Centella FrameWrok Reference Manual

0.1

Generated by Doxygen 1.1.3

Thu Jun 15 15:11:52 2000

Contents

Chapter 1

Centella FrameWrok Hierarchical Index

1.1 Centella FrameWrok Class Hierarchy

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

Chapter 2

Centella FrameWrok Compound Index

2.1 Centella FrameWrok Compound List

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

Chapter 3

Centella FrameWrok File Index

3.1 Centella FrameWrok File List

Here is a list of all files with brief descriptions:

Chapter 4

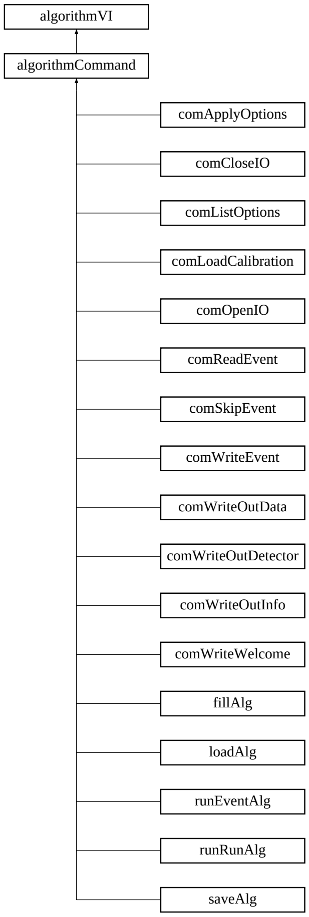
Centella FrameWrok Class Documentation

4.1 algorithmCommand Class Reference

Class to convert an algorithm into a command.

`#include <algorithmCommand.h>`

Inheritance diagram for algorithmCommand:



Public Methods

- virtual void **initialize** ()

  null the initalize.

- virtual void **finalize** ()

  null the finalize.

4.1.1 Detailed Description

Class to convert an algorithm into a command.

It has null methods **initialize**() (p. 9) and **finalize**() (p. 9). Its **run**() (p. 20) method is identicall to the **execute**() (p. 19) one.

Definition at line 13 of file algorithmCommand.h.

4.1.2 Member Function Documentation

4.1.2.1 void algorithmCommand::finalize () `[inline, virtual]`

null the finalize.

Reimplemented from **algorithmVI** (p. 20).

Definition at line 20 of file algorithmCommand.h.

4.1.2.2 void algorithmCommand::initialize () `[inline, virtual]`

null the initalize.

Reimplemented from **algorithmVI** (p. 20).

Definition at line 18 of file algorithmCommand.h.

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

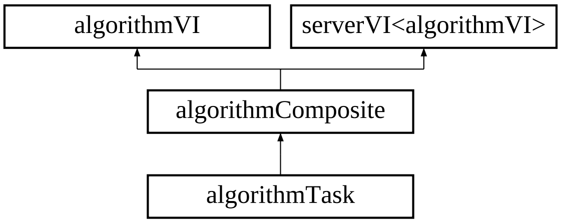
- **algorithmCommand.h**

4.2 algorithmComposite Class Reference

An algorithm composed by several algorithms.

`#include <algorithmComposite.h>`

Inheritance diagram for algorithmComposite:



Public Methods

- **algorithmComposite** ()

  default constructor.

- **~algorithmComposite** ()

  default destructor.

- virtual void **initialize** ()

  initializes all the algorithms.

- virtual void **execute** ()

  executes all the algorithms.

- virtual void **finalize** ()

  finalizes all the algorithms.

- virtual void **run** ()

  runs all the algorithmsa.

4.2.1 Detailed Description

An algorithm composed by several algorithms.

Is an algorithm.

Is a composite of algorithms.

The **run**() (p. 11) method runs all the **run**() (p. 11) methods of all the algorithms.

It used the **serverVI** (p. 152) template class to create the composite

Definition at line 20 of file algorithmComposite.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 algorithmComposite::algorithmComposite () `[inline]`

default constructor.

Definition at line 27 of file algorithmComposite.h.

4.2.2.2 algorithmComposite::~algorithmComposite () `[inline]`

default destructor.

Definition at line 29 of file algorithmComposite.h.

4.2.3 Member Function Documentation

4.2.3.1 void algorithmComposite::execute () `[inline, virtual]`

executes all the algorithms.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 34 of file algorithmComposite.h.

4.2.3.2 void algorithmComposite::finalize () `[inline, virtual]`

finalizes all the algorithms.

Reimplemented from **algorithmVI** (p. 20).

Definition at line 36 of file algorithmComposite.h.

4.2.3.3 void algorithmComposite::initialize () `[inline, virtual]`

initializes all the algorithms.

Reimplemented from **algorithmVI** (p. 20).

Definition at line 32 of file algorithmComposite.h.

4.2.3.4 void algorithmComposite::run () `[inline, virtual]`

runs all the algorithmsa.

Reimplemented from **algorithmVI** (p. 20).

Definition at line 38 of file algorithmComposite.h.

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

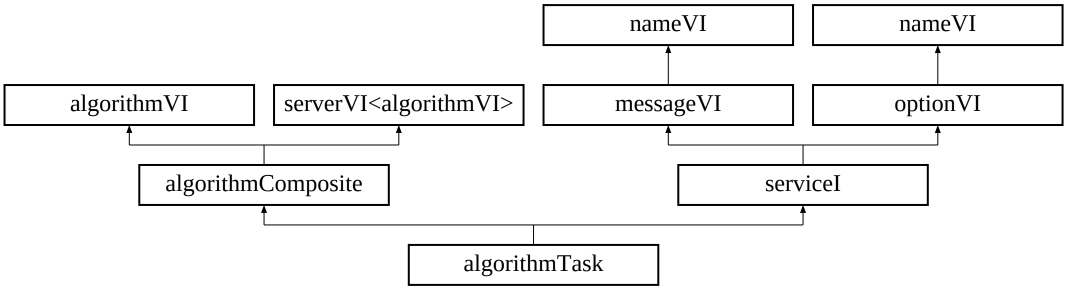
- **algorithmComposite.h**

4.3 algorithmTask Class Reference

A algorithmTask is a **algorithmComposite** (p. 10) with the services of the base class **serviceI** (p. 156).

`#include <algorithmTask.h>`

Inheritance diagram for algorithmTask:



Public Methods

- **algorithmTask** (std::string name)

  constructor - set a name.

- **~algorithmTask** ()

  destructor.

- virtual void **writeOut** () const

  information of this class.

Protected Methods

- virtual void **defineOption** ()

  define the options.

- virtual void **setOption** (std::string add, std::string algName)

  set options: declated operation: add **algorithmVI** (p. 18) *into* **algorithmComposite** (p. 10).

Private Attributes

- std::string **m_algName**

  dummy - last cut in composite, for **optionVI** (p. 114).

4.3.1 Detailed Description

A algorithmTask is a **algorithmComposite** (p. 10) with the services of the base class **serviceI** (p. 156).

- is an **algorithmComposite** (p. 10).
- uses all the services declared in **serviceI** (p. 156). To be included in the server every object of algorithmTask should have a name (**setName()** (p. 157) method of **serviceI** (p. 156)).
- the **setOption()** (p. 13) method from **optionVI** (p. 114) allows an external user (**optionManager** (p. 109)) to add dinamically algorithms to the **algorithmComposite** (p. 10).

Definition at line 18 of file algorithmTask.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 algorithmTask::algorithmTask (std::string *name*) `[inline]`

constructor - set a name.

Definition at line 25 of file algorithmTask.h.

4.3.2.2 algorithmTask::~algorithmTask ()

destructor.

4.3.3 Member Function Documentation

4.3.3.1 void algorithmTask::defineOption () `[inline, protected, virtual]`

define the options.

Reimplemented from **optionVI** (p. 116).

Definition at line 6 of file algorithmTask.cpp.

4.3.3.2 void algorithmTask::setOption (std::string *add*, std::string *algName*) `[inline, protected, virtual]`

set options: declated operation: add **algorithmVI** (p. 18) into **algorithmComposite** (p. 10).

Reimplemented from **optionVI** (p. 117).

Definition at line 12 of file algorithmTask.cpp.

4.3.3.3 void algorithmTask::writeOut () const `[inline, virtual]`

information of this class.

add an algorithm into the **algorithmComposite** (p. 10)

Reimplemented from **serviceI** (p. 158).

Definition at line 25 of file algorithmTask.cpp.

4.3.4 Member Data Documentation

4.3.4.1 std::string algorithmTask::m_algName `[private]`

dummy - last cut in composite, for **optionVI** (p. 114).

Definition at line 42 of file algorithmTask.h.

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

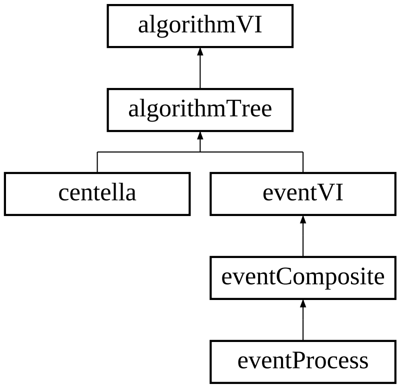
- **algorithmTask.h**
- **algorithmTask.cpp**

4.4 algorithmTree Class Reference

A multialgorithm that replace the initialize, execute and finalize methods for algorithmComposites.

`#include <algorithmTree.h>`

Inheritance diagram for algorithmTree:



Public Methods

- **algorithmTree** ()

  constructor - create the **algorithmComposite** (p. 10) *s.*

- **~algorithmTree** ()

  destructor - destroies the **algorithmComposite** (p. 10) *s.*

- virtual void **initialize** ()

  the initalize() method runs the ini **algorithmComposite** (p. 10).

- virtual void **execute** ()

  the **execute***()* (p. 16) *method runs the exe* **algorithmComposite** (p. 10).

- virtual void **finalize** ()

  the finalize method runs the fin **algorithmComposite** (p. 10).

Protected Methods

- **algorithmComposite**∗ **iniAlg** () const

  algorithmComposite (p. 10) *for the initialize algorithms.*

- **algorithmComposite**∗ **exeAlg** () const

  algorithmComposite (p. 10) *for the execute algorithms.*

- **algorithmComposite**∗ **finAlg** () const

  algorithmComposite (p. 10) *for the finalize algorithms.*

Private Attributes

- **algorithmComposite∗ m_ini**

  algorithmComposite (p. 10) *for the initialize algorithms.*

- **algorithmComposite∗ m_exe**

  algorithmComposite (p. 10) *for the execute algorithms.*

- **algorithmComposite∗ m_fin**

  algorithmComposite (p. 10) *for the finalize algorithms.*

4.4.1 Detailed Description

A multialgorithm that replace the initialize, execute and finalize methods for algorithmComposites.

The initalize(), **execute**() (p. 16) and **finalize**() (p. 17) methods runs their respective **algorithmComposite** (p. 10).

Definition at line 15 of file algorithmTree.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 algorithmTree::algorithmTree ()

constructor - create the **algorithmComposite** (p. 10) s.

Definition at line 4 of file algorithmTree.cpp.

4.4.2.2 algorithmTree::∼algorithmTree ()

destructor - destroies the **algorithmComposite** (p. 10) s.

Definition at line 12 of file algorithmTree.cpp.

4.4.3 Member Function Documentation

4.4.3.1 algorithmComposite ∗ algorithmTree::exeAlg () const `[inline, protected]`

algorithmComposite (p. 10) for the execute algorithms.

Definition at line 37 of file algorithmTree.h.

4.4.3.2 void algorithmTree::execute () `[inline, virtual]`

the **execute**() (p. 16) method runs the exe **algorithmComposite** (p. 10).

Reimplemented from **algorithmVI** (p. 19).

Definition at line 28 of file algorithmTree.h.

4.4.3.3 algorithmComposite * algorithmTree::finAlg () const `[inline, protected]`

algorithmComposite (p. 10) for the finalize algorithms.

Definition at line 39 of file algorithmTree.h.

4.4.3.4 void algorithmTree::finalize () `[inline, virtual]`

the finalize method runs the fin **algorithmComposite** (p. 10).

Reimplemented from **algorithmVI** (p. 20).

Definition at line 30 of file algorithmTree.h.

4.4.3.5 algorithmComposite * algorithmTree::iniAlg () const `[inline, protected]`

algorithmComposite (p. 10) for the initialize algorithms.

Definition at line 35 of file algorithmTree.h.

4.4.3.6 void algorithmTree::initialize () `[inline, virtual]`

the initalize() method runs the ini **algorithmComposite** (p. 10).

Reimplemented from **algorithmVI** (p. 20).

Definition at line 26 of file algorithmTree.h.

4.4.4 Member Data Documentation

4.4.4.1 algorithmComposite * algorithmTree::m_exe `[private]`

algorithmComposite (p. 10) for the execute algorithms.

Definition at line 46 of file algorithmTree.h.

4.4.4.2 algorithmComposite * algorithmTree::m_fin `[private]`

algorithmComposite (p. 10) for the finalize algorithms.

Definition at line 48 of file algorithmTree.h.

4.4.4.3 algorithmComposite * algorithmTree::m_ini `[private]`

algorithmComposite (p. 10) for the initialize algorithms.

Definition at line 44 of file algorithmTree.h.

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

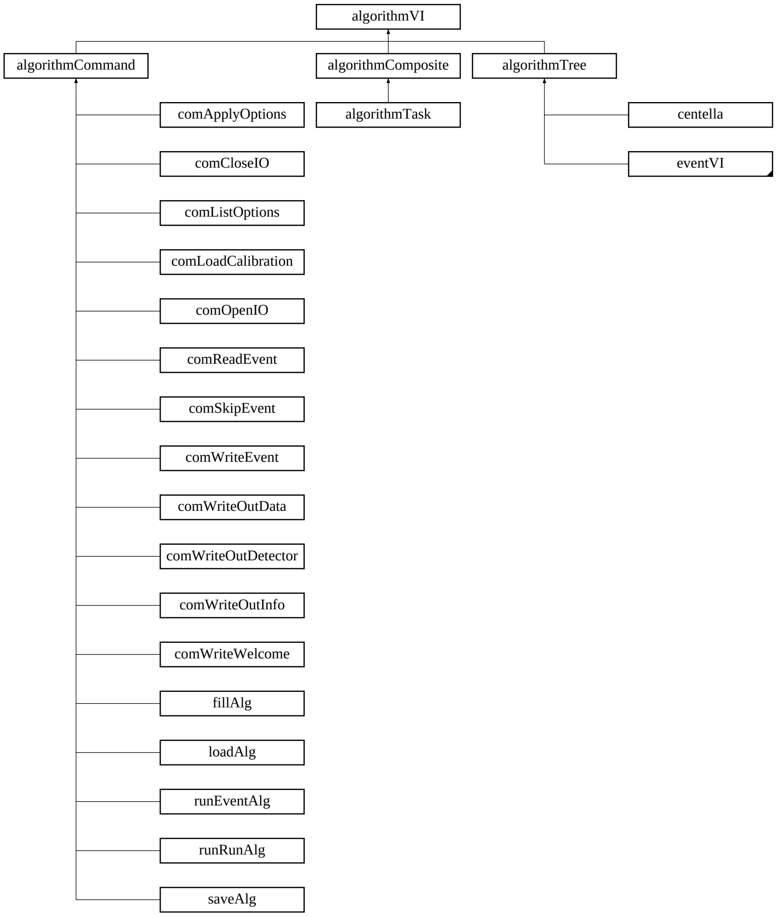
- **algorithmTree.h**
- **algorithmTree.cpp**

4.5 algorithmVI Class Reference

Base class for algorithms.

`#include <algorithmVI.h>`

Inheritance diagram for algorithmVI:



Public Methods

- **algorithmVI** ()

  default constructor.

- **~algorithmVI** ()

  default destructor.

- virtual void **run** ()

  virtual metho to run the algorithm.

- virtual void **initialize** () = 0

  pure virtual method initialize the algorithm.

- virtual void **execute** () = 0

pure virtual method to execute the algorithm.

- virtual void **finalize** () = 0

  pure virtual method to finalize the algorithm.

- virtual void **command** (std::string com)

  virtual method to execute a command.

4.5.1 Detailed Description

Base class for algorithms.

The algorithms of the program should inherit from this class.

- the main virtual method **run**() (p. 20) executes the following methods: **initialize**() (p. 20), **execute**() (p. 19) and **finalize**() (p. 20).
- the methods **initialize**() (p. 20), **execute**() (p. 19), and **finalize**() (p. 20) are pure virtual and have to be implemented by the derived classes.
- the method command(name) allows to execute the following commands: "run", "initialize","execute","finalize", each command executes the method of the same name. This is useful with composites, in particular it is used by the **serverVI** (p. 152) template class.

