobalTrack* = associatedPair->partnerTrack();
        }
    }


    return 1;
}
```

6.8 StiHitContainer_ex.cxx

Index