Definition at line 25 of file algorithmVI.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 algorithmVI::algorithmVI () `[inline]`

default constructor.

Definition at line 31 of file algorithmVI.h.

4.5.2.2 algorithmVI::~algorithmVI () `[inline]`

default destructor.

Definition at line 33 of file algorithmVI.h.

4.5.3 Member Function Documentation

4.5.3.1 void algorithmVI::command (std::string *com*) `[inline, virtual]`

virtual method to execute a command.

The commands are the methods "run", "initialize", "execute", "finalize"

Reimplemented in **eventVI** (p. 73).

Definition at line 13 of file algorithmVI.cpp.

4.5.3.2 void algorithmVI::execute () `[inline, pure virtual]`

pure virtual method to execute the algorithm.

Reimplemented in **algorithmComposite** (p. 11), **algorithmTree** (p. 16), **comApplyOptions** (p. 27), **comCloseIO** (p. 28), **comListOptions** (p. 29), **comLoadCalibration** (p. 30), **comOpenIO** (p. 31), **comReadEvent** (p. 32), **comSkipEvent** (p. 33), **comWriteEvent** (p. 34), **comWriteOutData** (p. 35), **comWriteOutDetector** (p. 36), **comWriteOutInfo** (p. 37), **comWriteWelcome** (p. 38), **fillAlg** (p. 76), **loadAlg** (p. 91), **runEventAlg** (p. 141), **runRunAlg** (p. 144), and **saveAlg** (p. 146).

4.5.3.3 void algorithmVI::finalize () `[inline, pure virtual]`

pure virtual method to finalize the algorithm.

Reimplemented in **algorithmCommand** (p. 9), **algorithmComposite** (p. 11), and **algorithmTree** (p. 17).

4.5.3.4 void algorithmVI::initialize () `[inline, pure virtual]`

pure virtual method initialize the algorithm.

Reimplemented in **algorithmCommand** (p. 9), **algorithmComposite** (p. 11), and **algorithmTree** (p. 17).

4.5.3.5 void algorithmVI::run () `[inline, virtual]`

virtual metho to run the algorithm.

executes the methods: **initialize**() (p. 20); **execute**() (p. 19); **finalize**() (p. 20);

Reimplemented in **algorithmComposite** (p. 11), and **eventVI** (p. 73).

Definition at line 5 of file algorithmVI.cpp.

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

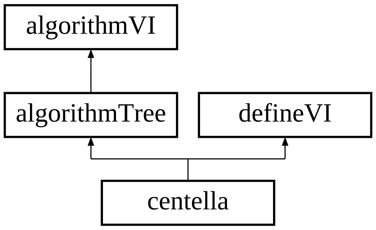
- **algorithmVI.h**
- **algorithmVI.cpp**

4.6 centella Class Reference

Application class.

`#include <centella.h>`

Inheritance diagram for centella:



Public Methods

- **centella** ()

  constructor - initalize managers.

- **~centella** ()

  destructors - delete managers.

- virtual void **define** ()

  define the class.

Static Public Methods

- centella∗ **instance** ()

  centella is a Singleton (to be use by centellaGUI). Access to the pointer.

Private Attributes

- **messageManager∗ m_messageManager**

  application independent **messageManager** (p. 93).

- **optionManager∗ m_optionManager**

  application independent **optionManager** (p. 109).

- **dataManager∗ m_dataManager**

  application dependent **dataManager** (p. 61).

- **processManager∗ m_processManager**

  application dependent **processManager** (p. 119).

- **selectionManager∗ m_selectionManager**

  application dependent **selectionManager** (p. 147).

Static Private Attributes

- centella∗ **m_instance** = 0

  singleton: pointer to the class.

4.6.1 Detailed Description

Application class.

Centella creates the **system** managers: **dataManager** (p. 61), **processManager** (p. 119), **selectionManager** (p. 147), **optionManager** (p. 109) and **messageManager** (p. 93).

Centella executes the program via its **algorithmTree** (p. 15) base class.

Centella is statically implemented in the code of its **define()** (p. 23) method.

- The system has five managers. Three should have a *Centella* specific implementation: **dataManager** (p. 61), **processManager** (p. 119), **selectionManager** (p. 147). Which declares the **data** (persistent and transient) the **processes** (algorithms and multialgorithms) and **selections** (cuts and multicuts). These managers declare and use the data, algorithms and selections relevant for the *centella* application. The virtual base classes for data, algorithms and selections are **trsDataVI** (p. 159), **algorithmVI** (p. 18) and **cutVI** (p. 49).
  - The data is clasified into detector data -geometry and calibration- (dataUserServerDet); event data - data relevant for every event - (dataUserServerEvent) and persistend data (dataUserServerROOT). In *Centella* the persisten data is stored into ROOT trees, histograms and ntuples. The **dataManager** (p. 61) serves the transient data via the `getData()` template method. In *Centella* calibration and event data are declared transient. The other data, persistent and geometry is for the moment provided via pointers. Most of the data is created one, it reference pointer stored into the server and its contents update.
  - The processes are algorithms. They are some system process and some user defined ones. The algorithms can be simple (**algorithmVI** (p. 18), **algorithmCommand** (p. 7)) or multialgorithms (**algorithmComposite** (p. 10), **algorithmTree** (p. 15)). Their main method is **run()** (p. 20). The **processManager** (p. 119) serves the algorithms vita the `getAlgorithm()` method. In the **processManager** (p. 119) there are servers to **algorithmTask** (p. 12) and **eventProcess** (p. 68) which are algorithms that can be defined dinamically, in particular, they can be created by the user via the input file and the **optionManager** (p. 109) `setOption()` method. The algorithms are created at one and not removed after stored into the server.
  - The selections are cuts. They are some system cuts and some defined by the user. They are simple cuts (**cutVI** (p. 49)) or multicuts (**cutComposite** (p. 44)). Their main method is `apply()`. The **selectionManager** (p. 147) serves the selection via the `getCut()` method. **selectionManager** (p. 147) has also a server for **cutSelection** (p. 46), a **cutComposite** (p. 44) that can be defined by the user. Again, the cuts should be construct an included in the server and not deleted.

  There are two independent managers: **optionManager** (p. 109) and **messageManager** (p. 93).

  - The **optionManager** (p. 109) serves to set options (that can be turn into operations) into servived classes derived from the base class **optionVI** (p. 114). Previously, paramaters of the classes, (types accepted: strings, double, int) should be turn into options using the method `defineOption()` declared in **optionVI** (p. 114). In this way it is possible to create objects dinamically, in particular algorithms and selections by the

  user. In *Centella* algorithms and selections are dinamically implemented via the input file.
  - The **messageManager** (p. 93) defines an ostream where to send the messages and provides some methods (i.e `message()`) to send them. The **messageManager** (p. 93) and the classes serviced by **messageVI** (p. 99) has a message level that determines if a message should or not realized.

- Centella is an **algorithmTree** (p. 15) that runs: 1) the execution of some initial algorithms 2) the execution of a run, which loop in a collection of events to apply some algorithms, 3) the execution of some final algorithms.

  - Some of the algorithms executed in the initial and finalize part are already declared in **define()** (p. 23), except two of them. They are the **algorithmTask** (p. 12) named "*iniOfRun*" and "*endOfRun*" where the **user** can define the algorithms to execute at the start and the end of the run.
  - The **execute**() (p. 16) method of *centella* run the **runRunAlg** (p. 143) algorithm, a system algorithm that performs the loop on the number of event of the persistent storage and call the algorithm **runEventAlg** (p. 140) to process every event.
  - The **runEventAlg** (p. 140) has an **eventProcess** (p. 68) (algorithm) called "*run-Event*" (a **eventProcess** (p. 68)) which the user can indicate which algorithms to run dinamically to. In addition **runEventAlg** (p. 140) has two **cutSelection** (p. 46) named "selRead" and "selAna" which decide if the event shoulb be processed and if the processed event should be intalled in persistent data. This two **cutSelection** (p. 46) can be dinamically defined by the user.

Definition at line 101 of file centella.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 centella::centella ()

constructor - initalize managers.

Definition at line 21 of file centella.cpp.

4.6.2.2 centella::∼centella ()

destructors - delete managers.

the **optionManager** (p. 109) should be created first

Definition at line 40 of file centella.cpp.

4.6.3 Member Function Documentation

4.6.3.1 void centella::define () `[inline, virtual]`

define the class.

Reimplemented from **defineVI** (p. 65).

Definition at line 51 of file centella.cpp.

4.6.3.2 centella ∗ centella::instance () `[inline, static]`

centella is a Singleton (to be use by centellaGUI). Access to the pointer.

Definition at line 117 of file centella.h.

4.6.4 Member Data Documentation

4.6.4.1 dataManager ∗ centella::m_dataManager `[private]`

application dependent **dataManager** (p. 61).

Definition at line 130 of file centella.h.

4.6.4.2 centella ∗ centella::m_instance = 0 `[static, private]`

singleton: pointer to the class.

Definition at line 122 of file centella.h.

4.6.4.3 messageManager ∗ centella::m_messageManager `[private]`

application independent **messageManager** (p. 93).

Definition at line 125 of file centella.h.

4.6.4.4 optionManager ∗ centella::m_optionManager `[private]`

application independent **optionManager** (p. 109).

Definition at line 127 of file centella.h.

4.6.4.5 processManager ∗ centella::m_processManager `[private]`

application dependent **processManager** (p. 119).

Definition at line 132 of file centella.h.

4.6.4.6 selectionManager ∗ centella::m_selectionManager `[private]`

application dependent **selectionManager** (p. 147).

Definition at line 134 of file centella.h.

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

- **centella.h**
- **centella.cpp**

4.7 centellaAlgorithms Class Reference

Server with the **system** and **user** declared algorithms.

`#include <centellaAlgorithms.h>`

Public Methods

- **centellaAlgorithms** ()
  constructor - creates the algorithms.
- **~centellaAlgorithms** ()
  destructor - destroies the algorithms.

4.7.1 Detailed Description

Server with the **system** and **user** declared algorithms.

All the static algorithms served by the **processManager** (p. 119) should be included here.

There are two **system** and **user** static algorithms.

The algorithms should exits untill the end of the program. They are not implicetely destroyed.

Definition at line 11 of file centellaAlgorithms.h.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 centellaAlgorithms::centellaAlgorithms ()

constructor - creates the algorithms.

Definition at line 9 of file centellaAlgorithms.cpp.

4.7.2.2 centellaAlgorithms::~centellaAlgorithms () `[inline]`

destructor - destroies the algorithms.

Definition at line 19 of file centellaAlgorithms.h.

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

- **centellaAlgorithms.h**
- **centellaAlgorithms.cpp**

4.8 centellaCuts Class Reference

`#include <centellaCuts.h>`

Public Methods

- **centellaCuts** ()

  constructor - constructs the User cuts.

- **~centellaCuts** ()

  destructor - destroies the User cuts.

4.8.1 Constructor & Destructor Documentation

4.8.1.1 centellaCuts::centellaCuts ()

constructor - constructs the User cuts.

the cuts derived from **cutVI** (p. 49) should be declared/construct here to make them available to the rest of the program via de **selectionManager** (p. 147)

Definition at line 8 of file centellaCuts.cpp.

4.8.1.2 centellaCuts::~centellaCuts () `[inline]`

destructor - destroies the User cuts.

Definition at line 20 of file centellaCuts.h.

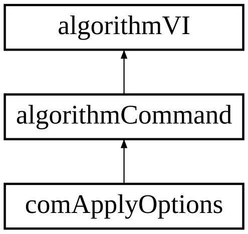
The documentation for this class was generated from the following files:

- **centellaCuts.h**
- **centellaCuts.cpp**

4.9 comApplyOptions Class Reference

`#include <centellaCommands.h>`

Inheritance diagram for comApplyOptions:



Public Methods

- virtual void **execute** ()
  algorithmCommand (p. 7) *to apply options to the classes.*

4.9.1 Member Function Documentation

4.9.1.1 void comApplyOptions::execute () `[inline, virtual]`

algorithmCommand (p. 7) to apply options to the classes.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 67 of file centellaCommands.cpp.

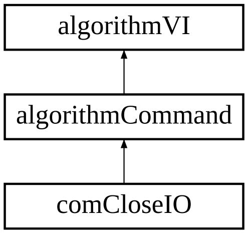
The documentation for this class was generated from the following files:

- **centellaCommands.h**
- **centellaCommands.cpp**

4.10 comCloseIO Class Reference

`#include <centellaCommands.h>`

Inheritance diagram for comCloseIO:



Public Methods

- virtual void **execute** ()
  algorithmCommand (p. 7) *to close the IO files.*

4.10.1 Member Function Documentation

4.10.1.1 void comCloseIO::execute () `[inline, virtual]`

algorithmCommand (p. 7) to close the IO files.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 22 of file centellaCommands.cpp.

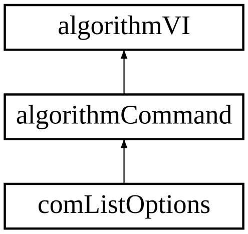
The documentation for this class was generated from the following files:

- **centellaCommands.h**
- **centellaCommands.cpp**

4.11 comListOptions Class Reference

`#include <centellaCommands.h>`

Inheritance diagram for comListOptions:



Public Methods

- virtual void **execute** ()
  algorithmCommand (p. 7) *to list options of the classes.*

4.11.1 Member Function Documentation

4.11.1.1 void comListOptions::execute () `[inline, virtual]`

algorithmCommand (p. 7) to list options of the classes.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 73 of file centellaCommands.cpp.

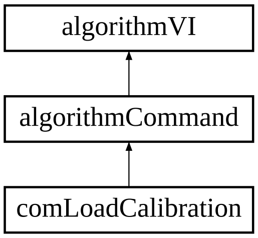
The documentation for this class was generated from the following files:

- **centellaCommands.h**
- **centellaCommands.cpp**

4.12 comLoadCalibration Class Reference

`#include <centellaCommands.h>`

Inheritance diagram for comLoadCalibration:



Public Methods

- virtual void **execute** ()
  algorithmCommand (p. 7) *class to load the detector configuration.*

4.12.1 Member Function Documentation

4.12.1.1 void comLoadCalibration::execute () `[inline, virtual]`

algorithmCommand (p. 7) class to load the detector configuration.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 10 of file centellaCommands.cpp.

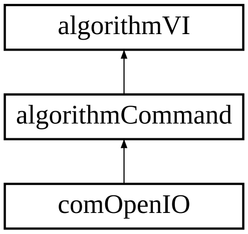
The documentation for this class was generated from the following files:

- **centellaCommands.h**
- **centellaCommands.cpp**

4.13 comOpenIO Class Reference

`#include <centellaCommands.h>`

Inheritance diagram for comOpenIO:



Public Methods

- virtual void **execute** ()
  algorithmCommand (p. 7) *class to open the IO files.*

4.13.1 Member Function Documentation

4.13.1.1 void comOpenIO::execute () `[inline, virtual]`

algorithmCommand (p. 7) class to open the IO files.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 16 of file centellaCommands.cpp.

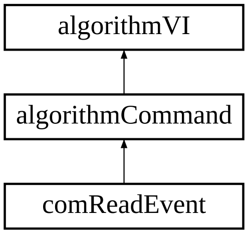
The documentation for this class was generated from the following files:

- **centellaCommands.h**
- **centellaCommands.cpp**

4.14 comReadEvent Class Reference

`#include <centellaCommands.h>`

Inheritance diagram for comReadEvent:



Public Methods

- virtual void **execute** ()

  algorithmCommand (p. 7) *to load the event.*

4.14.1 Member Function Documentation

4.14.1.1 void comReadEvent::execute () `[inline, virtual]`

algorithmCommand (p. 7) to load the event.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 28 of file centellaCommands.cpp.

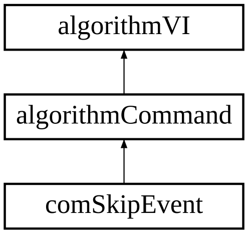
The documentation for this class was generated from the following files:

- **centellaCommands.h**
- **centellaCommands.cpp**

4.15 comSkipEvent Class Reference

`#include <centellaCommands.h>`

Inheritance diagram for comSkipEvent:



Public Methods

- virtual void **execute** ()
  algorithmCommand (p. 7) *to load the event.*

4.15.1 Member Function Documentation

4.15.1.1 void comSkipEvent::execute () `[inline, virtual]`

algorithmCommand (p. 7) to load the event.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 35 of file centellaCommands.cpp.

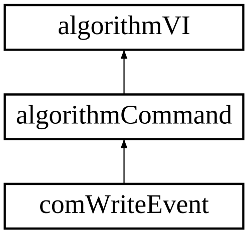
The documentation for this class was generated from the following files:

- **centellaCommands.h**
- **centellaCommands.cpp**

4.16 comWriteEvent Class Reference

`#include <centellaCommands.h>`

Inheritance diagram for comWriteEvent:



Public Methods

- virtual void **execute** ()
  algorithmCommand (p. 7) *to load the event.*

4.16.1 Member Function Documentation

4.16.1.1 void comWriteEvent::execute () `[inline, virtual]`

algorithmCommand (p. 7) to load the event.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 41 of file centellaCommands.cpp.

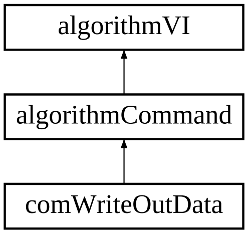
The documentation for this class was generated from the following files:

- **centellaCommands.h**
- **centellaCommands.cpp**

4.17 comWriteOutData Class Reference

`#include <centellaCommands.h>`

Inheritance diagram for comWriteOutData:



Public Methods

- virtual void **execute** ()
  algorithmCommand (p. 7) *to load the event.*

4.17.1 Member Function Documentation

4.17.1.1 void comWriteOutData::execute () `[inline, virtual]`

algorithmCommand (p. 7) to load the event.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 47 of file centellaCommands.cpp.

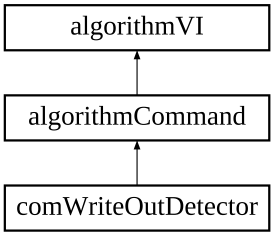
The documentation for this class was generated from the following files:

- **centellaCommands.h**
- **centellaCommands.cpp**

4.18 comWriteOutDetector Class Reference

`#include <centellaCommands.h>`

Inheritance diagram for comWriteOutDetector:



Public Methods

- virtual void **execute** ()
  algorithmCommand (p. 7) *to load the event.*

4.18.1 Member Function Documentation

4.18.1.1 void comWriteOutDetector::execute () `[inline, virtual]`

algorithmCommand (p. 7) to load the event.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 53 of file centellaCommands.cpp.

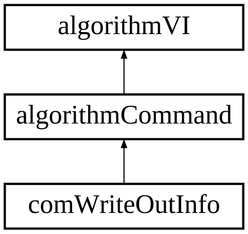
The documentation for this class was generated from the following files:

- **centellaCommands.h**
- **centellaCommands.cpp**

4.19 comWriteOutInfo Class Reference

`#include <centellaCommands.h>`

Inheritance diagram for comWriteOutInfo:



Public Methods

- virtual void **execute** ()

  algorithmCommand (p. 7) *to load the event.*

4.19.1 Member Function Documentation

4.19.1.1 void comWriteOutInfo::execute () `[inline, virtual]`

algorithmCommand (p. 7) to load the event.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 59 of file centellaCommands.cpp.

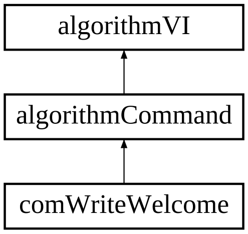
The documentation for this class was generated from the following files:

- **centellaCommands.h**
- **centellaCommands.cpp**

4.20 comWriteWelcome Class Reference

`#include <centellaCommands.h>`

Inheritance diagram for comWriteWelcome:



Public Methods

- virtual void **execute** ()
  algorithmCommand (p. 7) *to list options of the classes.*

4.20.1 Member Function Documentation

4.20.1.1 void comWriteWelcome::execute () `[inline, virtual]`

algorithmCommand (p. 7) to list options of the classes.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 78 of file centellaCommands.cpp.

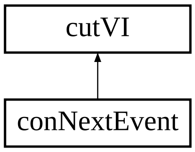
The documentation for this class was generated from the following files:

- **centellaCommands.h**
- **centellaCommands.cpp**

4.21 conNextEvent Class Reference

`#include <centellaCommands.h>`

Inheritance diagram for conNextEvent:



Public Methods

- virtual bool **apply** ()
  cutVI (p. 49) *that return tree is there is a next event in the rootTree.*

4.21.1 Member Function Documentation

4.21.1.1 bool conNextEvent::apply () `[inline, virtual]`

cutVI (p. 49) that return tree is there is a next event in the rootTree.

Reimplemented from **cutVI** (p. 50).

Definition at line 91 of file centellaCommands.cpp.

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

- **centellaCommands.h**
- **centellaCommands.cpp**

4.22 converterServer Class Reference

virtual server of converters.

`#include <converterServer.h>`

Protected Methods

- **converterServer** ()

  constructor - call to define the converters, makes the algorithms.

- ~**converterServer** ()

  default destructor.

- void **load** (std::string name)

  load (Persistent->Transient) a named class.

- void **save** (std::string name)

  save (transiente->persistent) a named class.

- **converterVI**∗ **getConverter** (std::string name)

  returns the converter of a given name.

- virtual void **defineConverters** () = 0

  pure virtual function to define the coverters to the server.

Protected Attributes

- **serverVI<converterVI>**∗ **m_server**

  server with the list of conveters.

Private Methods

- void **makeAlgorithms** ()

  make algorithms from the converters.

4.22.1 Detailed Description

virtual server of converters.

- The **load()** (p. 41) and **save()** (p. 41) method allows to tranform from persistent to Transient data or viceversa a given class.
- The name of the converters should be the same as the converter class.
- The converters are added into algorithms.

Definition at line 21 of file converterServer.h.

4.22.2 Constructor & Destructor Documentation

4.22.2.1 converterServer::converterServer () `[protected]`

constructor - call to define the converters, makes the algorithms.

Definition at line 7 of file converterServer.cpp.

4.22.2.2 converterServer::~converterServer () `[inline, protected]`

default destructor.

Definition at line 29 of file converterServer.h.

4.22.3 Member Function Documentation

4.22.3.1 void converterServer::defineConverters () `[inline, protected, pure virtual]`

pure virtual function to define the coverters to the server.

4.22.3.2 converterVI * converterServer::getConverter (std::string *name*) `[protected]`

returns the converter of a given name.

Definition at line 16 of file converterServer.cpp.

4.22.3.3 void converterServer::load (std::string *name*) `[inline, protected]`

load (Persistent->Transient) a named class.

Definition at line 32 of file converterServer.h.

4.22.3.4 void converterServer::makeAlgorithms () `[private]`

make algorithms from the converters.

Definition at line 25 of file converterServer.cpp.

4.22.3.5 void converterServer::save (std::string *name*) `[inline, protected]`

save (transiente->persistent) a named class.

Definition at line 34 of file converterServer.h.

4.22.4 Member Data Documentation

4.22.4.1 serverVI<converterVI>* converterServer::m_server `[protected]`

server with the list of conveters.

Definition at line 50 of file converterServer.h.

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

- **converterServer.h**
- **converterServer.cpp**

4.23 converterVI Class Reference

virtual base class for converter.

`#include <converterVI.h>`

Public Methods

- virtual void **load** () = 0
  virtual base method to load from permanent to transient.
- virtual void **save** () = 0
  virtual base method to save from transient to permanent.

4.23.1 Detailed Description

virtual base class for converter.

Definition at line 6 of file converterVI.h.

4.23.2 Member Function Documentation

4.23.2.1 void converterVI::load () `[inline, pure virtual]`

virtual base method to load from permanent to transient.

4.23.2.2 void converterVI::save () `[inline, pure virtual]`

virtual base method to save from transient to permanent.

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

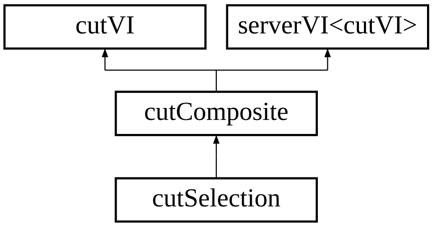
- **converterVI.h**

4.24 cutComposite Class Reference

A composite of cuts.

`#include <cutComposite.h>`

Inheritance diagram for cutComposite:



Public Methods

- **cutComposite** ()
  Default constructor.
- ~**cutComposite** ()
  default destructor.
- virtual bool **apply** ()
  apply all the cuts in the selection.

4.24.1 Detailed Description

A composite of cuts.

- It is a cut. Its **apply()** (p. 45) method executes and AND of all cuts in the composite.
- The cuts are served via a **serverVI** (p. 152). That means that individual cuts can be retreived via names.

Definition at line 16 of file cutComposite.h.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 cutComposite::cutComposite () `[inline]`

Default constructor.

Definition at line 22 of file cutComposite.h.

4.24.2.2 cutComposite::~cutComposite () `[inline]`

default destructor.

Definition at line 24 of file cutComposite.h.

4.24.3 Member Function Documentation

4.24.3.1 bool cutComposite::apply () `[inline, virtual]`

apply all the cuts in the selection.

Reimplemented from **cutVI** (p. 50).

Definition at line 5 of file cutComposite.cpp.

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

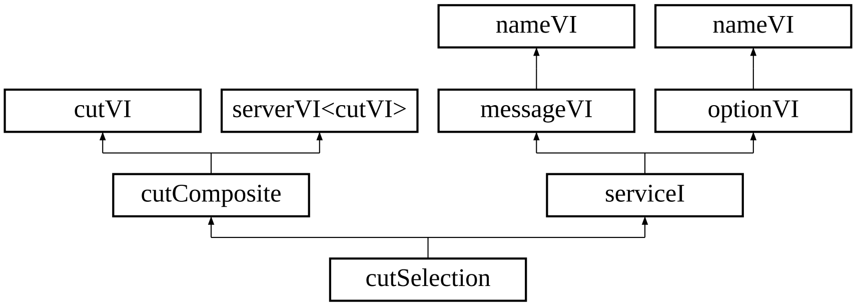
- **cutComposite.h**
- **cutComposite.cpp**

4.25 cutSelection Class Reference

A cutSelection is a **cutComposite** (p. 44) (a collection of cuts) with **optionVI** (p. 114) and **messageVI** (p. 99) services.

`#include <cutSelection.h>`

Inheritance diagram for cutSelection:



Public Methods

- **cutSelection** (std::string name)

  constructor - set a name and recive the services declared in **serviceI** (p. 156)*.*

- ~**cutSelection** ()

  destructor.

- virtual void **writeOut** () const

  write Out the information of this class.

Protected Methods

- virtual void **defineOption** ()

  define the options.

- virtual void **setOption** (std::string addCut, std::string cutName)

  operation : add cut into composite.

Private Attributes

- std::string **m_cutName**

  dummy - last cut in composite, for **optionVI** (p. 114)*.*

4.25.1 Detailed Description

A cutSelection is a **cutComposite** (p. 44) (a collection of cuts) with **optionVI** (p. 114) and **messageVI** (p. 99) services.

- is a **cutComposite** (p. 44).
- is a serviced class by the base class **serviceI** (p. 156). To be included in ther servers every object of cutSelection should have a unique name.
- A user can add cuts into the **cutComposite** (p. 44) via the **setOption()** (p. 47) methods of **optionVI** (p. 114).

Definition at line 16 of file cutSelection.h.

4.25.2 Constructor & Destructor Documentation

4.25.2.1 cutSelection::cutSelection (std::string *name*) `[inline]`

constructor - set a name and recive the services declared in **serviceI** (p. 156).

Definition at line 23 of file cutSelection.h.

4.25.2.2 cutSelection::~cutSelection ()

destructor.

4.25.3 Member Function Documentation

4.25.3.1 void cutSelection::defineOption () `[inline, protected, virtual]`

define the options.

Reimplemented from **optionVI** (p. 116).

Definition at line 6 of file cutSelection.cpp.

4.25.3.2 void cutSelection::setOption (std::string *addCut*, std::string *cutName*) `[inline, protected, virtual]`

operation : add cut into composite.

Reimplemented from **optionVI** (p. 117).

Definition at line 12 of file cutSelection.cpp.

4.25.3.3 void cutSelection::writeOut () const `[inline, virtual]`

write Out the information of this class.

Reimplemented from **serviceI** (p. 158).

Definition at line 24 of file cutSelection.cpp.

4.25.4 Member Data Documentation

4.25.4.1 std::string cutSelection::m_cutName `[private]`

dummy - last cut in composite, for **optionVI** (p. 114).

Definition at line 40 of file cutSelection.h.

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

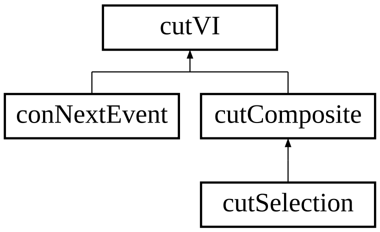
- **cutSelection.h**
- **cutSelection.cpp**

4.26 cutVI Class Reference

pure virtual class to define cut.

`#include <cutVI.h>`

Inheritance diagram for cutVI:



Public Methods

- **cutVI** ()

  default constructor.

- **~cutVI** ()

  defulat constructor.

- virtual bool **apply** () = 0

  pure virtual function to apply a given cut.

4.26.1 Detailed Description

pure virtual class to define cut.

All selections of the program should inherit from this class.

It has the pure virtual method **apply**() (p. 50)

Definition at line 9 of file cutVI.h.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 cutVI::cutVI () `[inline]`

default constructor.

Definition at line 15 of file cutVI.h.

4.26.2.2 cutVI::~cutVI () `[inline]`

defulat constructor.

Definition at line 17 of file cutVI.h.

4.26.3 Member Function Documentation

4.26.3.1 bool cutVI::apply () `[inline, pure virtual]`

pure virtual function to apply a given cut.

Reimplemented in **conNextEvent** (p. 39), and **cutComposite** (p. 45).

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

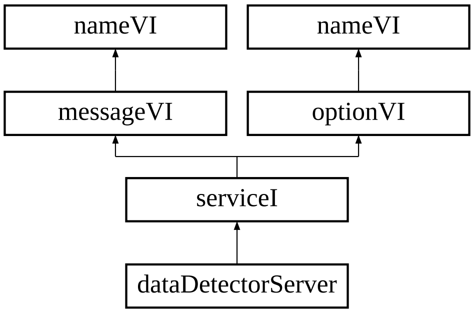
- **cutVI.h**

4.27 dataDetectorServer Class Reference

Handles the Detector data (geometry and calibration data).

`#include <dataDetectorServer.h>`

Inheritance diagram for dataDetectorServer:



Public Methods

- **dataDetectorServer** ()

  construct the data.

- **~dataDetectorServer** ()

  delete the data.

- void **loadCalibration** ()

  loads the calibration data.

- virtual void **writeOut** () const

  write out the information.

Protected Methods

- **serverVI<trsDataVI>∗ server** () const

  returns the server of the **trsDataVI** (p. 159) *for geometry and calibration data.*

- **trsDataVI∗ getTrsData** (std::string name) const

  returns the **trsDataVI** (p. 159) *with the name for geometry and calibration data.*

Private Attributes

- **serverVI<trsDataVI>∗ m_server**

  pointer to the server of the **trsDataVI** (p. 159) *of the transient data for the detector.*

Friends

- class **dataManager**
- class **comLoadDetector**

4.27.1 Detailed Description

Handles the Detector data (geometry and calibration data).

- It has accesible pointers to the detector geometry data. It includes the particular implementation of the geometry data of every detector.
- It has a protected server with the detector calibration data. The calibration data is declared transiend data (inherits from **trsDataVI** (p. 159)). The calibration data is accesible from **dataManager** (p. 61) instead of from this class, via the method `getData()`.
- the method **loadCalibration()** (p. 52) loads the calibration data.

Definition at line 28 of file dataDetectorServer.h.

4.27.2 Constructor & Destructor Documentation

4.27.2.1 dataDetectorServer::dataDetectorServer ()

construct the data.

Definition at line 5 of file dataDetectorServer.cpp.

4.27.2.2 dataDetectorServer::~dataDetectorServer ()

delete the data.

Definition at line 35 of file dataDetectorServer.cpp.

4.27.3 Member Function Documentation

4.27.3.1 trsDataVI * dataDetectorServer::getTrsData (std::string *name*) const `[protected]`

returns the **trsDataVI** (p. 159) with the name for geometry and calibration data.

Definition at line 21 of file dataDetectorServer.cpp.

4.27.3.2 void dataDetectorServer::loadCalibration ()

loads the calibration data.

the user should indicate his/her detector transiente data

Definition at line 16 of file dataDetectorServer.cpp.

4.27.3.3 serverVI<trsDataVI>* dataDetectorServer::server () const `[inline, protected]`

returns the server of the **trsDataVI** (p. 159) for geometry and calibration data.

Definition at line 53 of file dataDetectorServer.h.

4.27.3.4 void dataDetectorServer::writeOut () const `[inline, virtual]`

write out the information.

Reimplemented from **serviceI** (p. 158).

Definition at line 29 of file dataDetectorServer.cpp.

4.27.4 Friends And Related Function Documentation

4.27.4.1 class comLoadDetector `[friend]`

command to load the detector geometry and calibration data

Definition at line 34 of file dataDetectorServer.h.

4.27.4.2 class dataManager `[friend]`

To acces to **trsDataVI** (p. 159) of the server

Definition at line 32 of file dataDetectorServer.h.

4.27.5 Member Data Documentation

4.27.5.1 serverVI<trsDataVI>* dataDetectorServer::m_server `[private]`

pointer to the server of the **trsDataVI** (p. 159) of the transient data for the detector.

Definition at line 61 of file dataDetectorServer.h.

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

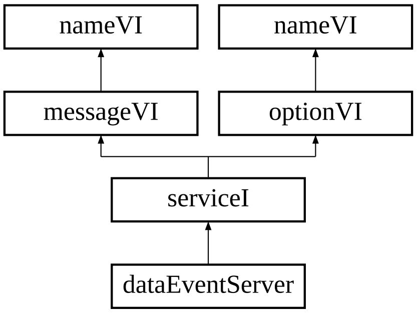
- **dataDetectorServer.h**
- **dataDetectorServer.cpp**

4.28 dataEventServer Class Reference

Server with the transiend data of the Event.

`#include <dataEventServer.h>`

Inheritance diagram for dataEventServer:



Public Methods

- **dataEventServer** ()

  default constructor - it creates all the event data of the run.

- ~**dataEventServer** ()

  default constructor.

- virtual void **writeOut** () const

  write out the information of the data.

Protected Methods

- **serverVI<trsDataVI>* server** ()

  return **trsDataVI** (p. 159)*.*

- **trsDataVI* getTrsData** (std::string name) const

  returns the **trsDataVI** (p. 159) *associated with this name.*

Private Attributes

- **serverVI<trsDataVI>* m_server**

  pointer to the server with all the **trsDataVI** (p. 159) *of type event.*

Friends

- class **dataManager**

4.28.1 Detailed Description

Server with the transiend data of the Event.

- It has a protected server (**serverVI** (p. 152)) of the event transiend data (**trsDataVI** (p. 159)). wtih restricted to the **dataManager** (p. 61).
- The user shoul declare all the events data in the constructor.

Definition at line 24 of file dataEventServer.h.

4.28.2 Constructor & Destructor Documentation

4.28.2.1 dataEventServer::dataEventServer ()

default constructor - it creates all the event data of the run.

Definition at line 7 of file dataEventServer.cpp.

4.28.2.2 dataEventServer::~dataEventServer ()

default constructor.

Definition at line 28 of file dataEventServer.cpp.

4.28.3 Member Function Documentation

4.28.3.1 trsDataVI * dataEventServer::getTrsData (std::string *name*) const `[protected]`

returns the **trsDataVI** (p. 159) associated with this name.

The user should define the event transient data

Definition at line 18 of file dataEventServer.cpp.

4.28.3.2 serverVI<trsDataVI>* dataEventServer::server () `[inline, protected]`

return **trsDataVI** (p. 159).

Definition at line 43 of file dataEventServer.h.

4.28.3.3 void dataEventServer::writeOut () const `[inline, virtual]`

write out the information of the data.

Reimplemented from **serviceI** (p. 158).

Definition at line 38 of file dataEventServer.h.

4.28.4 Friends And Related Function Documentation

4.28.4.1 class dataManager `[friend]`

dataManager (p. 61) friend to acces the server of transient data

Definition at line 28 of file dataEventServer.h.

4.28.5 Member Data Documentation

4.28.5.1 serverVI<trsDataVI>* dataEventServer::m_server `[private]`

pointer to the server with all the **trsDataVI** (p. 159) of type event.

Definition at line 51 of file dataEventServer.h.

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

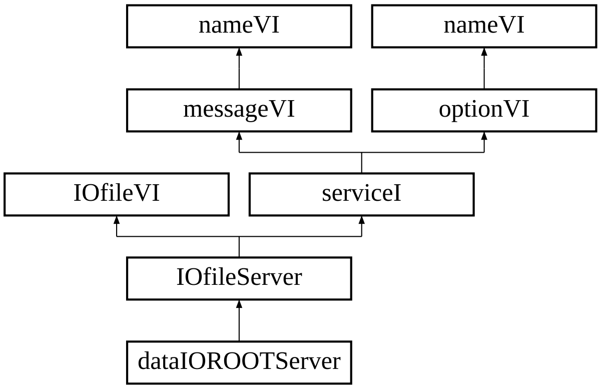
- **dataEventServer.h**
- **dataEventServer.cpp**

4.29 dataIOROOTServer Class Reference

It stores the ROOT trees Servers and histograms.

`#include <dataIOROOTServer.h>`

Inheritance diagram for dataIOROOTServer:



Public Methods

- **dataIOROOTServer** ()

  constructor.

- **~dataIOROOTServer** ()

  destructor.

Protected Methods

- **rTreeInServer* treeIn** () const

  returns a pointer to the Input root-tree Server.

- **rTreeOutServer* treeOut** () const

  returns a pointer to the Output root-tree Server.

- **rHistoOutServer* histo** () const

  returns a pointer to the histo Server.

- virtual void **defineIOfiles** ()

  define the input and output **IOfileVI** (p. 84)*.*

Private Attributes

- **rTreeInServer* m_TreeIn**

  pointer to the input root-tree Server.

- **rTreeOutServer∗ m_TreeOut**

  pointer to the output root-tree Server.

- **rHistoOutServer∗ m_Histo**

  pointer to the histo Server.

Friends

- class **dataManager**

4.29.1 Detailed Description

It stores the ROOT trees Servers and histograms.

dataIOROOTServer is an implementation of permanent data Server for ROOT.<bt> *dataIOROOTServer* has the services of the IOServer *dataIOROOTServer* has defined: a rTreeInServer, rTreeOutServer, **rHistoServer** (p. 131)

- The **IOfileVI** (p. 84) operations are handle via the IOServer.
- The ROOT permanent data is the core (or main branch) of the input/output ROOT trees. Some of the converters or algorithms maybe need acces to this core. The core is provided via **treeIn**() (p. 59)->core(). And this should be accesible anywhere.
- The user should typedef (see header file) their implementation of the input and ouput ROOT trees.
- The User should typedef (see header file) their implementation of **rHistoServer** (p. 131).
- If not output files (either tree o histo) are going to be use the User should typedef to **nullIOfile** (p. 107) (a system defined class).

Notes:

- a rTreeServer is a user defined class that should inherits from:
  - the template class **rTreeBase** (p. 136), where the template argument is the core (main branch) of the ROOT tree.
  - from a **converterServer** (p. 40) that will convert the **converterVI** (p. 43) s into **loadAlg** (p. 90) and **saveAlg** (p. 145) algorithms.
- a **rHistoServer** (p. 131) is a user defined class that should inherits from
  - a base rFileBase class
  - a **rHistoFolderServer** (p. 128) class. That will convert the rHistoFolders into **fillAlg** (p. 75) algorithms/

Definition at line 51 of file dataIOROOTServer.h.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 dataIOROOTServer::dataIOROOTServer ()

constructor.

Definition at line 17 of file dataIOROOTServer.cxx.

4.29.2.2 dataIOROOTServer::~dataIOROOTServer ()

destructor.

Definition at line 36 of file dataIOROOTServer.cxx.

4.29.3 Member Function Documentation

4.29.3.1 void dataIOROOTServer::defineIOfiles () `[inline, protected, virtual]`

define the input and output **IOfileVI** (p. 84).

Reimplemented from **IOfileServer** (p. 80).

Definition at line 28 of file dataIOROOTServer.cxx.

4.29.3.2 rHistoOutServer * dataIOROOTServer::histo () const `[protected]`

returns a pointer to the histo Server.

Definition at line 14 of file dataIOROOTServer.cxx.

4.29.3.3 rTreeInServer * dataIOROOTServer::treeIn () const `[protected]`

returns a pointer to the Input root-tree Server.

Definition at line 12 of file dataIOROOTServer.cxx.

4.29.3.4 rTreeOutServer * dataIOROOTServer::treeOut () const `[protected]`

returns a pointer to the Output root-tree Server.

Definition at line 13 of file dataIOROOTServer.cxx.

4.29.4 Friends And Related Function Documentation

4.29.4.1 class dataManager `[friend]`

Definition at line 54 of file dataIOROOTServer.h.

4.29.5 Member Data Documentation

4.29.5.1 rHistoOutServer * dataIOROOTServer::m_Histo `[private]`

pointer to the histo Server.

Definition at line 82 of file dataIOROOTServer.h.

4.29.5.2 rTreeInServer * dataIOROOTServer::m_TreeIn `[private]`

pointer to the input root-tree Server.

Definition at line 78 of file dataIOROOTServer.h.

4.29.5.3 rTreeOutServer ∗ dataIOROOTServer::m_TreeOut `[private]`

pointer to the output root-tree Server.

Definition at line 80 of file dataIOROOTServer.h.

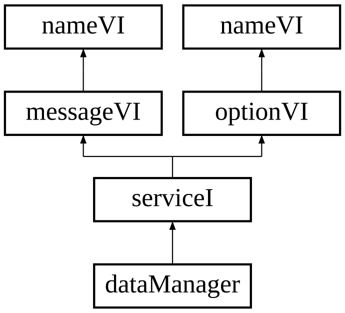
The documentation for this class was generated from the following files:

- **dataIOROOTServer.h**
- **dataIOROOTServer.cxx**

4.30 dataManager Class Reference

`#include <dataManager.h>`

Inheritance diagram for dataManager:



Public Methods

- **dataManager** ()

  constructor - construct the servers.

- ~**dataManager** ()

  destructor.

- template<class T> T∗ **getData** (std::string name, const T∗ dummy = 0) const

  returns the class derived from **trsDataVI** (p. 159)*.*

- **dataEventServer**∗ **evt** () const

  returns pointer to te **dataEventServer** (p. 54)*.*

- **dataDetectorServer**∗ **det** () const

  returns pointer to the **dataDetectorServer** (p. 51)*.*

- **dataIOServer**∗ **IO** () const

  returns pointer to the persistency data.

Static Public Methods

- dataManager∗ **instance** ()

  Singleton: pointer to itself.

Protected Methods

- **trsDataVI**∗ **getTrsData** (std::string name) const

  returns the trsData with that name in the servers.

Private Attributes

- **dataEventServer∗ m_evt**

  pointer to event data.

- **dataDetectorServer∗ m_det**

  pointer to detector data.

- **dataIOServer∗ m_IO**

  pointer to persistent data.

Static Private Attributes

- dataManager∗ **m_instance** = 0

  Singleton: pointer to itself.

4.30.1 Constructor & Destructor Documentation

4.30.1.1 dataManager::dataManager ()

constructor - construct the servers.

Definition at line 10 of file dataManager.cpp.

4.30.1.2 dataManager::∼dataManager ()

destructor.

Definition at line 22 of file dataManager.cpp.

4.30.2 Member Function Documentation

4.30.2.1 dataIOServer ∗ dataManager::IO () const `[inline]`

returns pointer to the persistency data.

Definition at line 60 of file dataManager.h.

4.30.2.2 dataDetectorServer ∗ dataManager::det () const `[inline]`

returns pointer to the **dataDetectorServer** (p. 51).

Definition at line 58 of file dataManager.h.

4.30.2.3 dataEventServer ∗ dataManager::evt () const `[inline]`

returns pointer to te **dataEventServer** (p. 54).

Definition at line 56 of file dataManager.h.

4.30.2.4 template<class T> T * dataManager::getData (std::string *name*, const T * *dummy* = 0) const `[inline]`

returns the class derived from **trsDataVI** (p. 159).

The pointer to a dummy class of the derived type is needed in order to identify the template. JAH errors with dinamic_cast.

Definition at line 50 of file dataManager.h.

4.30.2.5 trsDataVI * dataManager::getTrsData (std::string *name*) const `[protected]`

returns the trsData with that name in the servers.

Definition at line 30 of file dataManager.cpp.

4.30.2.6 dataManager * dataManager::instance () `[inline, static]`

Singleton: pointer to itself.

Definition at line 45 of file dataManager.h.

4.30.3 Member Data Documentation

4.30.3.1 dataIOServer * dataManager::m_IO `[private]`

pointer to persistent data.

Definition at line 77 of file dataManager.h.

4.30.3.2 dataDetectorServer * dataManager::m_det `[private]`

pointer to detector data.

Definition at line 75 of file dataManager.h.

4.30.3.3 dataEventServer * dataManager::m_evt `[private]`

pointer to event data.

Definition at line 73 of file dataManager.h.

4.30.3.4 dataManager * dataManager::m_instance = 0 `[static, private]`

Singleton: pointer to itself.

Definition at line 70 of file dataManager.h.

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

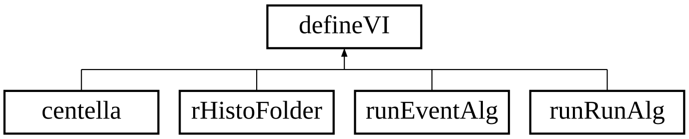
- **dataManager.h**
- **dataManager.cpp**

4.31 defineVI Class Reference

A bare class to allow classes to be defined after been construct.

`#include <defineVI.h>`

Inheritance diagram for defineVI:



Protected Methods

- **defineVI** ()

  constructor - set definition to false.

- ~**defineVI** ()

  destructor.

- virtual void **define** () = 0

  pure virtual class to define a class.

- void **setDefine** (bool t)

  set the definition.

- bool **defined** () const

  returns false if the class has not bee defined.

Private Attributes

- bool **m_define**

  keeps track if the clas has been run time defined or not.

4.31.1 Detailed Description

A bare class to allow classes to be defined after been construct.

As an example some algorithms needs other to be executed. But at construction time these algorithms are maybe not constructed yet. At execution time is the class has not bee defined (**defined()** (p. 65) method) then the method **define()** (p. 65) will be executed to define the object.

Definition at line 11 of file defineVI.h.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 defineVI::defineVI () `[inline, protected]`

constructor - set definition to false.

Definition at line 17 of file defineVI.h.

4.31.2.2 defineVI::~defineVI () `[inline, protected]`

destructor.

Definition at line 19 of file defineVI.h.

4.31.3 Member Function Documentation

4.31.3.1 void defineVI::define () `[inline, protected, pure virtual]`

pure virtual class to define a class.

Reimplemented in **centella** (p. 23), **rHistoFolder** (p. 126), **runEventAlg** (p. 141), and **runRunAlg** (p. 144).

4.31.3.2 bool defineVI::defined () const `[inline, protected]`

returns false if the class has not bee defined.

Definition at line 27 of file defineVI.h.

4.31.3.3 void defineVI::setDefine (bool *t*) `[inline, protected]`

set the definition.

Definition at line 25 of file defineVI.h.

4.31.4 Member Data Documentation

4.31.4.1 bool defineVI::m_define `[private]`

keeps track if the clas has been run time defined or not.

Definition at line 32 of file defineVI.h.

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

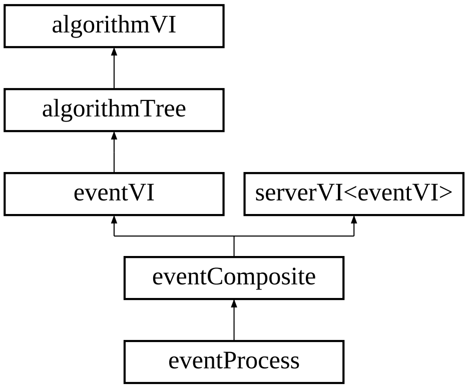
- **defineVI.h**

4.32 eventComposite Class Reference

A **eventVI** (p. 71) composite of eventVIs.

`#include <eventComposite.h>`

Inheritance diagram for eventComposite:



Public Methods

- **eventComposite** ()

  default constructor.

- ~**eventComposite** ()

  defautl destructor.

- virtual void **generate** ()

  the method generate, reconstruct, analyze are commands.

- virtual void **reconstruct** ()

  reconstruct the process running an **algorithmComposite** (p. 10)*.*

- virtual void **analyze** ()

  reconstruct the porcess running an **algorithmComposite** (p. 10)*.*

4.32.1 Detailed Description

A **eventVI** (p. 71) composite of eventVIs.

- Is an **eventVI** (p. 71)
- Is composed of several **eventVI** (p. 71).
- The **generate**() (p. 67), **reconstruct**() (p. 67), and **analyze**() (p. 67) methods, execute first the algorithms of this **eventVI** (p. 71) and later the same methods of the **eventVI** (p. 71) of the **serverVI** (p. 152).
- Note the the run is not overwriten. The run method executes in order the generate, reconstruct, analyze commands. It implies that the **`run()`** (p. 73) method executes first all the **`generate()`** (p. 67) methods, first itself, them the rest of the **eventVI** (p. 71) in the

composite; later executed the **`reconstruct()`** (p. 67), again first itself and later the others **eventVI** (p. 71); and finally the `analysis()`, in the same order, first itself, later the rest of **eventVI** (p. 71) in the composite.

Definition at line 23 of file eventComposite.h.

4.32.2 Constructor & Destructor Documentation

4.32.2.1 eventComposite::eventComposite () `[inline]`

default constructor.

Definition at line 30 of file eventComposite.h.

4.32.2.2 eventComposite::~eventComposite () `[inline]`

defautl destructor.

Definition at line 32 of file eventComposite.h.

4.32.3 Member Function Documentation

4.32.3.1 void eventComposite::analyze () `[inline, virtual]`

reconstruct the porcess running an **algorithmComposite** (p. 10).

Reimplemented from **eventVI** (p. 72).

Definition at line 40 of file eventComposite.h.

4.32.3.2 void eventComposite::generate () `[inline, virtual]`

the method generate, reconstruct, analyze are commands.

this methods execute first the algorithm composite of **eventVI** (p. 71), and later the method of all the **eventVI** (p. 71) algorithms in the **serverVI** (p. 152) via the command(name) method.

Reimplemented from **eventVI** (p. 73).

Definition at line 38 of file eventComposite.h.

4.32.3.3 void eventComposite::reconstruct () `[inline, virtual]`

reconstruct the process running an **algorithmComposite** (p. 10).

Reimplemented from **eventVI** (p. 73).

Definition at line 39 of file eventComposite.h.

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

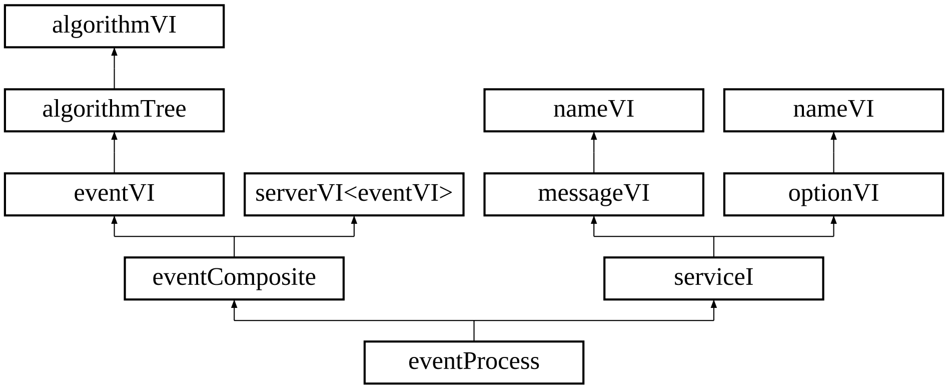
- **eventComposite.h**

4.33 eventProcess Class Reference

and eventProcess is a **eventComposite** (p. 66) with **optionVI** (p. 114) and **messageVI** (p. 99) services via the class **serviceI** (p. 156).

`#include <eventProcess.h>`

Inheritance diagram for eventProcess:



Public Methods

- **eventProcess** (std::string name)

  constructor - set the name.

- **~eventProcess** ()

  default destructor.

- virtual void **writeOut** () const

  information of this class.

Protected Methods

- virtual void **defineOption** ()

  define the options (operations) into **optionVI** (p. 114)*.*

- virtual void **setOption** (std::string task, std::string value)

  It performs the operations associated with the options.

Private Attributes

- std::string **m_genName**

  dummy strings for the option.

- std::string **m_recName**
- std::string **m_anaName**
- std::string **m_eventProcessName**

4.33.1 Detailed Description

and eventProcess is a **eventComposite** (p. 66) with **optionVI** (p. 114) and **messageVI** (p. 99) services via the class **serviceI** (p. 156).

- Is an **eventComposite** (p. 66).
- Is can use all the service provided by the base class **serviceI** (p. 156). To be included in the server every object of **eventComposite** (p. 66) should have a unique name (use **setName()** (p. 157) method of **serviceI** (p. 156)).
- Via the **setOption()** (p. 69) method of **optionVI** (p. 114), and external user (**optionManager** (p. 109)) can: 1) add an **eventComposite** (p. 66) to this eventProcess (also **eventComposite** (p. 66)), or 2) add algorithms to the **algorithmComposite** (p. 10) of the **eventVI** (p. 71) (main **eventVI** (p. 71) of the **eventComposite** (p. 66)) in the generation, reconstruction, and analysis taks of this **eventComposite** (p. 66).

Definition at line 24 of file eventProcess.h.

4.33.2 Constructor & Destructor Documentation

4.33.2.1 eventProcess::eventProcess (std::string *name*)

constructor - set the name.

Definition at line 6 of file eventProcess.cpp.

4.33.2.2 eventProcess::~eventProcess ()

default destructor.

4.33.3 Member Function Documentation

4.33.3.1 void eventProcess::defineOption () `[inline, protected, virtual]`

define the options (operations) into **optionVI** (p. 114).

add and **algorithmVI** (p. 18) into the **algorithmComposite** (p. 10) of the main **eventVI** (p. 71) of this **eventComposite** (p. 66)

Reimplemented from **optionVI** (p. 116).

Definition at line 40 of file eventProcess.cpp.

4.33.3.2 void eventProcess::setOption (std::string *task*, std::string *value*) `[inline, protected, virtual]`

It performs the operations associated with the options.

Reimplemented from **optionVI** (p. 117).

Definition at line 18 of file eventProcess.cpp.

4.33.3.3 void eventProcess::writeOut () const `[inline, virtual]`

information of this class.

Reimplemented from **serviceI** (p. 158).

Definition at line 52 of file eventProcess.cpp.

4.33.4 Member Data Documentation

4.33.4.1 std::string eventProcess::m_anaName `[private]`

Definition at line 50 of file eventProcess.h.

4.33.4.2 std::string eventProcess::m_eventProcessName `[private]`

Definition at line 52 of file eventProcess.h.

4.33.4.3 std::string eventProcess::m_genName `[private]`

dummy strings for the option.

The have the last algorithm name added in this object

Definition at line 48 of file eventProcess.h.

4.33.4.4 std::string eventProcess::m_recName `[private]`

Definition at line 49 of file eventProcess.h.

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

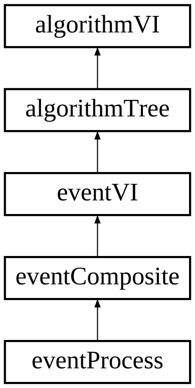
- **eventProcess.h**
- **eventProcess.cpp**

4.34 eventVI Class Reference

eventVI : bare class to define an event algorithm.

`#include <eventVI.h>`

Inheritance diagram for eventVI:



Public Methods

- **eventVI** ()

  constructor, construct the pointers to the algorithmComposites.

- **~eventVI** ()

  destructor, destroies the pointers to the algorithmComposites.

- virtual void **run** ()

  overwrite the virtuam method **run***()* (p. 73) *that now executes the* **generate***()* (p. 73)*,* **reconstruct***()* (p. 73)*, and analysis() methods;.*

- virtual void **generate** ()

  generate the process running an **algorithmComposite** (p. 10)*.*

- virtual void **reconstruct** ()

  reconstruct the process running an **algorithmComposite** (p. 10)*.*

- virtual void **analyze** ()

  reconstruct the porcess running an **algorithmComposite** (p. 10)*.*

- virtual void **command** (std::string com)

  execute the methods via the command.

Protected Methods

- **algorithmComposite**∗ **genAlg** () const

  pointer to generate **algorithmComposite** (p. 10)*.*

- **algorithmComposite∗ recAlg** () const

  pointer to the reconstruct **algorithmComposite** (p. 10)*.*

- **algorithmComposite∗ anaAlg** () const

  pointer to the analyze **algorithmComposite** (p. 10)*.*

4.34.1 Detailed Description

eventVI : bare class to define an event algorithm.

- It reuses an **algorithmTree** (p. 15).
- An event algorithm reflect the "physical" process of an **event**. An event is the process of an interacion of a particle with the detector and how the physicits retrieve the initial information form the detector response. An event has three separated task: **generation** (from the interaction of the particle to the response of the detector) this algorithm creates the Raw data; reconstruction (from the raw data to reconstructed data, data which is still not physical); and analysis (from reconstructed data to physics variables). Example: tracker device: 1) generation: from the interacion of the particle to the list of hits, 2) reconstruction: from the list of hits to reconstructed tracks; 3) analysis from the list of track to the initial direction of the incident particle.
- An event algorithm executes three algorithms (also methods): **generate()** (p. 73), **reconstruct()** (p. 73), and `analize()`. Each one is an **algorithmComposite** (p. 10), so it can be composed of several algorithms.
- The event class uses an **algorithmTree** (p. 15) class and renames the initalize, execute and finalize methods into generate, reconstruct and analyze methods.

Definition at line 31 of file eventVI.h.

4.34.2 Constructor & Destructor Documentation

4.34.2.1 eventVI::eventVI () `[inline]`

constructor, construct the pointers to the algorithmComposites.

Definition at line 38 of file eventVI.h.

4.34.2.2 eventVI::~eventVI () `[inline]`

destructor, destroies the pointers to the algorithmComposites.

Definition at line 40 of file eventVI.h.

4.34.3 Member Function Documentation

4.34.3.1 algorithmComposite ∗ eventVI::anaAlg () const `[inline, protected]`

pointer to the analyze **algorithmComposite** (p. 10).

Definition at line 64 of file eventVI.h.

4.34.3.2 void eventVI::analyze () `[inline, virtual]`

reconstruct the porcess running an **algorithmComposite** (p. 10).

Reimplemented in **eventComposite** (p. 67).

Definition at line 52 of file eventVI.h.

4.34.3.3 void eventVI::command (std::string *com*) `[inline, virtual]`

execute the methods via the command.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 13 of file eventVI.cpp.

4.34.3.4 algorithmComposite * eventVI::genAlg () const `[inline, protected]`

pointer to generate **algorithmComposite** (p. 10).

Definition at line 60 of file eventVI.h.

4.34.3.5 void eventVI::generate () `[inline, virtual]`

generate the process running an **algorithmComposite** (p. 10).

Reimplemented in **eventComposite** (p. 67).

Definition at line 48 of file eventVI.h.

4.34.3.6 algorithmComposite * eventVI::recAlg () const `[inline, protected]`

pointer to the reconstruct **algorithmComposite** (p. 10).

Definition at line 62 of file eventVI.h.

4.34.3.7 void eventVI::reconstruct () `[inline, virtual]`

reconstruct the process running an **algorithmComposite** (p. 10).

Reimplemented in **eventComposite** (p. 67).

Definition at line 50 of file eventVI.h.

4.34.3.8 void eventVI::run () `[inline, virtual]`

overwrite the virtuam method **run**() (p. 73) that now executes the **generate**() (p. 73), **reconstruct**() (p. 73), and analysis() methods;.

this methods are in fact algorithmComposites from **algorithmTree** (p. 15) **initialize**() (p. 17), **execute**() (p. 16) and **finalize**() (p. 17)

Reimplemented from **algorithmVI** (p. 20).

Definition at line 5 of file eventVI.cpp.

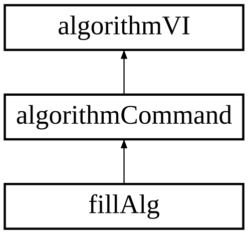
The documentation for this class was generated from the following files:

- **eventVI.h**
- **eventVI.cpp**

4.35 fillAlg Class Reference

`#include <fillAlg.h>`

Inheritance diagram for fillAlg:



Public Methods

- **fillAlg** (**rHistoFolder**∗ histo)

  constructor - set the **rHistoFolder** (p. 125)*.*

- **~fillAlg** ()

  default destructor.

- virtual void **execute** ()

  execute - fill the **rHistoFolder** (p. 125)*.*

Private Attributes

- **rHistoFolder**∗ **m_histo**

  pointer to the **rHistoFolder** (p. 125)*.*

4.35.1 Constructor & Destructor Documentation

4.35.1.1 fillAlg::fillAlg (rHistoFolder ∗ *histo*)

constructor - set the **rHistoFolder** (p. 125).

Definition at line 15 of file fillAlg.h.

4.35.1.2 fillAlg::~fillAlg () `[inline]`

default destructor.

Definition at line 17 of file fillAlg.h.

4.35.2 Member Function Documentation

4.35.2.1 void fillAlg::execute () `[inline, virtual]`

execute - fill the **rHistoFolder** (p. 125).

Reimplemented from **algorithmVI** (p. 19).

Definition at line 20 of file fillAlg.h.

4.35.3 Member Data Documentation

4.35.3.1 rHistoFolder * fillAlg::m_histo `[private]`

pointer to the **rHistoFolder** (p. 125).

Definition at line 25 of file fillAlg.h.

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

- **fillAlg.h**

4.36 IOBase Class Reference

`#include <IOBase.h>`

Public Types

- enum **MODE** { **READ**, **WRITE** }

4.36.1 Member Enumeration Documentation

4.36.1.1 enum IOBase::MODE

Enumeration values:

READ

WRITE

Definition at line 10 of file IOBase.h.

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

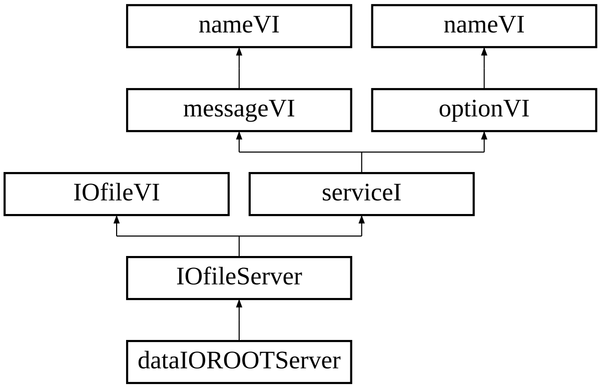
- **IOBase.h**

4.37 IOfileServer Class Reference

Handles the input and output of permanent data files.

`#include <IOfileServer.h>`

Inheritance diagram for IOfileServer:



Public Methods

- virtual void **open** ()

  open the files in server named "input", "output" and "histo".

- virtual void **close** ()

  close the files in server named "input", "output" and "histo".

- virtual bool **nextEvent** ()

  virtual method to know if there is one event more in the file.

- virtual void **readEvent** ()

  virtual method to read the next event - overwrite by the derived class.

- virtual void **writeEvent** ()

  virtual method to write an event - overwrite by the derived class.

- virtual void **skipEvent** ()

  virtual method to skip next event - overwrite by the derived class.

Protected Methods

- **IOfileServer** ()

  constructor - executed defineIOfiles.

- **~IOfileServer** ()

  destructor.

- virtual void **defineIOfiles** () = 0

the User should declare the IOfiles in this method.

- bool **search** (std::string name) const

  returns true is there is a file with that name in server.

- virtual void **addIOfile** (std::string name, **IOfileVI**∗ file)

  add a file into the server.

- **IOfileVI**∗ **getIOfile** (std::string name) const

  returns the **IOfileVI** (p. 84) *file in server.*

- std::string **getNextInFileName** ()

  returns the next input file name to open.

- std::string **getHistoFileName** () const

  returns the histogram file name.

- std::string **getOutFileName** () const

  returns the output file name.

- virtual void **defineOption** ()

  define the options **optionVI** (p. 114)*.*

- virtual void **setOption** (std::string par, std::string v)

  method to define the names of the input/output IO files.

Private Methods

- void **displayFrequency** () const

  display message.

Private Attributes

- **serverVI<IOfileVI>**∗ **m_server**
- int **m_ifile**

  number of the input file currently open.

- std::vector<std::string> **m_InFileList**

  List of input files.

- std::string **m_InFile**

  name of the last input input file.

- std::string **m_histoFile**

  name of the histo file.

- std::string **m_OutFile**

  name of the output file.

- int **m_displayFreq**

  frequency to display a message.

4.37.1 Detailed Description

Handles the input and output of permanent data files.

- The files operations are defined int **IOfileVI** (p. 84)
- It is a server with the input/output data files.

  There are three posible files to open:

  - input permanent data file named "input"
  - output permanent data file named "output"
  - output histogram file named "histo"
- The User can define the fine names of those fileVI via the services of **serviceI** (p. 156). see **defineOption**() (p. 81) implementation.

Definition at line 35 of file IOfileServer.h.

4.37.2 Constructor & Destructor Documentation

4.37.2.1 IOfileServer::IOfileServer () `[protected]`

constructor - executed defineIOfiles.

Definition at line 7 of file IOfileServer.cpp.

4.37.2.2 IOfileServer::~IOfileServer () `[protected]`

destructor.

Definition at line 24 of file IOfileServer.cpp.

4.37.3 Member Function Documentation

4.37.3.1 void IOfileServer::addIOfile (std::string *name*, IOfileVI * *file*) `[inline, protected, virtual]`

add a file into the server.

Definition at line 68 of file IOfileServer.h.

4.37.3.2 void IOfileServer::close () `[inline, virtual]`

close the files in server named "input", "output" and "histo".

Reimplemented from **IOfileVI** (p. 86).

Definition at line 91 of file IOfileServer.cpp.

4.37.3.3 void IOfileServer::defineIOfiles () `[inline, protected, pure virtual]`

the User should declare the IOfiles in this method.

Reimplemented in **dataIOROOTServer** (p. 59).

4.37.3.4 void IOfileServer::defineOption () `[inline, protected, virtual]`

define the options **optionVI** (p. 114).

Reimplemented from **optionVI** (p. 116).

Definition at line 116 of file IOfileServer.cpp.

4.37.3.5 void IOfileServer::displayFrequency () const `[private]`

display message.

Definition at line 100 of file IOfileServer.cpp.

4.37.3.6 std::string IOfileServer::getHistoFileName () const `[inline, protected]`

returns the histogram file name.

Definition at line 78 of file IOfileServer.h.

4.37.3.7 IOfileVI * IOfileServer::getIOfile (std::string *name*) const `[protected]`

returns the **IOfileVI** (p. 84) file in server.

Definition at line 29 of file IOfileServer.cpp.

4.37.3.8 std::string IOfileServer::getNextInFileName () `[protected]`

returns the next input file name to open.

Definition at line 107 of file IOfileServer.cpp.

4.37.3.9 std::string IOfileServer::getOutFileName () const `[inline, protected]`

returns the output file name.

Definition at line 80 of file IOfileServer.h.

4.37.3.10 bool IOfileServer::nextEvent () `[inline, virtual]`

virtual method to know if there is one event more in the file.

Reimplemented from **IOfileVI** (p. 87).

Definition at line 63 of file IOfileServer.cpp.

4.37.3.11 void IOfileServer::open () `[inline, virtual]`

open the files in server named "input", "output" and "histo".

Reimplemented from **IOfileVI** (p. 87).

Definition at line 37 of file IOfileServer.cpp.

4.37.3.12 void IOfileServer::readEvent () `[inline, virtual]`

virtual method to read the next event - overwrite by the derived class.

Reimplemented from **IOfileVI** (p. 87).

Definition at line 70 of file IOfileServer.cpp.

4.37.3.13 bool IOfileServer::search (std::string *name*) const `[inline, protected]`

returns true is there is a file with that name in server.

Definition at line 65 of file IOfileServer.h.

4.37.3.14 void IOfileServer::setOption (std::string *par*, std::string *v*) `[inline, protected, virtual]`

method to define the names of the input/output IO files.

Reimplemented from **optionVI** (p. 117).

Definition at line 126 of file IOfileServer.cpp.

4.37.3.15 void IOfileServer::skipEvent () `[inline, virtual]`

virtual method to skip next event - overwrite by the derived class.

Reimplemented from **IOfileVI** (p. 88).

Definition at line 84 of file IOfileServer.cpp.

4.37.3.16 void IOfileServer::writeEvent () `[inline, virtual]`

virtual method to write an event - overwrite by the derived class.

Reimplemented from **IOfileVI** (p. 88).

Definition at line 77 of file IOfileServer.cpp.

4.37.4 Member Data Documentation

4.37.4.1 std::string IOfileServer::m_InFile `[private]`

name of the last input input file.

Definition at line 101 of file IOfileServer.h.

4.37.4.2 std::vector<std::string> IOfileServer::m_InFileList `[private]`

List of input files.

Definition at line 99 of file IOfileServer.h.

4.37.4.3 std::string IOfileServer::m_OutFile `[private]`

name of the output file.

Definition at line 105 of file IOfileServer.h.

4.37.4.4 int IOfileServer::m_displayFreq `[private]`

frequency to display a message.

Definition at line 108 of file IOfileServer.h.

4.37.4.5 std::string IOfileServer::m_histoFile `[private]`

name of the histo file.

Definition at line 103 of file IOfileServer.h.

4.37.4.6 int IOfileServer::m_ifile `[private]`

number of the input file currently open.

Definition at line 97 of file IOfileServer.h.

4.37.4.7 serverVI<IOfileVI>* IOfileServer::m_server `[private]`

Definition at line 94 of file IOfileServer.h.

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

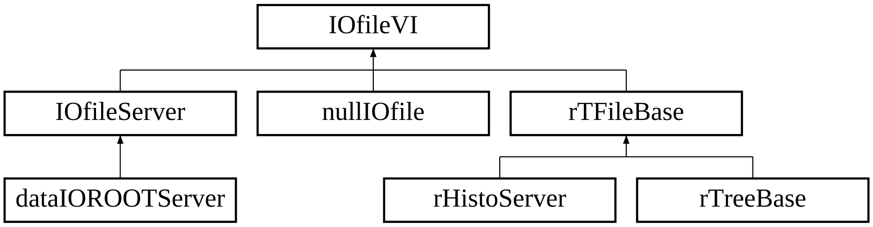
- **IOfileServer.h**
- **IOfileServer.cpp**

4.38 IOfileVI Class Reference

virtual inteface to handle a persisntent data file with a list of events.

`#include <IOfileVI.h>`

Inheritance diagram for IOfileVI:



Public Methods

- **IOfileVI** ()

  default constructor.

- ~**IOfileVI** ()

  default destructor.

- virtual void **setFileName** (std::string name)

  set the name of the file.

- virtual void **setMode** (**IOBase::MODE** m)

  set the mode.

- virtual void **setNumEvents** (int nevt)

  set total number of events.

- virtual void **setOpen** (bool t)

  set the file to Open.

- virtual void **increase** ()

  virtual method to increase one event.

- virtual void **open** () = 0

  virtual method to open the file.

- virtual void **close** () = 0

  virtual method to close the file.

- virtual bool **nextEvent** ()

  virtual method to know if there is one event more in the file.

- virtual void **readEvent** ()

  virtual method to read the next event - overwrite by the derived class.

- virtual void **writeEvent** ()

  virtual method to write an event - overwrite by the derived class.

- virtual void **skipEvent** ()

  virtual method to skip next event - overwrite by the derived class.

- virtual std::string **getFileName** () const

  returns the name of the file.

- **IOBase::MODE getMode** () const

  returns the mode READ or WRITE.

- virtual int **eventNumber** () const

  virtual method to return the current Event.

- virtual int **numEvents** () const

  virtual method ro return the total number of events;.

- virtual bool **isOpen** () const

  virtual method to know if the file is open or not.

Protected Methods

- virtual void **clear** ()

  clear all the variables.

Private Attributes

- std::string **m_nameFile**

  name of the file.

- **IOBase::MODE m_mode**

  mode of open the file READ or WRITE.

- bool **m_open**

  is the file open?

- int **m_numEvents**

  number of total events in the file.

- int **m_currentEvent**

  the current event number.

4.38.1 Detailed Description

virtual inteface to handle a persisntent data file with a list of events.

Definition at line 11 of file IOfileVI.h.

4.38.2 Constructor & Destructor Documentation

4.38.2.1 IOfileVI::IOfileVI () `[inline]`

default constructor.

Definition at line 17 of file IOfileVI.h.

4.38.2.2 IOfileVI::~IOfileVI () `[inline]`

default destructor.

Definition at line 19 of file IOfileVI.h.

4.38.3 Member Function Documentation

4.38.3.1 void IOfileVI::clear () `[inline, protected, virtual]`

clear all the variables.

Reimplemented in **rTFileBase** (p. 134), and **rTreeBase** (p. 137).

Definition at line 5 of file IOfileVI.cpp.

4.38.3.2 void IOfileVI::close () `[inline, pure virtual]`

virtual method to close the file.

Reimplemented in **IOfileServer** (p. 80), **nullIOfile** (p. 108), and **rTFileBase** (p. 134).

4.38.3.3 int IOfileVI::eventNumber () const `[inline, virtual]`

virtual method to return the current Event.

Definition at line 51 of file IOfileVI.h.

4.38.3.4 std::string IOfileVI::getFileName () const `[inline, virtual]`

returns the name of the file.

Definition at line 47 of file IOfileVI.h.

4.38.3.5 IOBase::MODE IOfileVI::getMode () const `[inline]`

returns the mode READ or WRITE.

Definition at line 49 of file IOfileVI.h.

4.38.3.6 void IOfileVI::increase () `[inline, virtual]`

virtual method to increase one event.

Definition at line 30 of file IOfileVI.h.

4.38.3.7 bool IOfileVI::isOpen () const `[inline, virtual]`

virtual method to know if the file is open or not.

Definition at line 55 of file IOfileVI.h.

4.38.3.8 bool IOfileVI::nextEvent () `[inline, virtual]`

virtual method to know if there is one event more in the file.

Reimplemented in **IOfileServer** (p. 81).

Definition at line 38 of file IOfileVI.h.

4.38.3.9 int IOfileVI::numEvents () const `[inline, virtual]`

virtual method ro return the total number of events;.

Definition at line 53 of file IOfileVI.h.

4.38.3.10 void IOfileVI::open () `[inline, pure virtual]`

virtual method to open the file.

Reimplemented in **IOfileServer** (p. 81), **nullIOfile** (p. 108), **rHistoServer** (p. 132), **rTFileBase** (p. 134), and **rTreeBase** (p. 138).

4.38.3.11 void IOfileVI::readEvent () `[inline, virtual]`

virtual method to read the next event - overwrite by the derived class.

Reimplemented in **IOfileServer** (p. 82), and **rTreeBase** (p. 138).

Definition at line 40 of file IOfileVI.h.

4.38.3.12 void IOfileVI::setFileName (std::string *name*) `[inline, virtual]`

set the name of the file.

Definition at line 22 of file IOfileVI.h.

4.38.3.13 void IOfileVI::setMode (IOBase::MODE *m*) `[inline, virtual]`

set the mode.

Definition at line 24 of file IOfileVI.h.

4.38.3.14 void IOfileVI::setNumEvents (int *nevt*) `[inline, virtual]`

set total number of events.

Definition at line 26 of file IOfileVI.h.

4.38.3.15 void IOfileVI::setOpen (bool *t*) `[inline, virtual]`

set the file to Open.

Definition at line 28 of file IOfileVI.h.

4.38.3.16 void IOfileVI::skipEvent () `[inline, virtual]`

virtual method to skip next event - overwrite by the derived class.

Reimplemented in **IOfileServer** (p. 82), and **rTreeBase** (p. 138).

Definition at line 44 of file IOfileVI.h.

4.38.3.17 void IOfileVI::writeEvent () `[inline, virtual]`

virtual method to write an event - overwrite by the derived class.

Reimplemented in **IOfileServer** (p. 82), and **rTreeBase** (p. 138).

Definition at line 42 of file IOfileVI.h.

4.38.4 Member Data Documentation

4.38.4.1 int IOfileVI::m_currentEvent `[private]`

the current event number.

Definition at line 74 of file IOfileVI.h.

4.38.4.2 IOBase::MODE IOfileVI::m_mode `[private]`

mode of open the file READ or WRITE.

Definition at line 67 of file IOfileVI.h.

4.38.4.3 std::string IOfileVI::m_nameFile `[private]`

name of the file.

Definition at line 65 of file IOfileVI.h.

4.38.4.4 int IOfileVI::m_numEvents `[private]`

number of total events in the file.

Definition at line 72 of file IOfileVI.h.

4.38.4.5 bool IOfileVI::m_open `[private]`

is the file open?

Definition at line 69 of file IOfileVI.h.

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

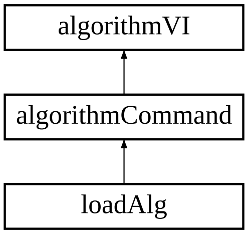
- **IOfileVI.h**
- **IOfileVI.cpp**

4.39 loadAlg Class Reference

transform a converter from persistent to transient into an algorithm.

`#include <loadAlg.h>`

Inheritance diagram for loadAlg:



Public Methods

- **loadAlg** (**converterVI**∗ conv)

  constructor - set the converter.

- **~loadAlg** ()

  destructor.

- void **execute** ()

  execute export() method.

Private Attributes

- **converterVI**∗ **m_conv**

  pointer to the converter.

4.39.1 Detailed Description

transform a converter from persistent to transient into an algorithm.

Definition at line 11 of file loadAlg.h.

4.39.2 Constructor & Destructor Documentation

4.39.2.1 loadAlg::loadAlg (converterVI ∗ *conv*)

constructor - set the converter.

Definition at line 16 of file loadAlg.h.

4.39.2.2 loadAlg::~loadAlg () `[inline]`

destructor.

Definition at line 18 of file loadAlg.h.

4.39.3 Member Function Documentation

4.39.3.1 void loadAlg::execute () `[inline, virtual]`

execute export() method.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 21 of file loadAlg.h.

4.39.4 Member Data Documentation

4.39.4.1 converterVI * loadAlg::m_conv `[private]`

pointer to the converter.

Definition at line 26 of file loadAlg.h.

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

- **loadAlg.h**

4.40 messageBase Class Reference

`#include <messageBase.h>`

Public Types

- enum **LEVEL** { **GENERAL**, **RELEASE**, **DEBUG**, **DEBUG_MEDIUM**, **DEBUG_HIGH** }

  enumeration of the different level of information/debugging.

Static Public Methods

- enum **LEVEL getLevel** (std::string level)

  returns the enumeration of the LEVEL from a string.

4.40.1 Member Enumeration Documentation

4.40.1.1 enum messageBase::LEVEL

enumeration of the different level of information/debugging.

Enumeration values:

GENERAL

RELEASE

DEBUG

DEBUG_MEDIUM

DEBUG_HIGH

Definition at line 13 of file messageBase.h.

4.40.2 Member Function Documentation

4.40.2.1 enum LEVEL messageBase::getLevel (std::string *level*) `[static]`

returns the enumeration of the LEVEL from a string.

Definition at line 4 of file messageBase.cpp.

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

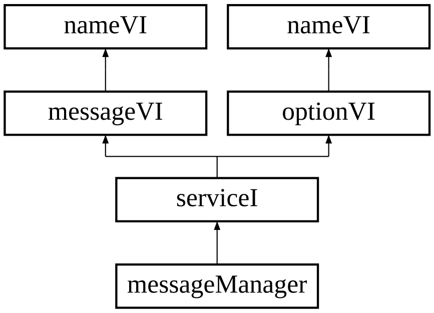
- **messageBase.h**
- **messageBase.cpp**

4.41 messageManager Class Reference

Manager to handle the message of the classes and define an ostream output.

`#include <messageManager.h>`

Inheritance diagram for messageManager:



Public Methods

- **messageManager** ()

  default constructor.

- ~**messageManager** ()

  default constructor.

- std::ostream& **out** () const

  ostream of the output.

- void **open** (std::string fileName)

  open the output fileName file for messages.

- void **close** ()

  close the output file for messages.

- void **message** (std::string message, std::string ilevel = "RELEASE") const

  *send a string message with level (***messageVI** (p. 99)*).*

- void **message** (std::string message, double val, std::string ilevel = "RELEASE") const

  *send a string + double message with level (***messageVI** (p. 99)*).*

- bool **acceptLevel** (std::string level) const

  make it public.

- void **writeOut** (std::string name)

  execute the **writeOut***()* (p. 97) *method for the object with name;.*

- virtual void **writeOut** () const

  writeOut the information of the messageManager only.

Static Public Methods

- messageManager∗ **instance** ()

  Singleton.

Protected Methods

- bool **acceptLevel** () const

  accept the level.

- **messageVI**∗ **getMessage** (std::string name)

  returns the **messageVI** (p. 99) *of the object with name.*

- **serverVI<messageVI>**∗ **server** ()

  returns the server.

- virtual void **defineOption** ()

  define the options of this class.

- virtual void **defineMessage** ()

  include **optionManager** (p. 109) *in the services of messageManager.*

- virtual void **setOption** (std::string var, std::string content)

  overwrites virtual method to define option.

- virtual void **setInServer** (std::string name, **messageVI**∗ mp)

  add an object, with a name into the server.

Private Attributes

- **serverVI<messageVI>**∗ **m_server**

  server with the classes derived from **messageVI** (p. 99)*.*

- std::string **m_level**

  level of the message.

- std::string **m_fileName**

  name of the output file.

- bool **m_definedFile**

  bool is the file has been defined.

- std::ofstream **m_file**

  the ostream of the output.

Static Private Attributes

- messageManager∗ **m_instance** = 0

  Singleton: pointer to itself.

Friends

- class **messageVI**

4.41.1 Detailed Description

Manager to handle the message of the classes and define an ostream output.

- It defines an ostream& output (method `out()`) where to send the messages. The output can be a file or the screen (std::cout). The method **`open()`** (p. 97) define and open a file to send the messages to.
- A collection of **message**() (p. 96) methods allows a external class to send a message (either a string or a string + a double value) to the ostream output. The messageManager has a enumeration LEVEL (from **messageBase** (p. 92)) that defines which are the messages that the messageManager will process out, via the **`acceptLevel()`** (p. 95) method. The **message**() (p. 96) methods are inherits and overwriten from **messageVI** (p. 99).
- It has a protected server with the list of classes that can be served with messages (that inherit from **messageVI** (p. 99)). the method `writeOut(name)` executes the **`writeOut()`** (p. 97) method of the object with name stored in the server.

Definition at line 33 of file messageManager.h.

4.41.2 Constructor & Destructor Documentation

4.41.2.1 messageManager::messageManager ()

default constructor.

Definition at line 8 of file messageManager.cpp.

4.41.2.2 messageManager::∼messageManager ()

default constructor.

Definition at line 47 of file messageManager.cpp.

4.41.3 Member Function Documentation

4.41.3.1 bool messageManager::acceptLevel () const `[inline, protected]`

accept the level.

Reimplemented from **messageVI** (p. 101).

Definition at line 74 of file messageManager.h.

4.41.3.2 bool messageManager::acceptLevel (std::string *level*) const `[inline]`

make it public.

Reimplemented from **messageVI** (p. 101).

Definition at line 63 of file messageManager.h.

4.41.3.3 void messageManager::close ()

close the output file for messages.

Definition at line 32 of file messageManager.cpp.

4.41.3.4 void messageManager::defineMessage () `[inline, protected, virtual]`

include **optionManager** (p. 109) in the services of messageManager.

Reimplemented from **serviceI** (p. 157).

Definition at line 88 of file messageManager.cpp.

4.41.3.5 void messageManager::defineOption () `[inline, protected, virtual]`

define the options of this class.

Reimplemented from **optionVI** (p. 116).

Definition at line 80 of file messageManager.cpp.

4.41.3.6 messageVI * messageManager::getMessage (std::string *name*) `[protected]`

returns the **messageVI** (p. 99) of the object with name.

Definition at line 95 of file messageManager.cpp.

4.41.3.7 messageManager * messageManager::instance () `[inline, static]`

Singleton.

Definition at line 47 of file messageManager.h.

4.41.3.8 void messageManager::message (std::string *message*, double *val*, std::string *ilevel* = "RELEASE") const

send a string + double message with level (**messageVI** (p. 99)).

Reimplemented from **messageVI** (p. 101).

Definition at line 73 of file messageManager.cpp.

4.41.3.9 void messageManager::message (std::string *message*, std::string *ilevel* = "RELEASE") const

send a string message with level (**messageVI** (p. 99)).

Reimplemented from **messageVI** (p. 102).

Definition at line 67 of file messageManager.cpp.

4.41.3.10 void messageManager::open (std::string *fileName*)

open the output fileName file for messages.

Definition at line 23 of file messageManager.cpp.

4.41.3.11 std::ostream & messageManager::out () const

ostream of the output.

Definition at line 59 of file messageManager.cpp.

4.41.3.12 serverVI<messageVI>∗ messageManager::server () `[inline, protected]`

returns the server.

Definition at line 80 of file messageManager.h.

4.41.3.13 void messageManager::setInServer (std::string *name*, messageVI ∗ *mp*) `[inline, protected, virtual]`

add an object, with a name into the server.

Definition at line 53 of file messageManager.cpp.

4.41.3.14 void messageManager::setOption (std::string *var*, std::string *content*) `[inline, protected, virtual]`

overwrites virtual method to define option.

It open the file or sets the level

Reimplemented from **optionVI** (p. 117).

Definition at line 39 of file messageManager.cpp.

4.41.3.15 void messageManager::writeOut () const `[inline, virtual]`

writeOut the information of the messageManager only.

Reimplemented from **serviceI** (p. 158).

Definition at line 106 of file messageManager.cpp.

4.41.3.16 void messageManager::writeOut (std::string *name*) `[inline]`

execute the **writeOut**() (p. 97) method for the object with name;.

Definition at line 66 of file messageManager.h.

4.41.4 Friends And Related Function Documentation

4.41.4.1 class messageVI `[friend]`

Definition at line 36 of file messageManager.h.

4.41.5 Member Data Documentation

4.41.5.1 bool messageManager::m_definedFile `[private]`

bool is the file has been defined.

Definition at line 108 of file messageManager.h.

4.41.5.2 std::ofstream messageManager::m_file `[private]`

the ostream of the output.

Definition at line 111 of file messageManager.h.

4.41.5.3 std::string messageManager::m_fileName `[private]`

name of the output file.

Definition at line 105 of file messageManager.h.

4.41.5.4 messageManager * messageManager::m_instance = 0 `[static, private]`

Singleton: pointer to itself.

Definition at line 97 of file messageManager.h.

4.41.5.5 std::string messageManager::m_level `[private]`

level of the message.

Reimplemented from **messageVI** (p. 102).

Definition at line 103 of file messageManager.h.

4.41.5.6 serverVI<messageVI>* messageManager::m_server `[private]`

server with the classes derived from **messageVI** (p. 99).

Definition at line 100 of file messageManager.h.

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

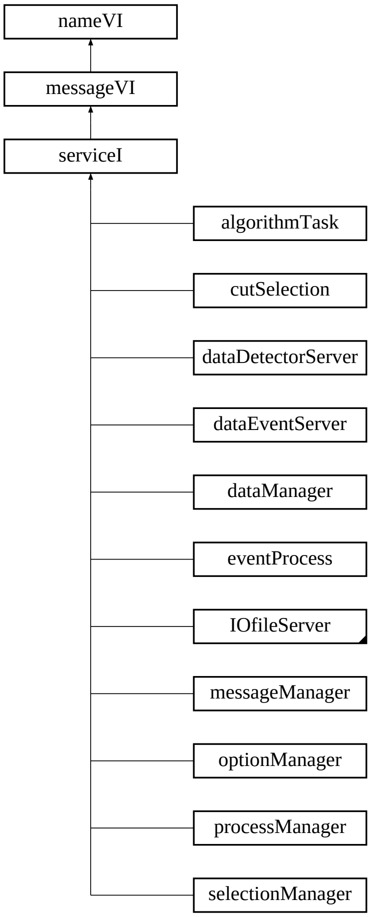
- **messageManager.h**
- **messageManager.cpp**

4.42 messageVI Class Reference

Interface class between to send messages to the **messageManager** (p. 93).

`#include <messageVI.h>`

Inheritance diagram for messageVI:



Protected Methods

- **messageVI** ()

  constructor - set the level to the lowest.

- **~messageVI** ()

  destructor.

- virtual void **setName** (std::string name)

  gives the name to the object and store it into the **messageManager** (p. 93) *server.*

- void **setLevel** (std::string lev)

  sets the Level - make a conversion from string to enum LEVEL.

- bool **acceptLevel** () const

returns true is the info should be expresed.

- virtual void **defineMessage** ()

  base virtual method to define the message to the derived class.

- virtual void **writeOut** () const

  base virtual method to write information of the derived class via the **messageManager** (p. 93).

- bool **acceptLevel** (std::string lev) const

  returns true this level should be expressed according with the string level.

- void **message** (std::string message, std::string level="GENERAL") const

  send string messages to the **messageManager** (p. 93).

- void **message** (std::string message, double value, std::string level="GENERAL") const

  send double value messages to the **messageManager** (p. 93).

- enum **messageBase::LEVEL getLevel** () const

  returns the enum LEVEL of this class.

Private Attributes

- enum **messageBase::LEVEL m_level**

  level of this class.

Friends

- class **messageManager**

4.42.1 Detailed Description

Interface class between to send messages to the **messageManager** (p. 93).

- The method **message()** (p. 102) allows the derived class to send messages (either strings, or string + double values) to the **messageManager** (p. 93).
- It defines a LEVEL (from **messageBase** (p. 92)) indicating which is the level of information provided by the derived class. The **acceptLevel()** (p. 101) indicates when the level of the class is superior to the level of the **messageManager** (p. 93).
- The virtual method **writeOut()** (p. 102) is the place where the derived class should put the information that wants to write out. this method could/should be overwriten by the User.
- The method **setName()** (p. 102) set the derived object into the server of **messageManager** (p. 93), the object will be refered with it name. In addition executed the virtual method **defineMessage**() (p. 101), this method, it used by the derived class to make some implementations obut the message of the class.

Definition at line 26 of file messageVI.h.

4.42.2 Constructor & Destructor Documentation

4.42.2.1 messageVI::messageVI () `[protected]`

constructor - set the level to the lowest.

Definition at line 34 of file messageVI.h.

4.42.2.2 messageVI::~messageVI () `[inline, protected]`

destructor.

Definition at line 36 of file messageVI.h.

4.42.3 Member Function Documentation

4.42.3.1 bool messageVI::acceptLevel (std::string *level*) const `[protected]`

returns true this level should be expressed according with the string level.

compare the level of the derived class with those one of the **messageManager** (p. 93)

Reimplemented in **messageManager** (p. 95).

Definition at line 40 of file messageVI.cpp.

4.42.3.2 bool messageVI::acceptLevel () const `[protected]`

returns true is the info should be expresed.

compare the level of the derived class with those one of the **messageManager** (p. 93)

Reimplemented in **messageManager** (p. 95).

Definition at line 46 of file messageVI.cpp.

4.42.3.3 void messageVI::defineMessage () `[inline, protected, virtual]`

base virtual method to define the message to the derived class.

Reimplemented in **messageManager** (p. 96), and **serviceI** (p. 157).

Definition at line 51 of file messageVI.h.

4.42.3.4 enum messageBase::LEVEL messageVI::getLevel () const `[inline, protected]`

returns the enum LEVEL of this class.

Definition at line 64 of file messageVI.h.

4.42.3.5 void messageVI::message (std::string *message*, double *val*, std::string *ilevel* = "GENERAL") const `[protected]`

send double value messages to the **messageManager** (p. 93).

Reimplemented in **messageManager** (p. 96).

Definition at line 32 of file messageVI.cpp.

4.42.3.6 void messageVI::message (std::string *message*, std::string *ilevel* = "GENERAL") const `[protected]`

send string messages to the **messageManager** (p. 93).

Reimplemented in **messageManager** (p. 96).

Definition at line 26 of file messageVI.cpp.

4.42.3.7 void messageVI::setLevel (std::string *lev*) `[inline, protected]`

sets the Level - make a conversion from string to enum LEVEL.

Definition at line 43 of file messageVI.h.

4.42.3.8 void messageVI::setName (std::string *name*) `[inline, protected, virtual]`

gives the name to the object and store it into the **messageManager** (p. 93) server.

It also calls **defineMessage**() (p. 101)

Reimplemented from **nameVI** (p. 105).

Reimplemented in **serviceI** (p. 157).

Definition at line 6 of file messageVI.cpp.

4.42.3.9 void messageVI::writeOut () const `[inline, protected, virtual]`

base virtual method to write information of the derived class via the **messageManager** (p. 93).

Reimplemented in **algorithmTask** (p. 13), **cutSelection** (p. 47), **dataDetectorServer** (p. 53), **dataEventServer** (p. 55), **eventProcess** (p. 69), **messageManager** (p. 97), **optionManager** (p. 112), **processManager** (p. 122), **selectionManager** (p. 150), and **serviceI** (p. 158).

Definition at line 18 of file messageVI.cpp.

4.42.4 Friends And Related Function Documentation

4.42.4.1 class messageManager `[friend]`

Definition at line 29 of file messageVI.h.

4.42.5 Member Data Documentation

4.42.5.1 enum messageBase::LEVEL messageVI::m_level `[private]`

level of this class.

Reimplemented in **messageManager** (p. 98).

Definition at line 69 of file messageVI.h.

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

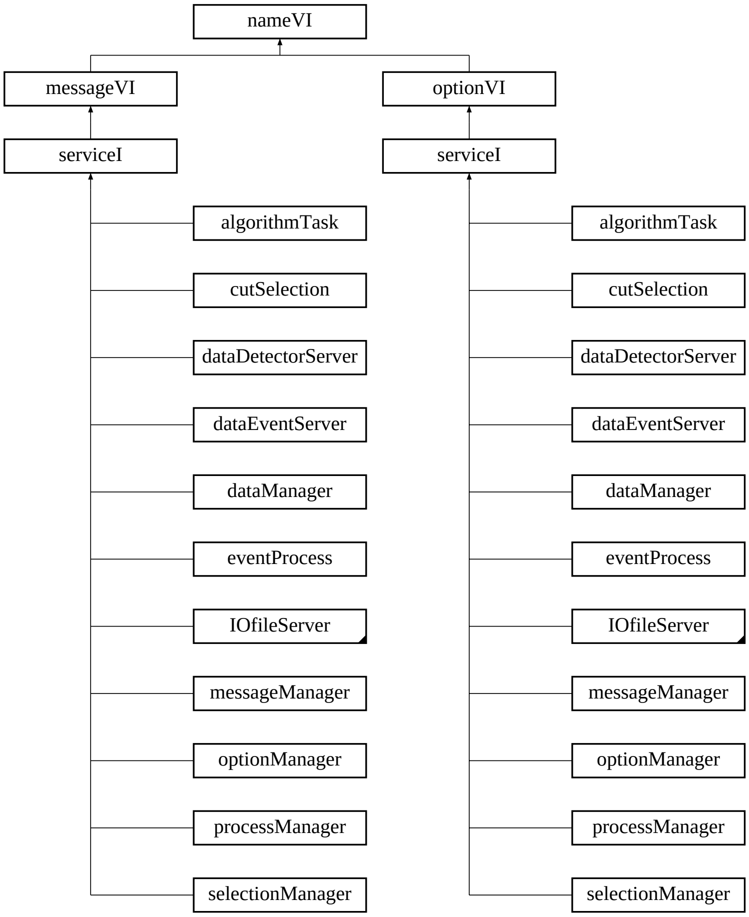
- **messageVI.h**
- **messageVI.cpp**

4.43 nameVI Class Reference

bare class to set and store a name to an object.

`#include <nameVI.h>`

Inheritance diagram for nameVI:



Protected Methods

- **nameVI** (std::string name)

  constructor - set the name.

- **~nameVI** ()

  default destructor.

- virtual std::string **name** () const

  returns the name.

- virtual void **setName** (std::string name)

  set the name.

Private Attributes

- std::string **m_name**

  the name of the object.

4.43.1 Detailed Description

bare class to set and store a name to an object.

Definition at line 9 of file nameVI.h.

4.43.2 Constructor & Destructor Documentation

4.43.2.1 nameVI::nameVI (std::string *name*) `[protected]`

constructor - set the name.

Definition at line 15 of file nameVI.h.

4.43.2.2 nameVI::~nameVI () `[inline, protected]`

default destructor.

Definition at line 17 of file nameVI.h.

4.43.3 Member Function Documentation

4.43.3.1 std::string nameVI::name () const `[inline, protected, virtual]`

returns the name.

Reimplemented in **serviceI** (p. 157).

Definition at line 20 of file nameVI.h.

4.43.3.2 void nameVI::setName (std::string *name*) `[inline, protected, virtual]`

set the name.

Reimplemented in **messageVI** (p. 102), **optionVI** (p. 116), and **serviceI** (p. 157).

Definition at line 23 of file nameVI.h.

4.43.4 Member Data Documentation

4.43.4.1 std::string nameVI::m_name `[private]`

the name of the object.

Definition at line 28 of file nameVI.h.

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

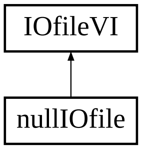
- **nameVI.h**

4.44 nullIOfile Class Reference

null **IOfileVI** (p. 84).

`#include <nullIOfile.h>`

Inheritance diagram for nullIOfile:



Public Methods

- **nullIOfile** ()

  default constructor.

- **~nullIOfile** ()

  default destructor.

- void **open** ()

  null open.

- void **close** ()

  null close.

4.44.1 Detailed Description

null **IOfileVI** (p. 84).

The open, close functions are null

Definition at line 13 of file nullIOfile.h.

4.44.2 Constructor & Destructor Documentation

4.44.2.1 nullIOfile::nullIOfile () `[inline]`

default constructor.

Definition at line 19 of file nullIOfile.h.

4.44.2.2 nullIOfile::~nullIOfile () `[inline]`

default destructor.

Definition at line 21 of file nullIOfile.h.

4.44.3 Member Function Documentation

4.44.3.1 void nullIOfile::close () `[inline, virtual]`

null close.

Reimplemented from **IOfileVI** (p. 86).

Definition at line 26 of file nullIOfile.h.

4.44.3.2 void nullIOfile::open () `[inline, virtual]`

null open.

Reimplemented from **IOfileVI** (p. 87).

Definition at line 24 of file nullIOfile.h.

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

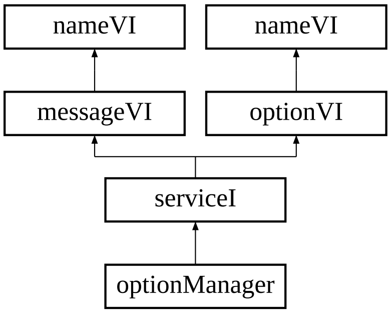
- **nullIOfile.h**

4.45 optionManager Class Reference

Option Manager.

`#include <optionManager.h>`

Inheritance diagram for optionManager:



Public Methods

- **optionManager** ()

  Default constructor.

- ~**optionManager** ()

  default destructor.

- void **setOption** (std::string className, std::string par, std::string con)

  set a string option in an object named className inserted in the server.

- void **setOption** (std::string className, std::string par, double d)

  set a double option in an object named className.

- void **setOption** (std::string className, std::string par, int i)

  set a int option in an object named className.

- void **setOption** ()

  set a collection of options from a external file (set in m_optionFileName);.

- void **setOption** (int argc, char∗ argv[])

  reads the input line C arguments and translate them into set options.

- virtual void **writeOut** () const

  info of the class.

Static Public Methods

- optionManager∗ **instance** ()

  Singleton: returns pointer to the class.

Protected Methods

- **serverVI<optionVI>∗ server** () const

  returns a pointer to the server of the **optionVI** (p. 114) *classes.*

- **optionVI∗ getOption** (std::string name) const

  returns the **optionVI** (p. 114) *of the object with name.*

- virtual void **defineOption** ()

  define the options.

- virtual void **setInServer** (std::string name, **optionVI**∗ op)

  method to include a **optionVI** (p. 114) *class in the server.*

Private Attributes

- **serverVI<optionVI>∗ m_server**

  Server with the classes where **optionVI** (p. 114) *can be apply().*

- std::string **m_optionFileName**

  default ASCII file with the list of options.

Static Private Attributes

- optionManager∗ **m_instance** = 0

  Singleton: pointer to this class.

Friends

- class **optionVI**

4.45.1 Detailed Description

Option Manager.

OptionManager sets options to classed derived from **optionVI** (p. 114) an declated in ther server. *OptionManager* read and sets options from a file (i.e centella.in).

- The options can be set to classes derived from **optionVI** (p. 114). In order to declare the object into the server the **setName**() (p. 157) method from **optionVI** (p. 114) should be executed. Every object should have a name and should be unique.
- The set option methods `setOption(objectName`, parName, parContent) set an option into a object of name objectName. The option is called parName and the content is parContent. Note that this methods are public.
- The **setOption()** (p. 112) method allows to set a collection of options from a external file. In that file the option should be indicated in this way: "*objectName* parName parType parContent", where *parType* can be "S" for string, "D" for double or "I" for integer. A line started with "//" in the file will be ignored (it can be use to put comments).

- The method `setOption(int`, char∗) allows to pass the C **main**() (p. 213) arguments into the optionManager and handle them as convinient.
- The manager has a protected **serverVI** (p. 152) of **optionVI** (p. 114) where the objects declared, and to which the **setOption**() (p. 112) methods can be apply. The options are apply only to object declared into the server!.

Definition at line 35 of file optionManager.h.

4.45.2 Constructor & Destructor Documentation

4.45.2.1 optionManager::optionManager ()

Default constructor.

Definition at line 8 of file optionManager.cpp.

4.45.2.2 optionManager::∼optionManager () `[inline]`

default destructor.

Definition at line 46 of file optionManager.h.

4.45.3 Member Function Documentation

4.45.3.1 void optionManager::defineOption () `[inline, protected, virtual]`

define the options.

Reimplemented from **optionVI** (p. 116).

Definition at line 131 of file optionManager.cpp.

4.45.3.2 optionVI ∗ optionManager::getOption (std::string *name*) const `[protected]`

returns the **optionVI** (p. 114) of the object with name.

Definition at line 115 of file optionManager.cpp.

4.45.3.3 optionManager ∗ optionManager::instance () `[inline, static]`

Singleton: returns pointer to the class.

Definition at line 49 of file optionManager.h.

4.45.3.4 serverVI<optionVI>∗ optionManager::server () const `[inline, protected]`

returns a pointer to the server of the **optionVI** (p. 114) classes.

Definition at line 70 of file optionManager.h.

4.45.3.5 void optionManager::setInServer (std::string *name*, optionVI * *op*) `[inline, protected, virtual]`

method to include a **optionVI** (p. 114) class in the server.

Definition at line 125 of file optionManager.cpp.

4.45.3.6 void optionManager::setOption (int *argc*, char * *argv*[ ])

reads the input line C arguments and translate them into set options.

Definition at line 23 of file optionManager.cpp.

4.45.3.7 void optionManager::setOption ()

set a collection of options from a external file (set in m_optionFileName);.

The first argument of the main C arg is the name of the input file

Definition at line 32 of file optionManager.cpp.

4.45.3.8 void optionManager::setOption (std::string *className*, std::string *par*, int i)

set a int option in an object named className.

Definition at line 82 of file optionManager.cpp.

4.45.3.9 void optionManager::setOption (std::string *className*, std::string *par*, double *d*)

set a double option in an object named className.

Definition at line 82 of file optionManager.cpp.

4.45.3.10 void optionManager::setOption (std::string *className*, std::string *par*, std::string *con*)

set a string option in an object named className inserted in the server.

Definition at line 82 of file optionManager.cpp.

4.45.3.11 void optionManager::writeOut () const `[inline, virtual]`

info of the class.

Reimplemented from **serviceI** (p. 158).

Definition at line 137 of file optionManager.cpp.

4.45.4 Friends And Related Function Documentation

4.45.4.1 class optionVI `[friend]`

Definition at line 38 of file optionManager.h.

4.45.5 Member Data Documentation

4.45.5.1 optionManager * optionManager::m_instance = 0 `[static, private]`

Singleton: pointer to this class.

Definition at line 86 of file optionManager.h.

4.45.5.2 std::string optionManager::m_optionFileName `[private]`

default ASCII file with the list of options.

Definition at line 92 of file optionManager.h.

4.45.5.3 serverVI<optionVI>* optionManager::m_server `[private]`

Server with the classes where **optionVI** (p. 114) can be apply().

Definition at line 89 of file optionManager.h.

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

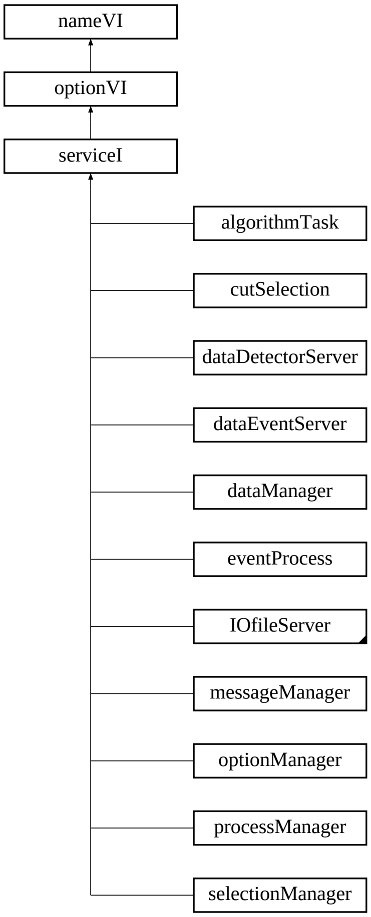
- **optionManager.h**
- **optionManager.cpp**

4.46 optionVI Class Reference

Interface to set options externally into a derived class.

`#include <optionVI.h>`

Inheritance diagram for optionVI:



Protected Methods

- **optionVI** ()

  constructor - set an empty name.

- ~**optionVI** ()

  default destructor.

- virtual void **setName** (std::string name)

  set a name for the class and store it into the server of **optionManager** (p. 109).

- virtual void **defineOption** ()

  virtual method to be overwriten to define the options of the derived class.

- void **defineOption** (std::string name, std::string$*$ st) *define a string parameter of the derived class as option string.*

- void **defineOption** (std::string name, double∗ d)

  define a double parameter of the derived class as option double.

- void **defineOption** (std::string name, int∗ i)

  define a integer paramter of the derived class as option integer.

- virtual void **setOption** (std::string name, std::string c)

  virtual method to set the option in a string parameter.

- virtual void **setOption** (std::string name, double c)

  virtual method to set the option in a double parameter.

- virtual void **setOption** (std::string name, int c)

  virtual method to set the option in a integer/bool parameter.

Protected Attributes

- **serverVI**<std::string> **m_sserver**

  string option server.

- **serverVI**<double> **m_dserver**

  double option server.

- **serverVI**<int> **m_iserver**

  integer option server.

Friends

- class **optionManager**

4.46.1 Detailed Description

Interface to set options externally into a derived class.

- The derived class can declare some of its parameters as options. The parameters should be strings, doubles, integeres or bool. The derived class should rewrite the virtual method **defineOption()** (p. 116) to include the `defineOption(parName`, parPointer) methods with the parameter name and the pointer to this parameter. Usually `defineOptions()` is just a collection of `defineOption(parname`, parpointer).
- The derived class or a external friend class (**optionManager** (p. 109)) can set the value of options via the methods `setOption(parName`, parContent). The options are identified with their names.
- An object of the derived class is declated and set in the **optionManager** (p. 109) server via the **setName()** (p. 116) method. Every object should have a unique name. Unless a name is declared for the object, this will not be included in the servers. The **setName()** (p. 116) method also executed **defineOption()** (p. 116), so this method does not need to be in the constructor of the class.

- Note that the methods **setOption()** (p. 117) are not protected and they not returns information of errors. The external class that set the options should handle that.

Definition at line 35 of file optionVI.h.

4.46.2 Constructor & Destructor Documentation

4.46.2.1 optionVI::optionVI () `[protected]`

constructor - set an empty name.

Definition at line 44 of file optionVI.h.

4.46.2.2 optionVI::~optionVI () `[inline, protected]`

default destructor.

Definition at line 46 of file optionVI.h.

4.46.3 Member Function Documentation

4.46.3.1 void optionVI::defineOption (std::string *name*, int * *i*) `[inline, protected]`

define a integer paramter of the derived class as option integer.

Definition at line 58 of file optionVI.h.

4.46.3.2 void optionVI::defineOption (std::string *name*, double * *d*) `[inline, protected]`

define a double parameter of the derived class as option double.

Definition at line 56 of file optionVI.h.

4.46.3.3 void optionVI::defineOption (std::string *name*, std::string * *st*) `[inline, protected]`

define a string parameter of the derived class as option string.

Definition at line 54 of file optionVI.h.

4.46.3.4 void optionVI::defineOption () `[inline, protected, virtual]`

virtual method to be overwriten to define the options of the derived class.

Reimplemented in **IOfileServer** (p. 81), **algorithmTask** (p. 13), **cutSelection** (p. 47), **eventProcess** (p. 69), **messageManager** (p. 96), **optionManager** (p. 111), **processManager** (p. 122), and **selectionManager** (p. 150).

Definition at line 52 of file optionVI.h.

4.46.3.5 void optionVI::setName (std::string *name*) `[inline, protected, virtual]`

set a name for the class and store it into the server of **optionManager** (p. 109).

Reimplemented from **nameVI** (p. 105).

Reimplemented in **serviceI** (p. 157).

Definition at line 6 of file optionVI.cpp.

4.46.3.6 void optionVI::setOption (std::string *selectionName*, int *c*) `[inline, protected, virtual]`

virtual method to set the option in a integer/bool parameter.

Definition at line 65 of file optionVI.h.

4.46.3.7 void optionVI::setOption (std::string *selectionName*, double *c*) `[inline, protected, virtual]`

virtual method to set the option in a double parameter.

Definition at line 63 of file optionVI.h.

4.46.3.8 void optionVI::setOption (std::string *selectionName*, std::string *cutName*) `[inline, protected, virtual]`

virtual method to set the option in a string parameter.

Reimplemented in **IOfileServer** (p. 82), **algorithmTask** (p. 13), **cutSelection** (p. 47), **eventProcess** (p. 69), **messageManager** (p. 97), **processManager** (p. 122), and **selectionManager** (p. 150).

Definition at line 61 of file optionVI.h.

4.46.4 Friends And Related Function Documentation

4.46.4.1 class optionManager `[friend]`

To execute the options by **optionManager** (p. 109)

Definition at line 39 of file optionVI.h.

4.46.5 Member Data Documentation

4.46.5.1 serverVI<double> optionVI::m_dserver `[protected]`

double option server.

Definition at line 72 of file optionVI.h.

4.46.5.2 serverVI<int> optionVI::m_iserver `[protected]`

integer option server.

Definition at line 74 of file optionVI.h.

4.46.5.3 serverVI<std::string> optionVI::m_sserver `[protected]`

string option server.

Definition at line 70 of file optionVI.h.

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

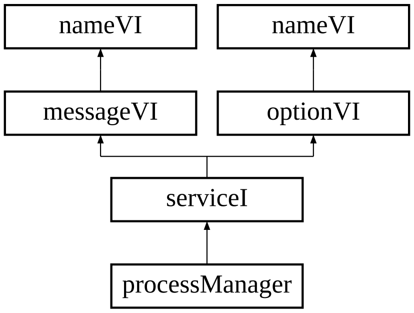
- **optionVI.h**
- **optionVI.cpp**

4.47 processManager Class Reference

Handle the algorithms and the multialgorithms.

`#include <processManager.h>`

Inheritance diagram for processManager:



Public Methods

- **processManager** ()

  constructor - creates the algorithmUserServer.

- **~processManager** ()

  destructor - delete the algorithmUserServer.

- **algorithmVI**∗ **getAlgorithm** (std::string name) const

  returns an algorithm with an name.

- void **addAlgorithm** (std::string name, **algorithmVI**∗ alg)

  add algorithms into the server.

- virtual void **writeOut** () const

  write the information of this class.

Static Public Methods

- processManager∗ **instance** ()

  Singleton, pointer to the class.

Protected Methods

- **eventProcess**∗ **getEventProcess** (std::string name) const

  returns a **eventProcess** (p. 68) *with a name.*

- **algorithmTask**∗ **getAlgorithmTask** (std::string name) const

  returns a **algorithmTask** (p. 12) *with a name.*

- virtual void **defineOption** ()

  define the options.

- virtual void **setOption** (std::string type, std::string name)

  create and include the evtProcess and endRunProcess that the User wants to execute.

Private Attributes

- **serverVI<algorithmVI>* m_algorithms**

  pointer to the server of algorithmUserServer.

- **serverVI<eventProcess>* m_eventProcesses**

  pointer to the evtProcess Server.

- **serverVI<algorithmTask>* m_algorithmTasks**

  pointer to the server of endRunProcess.

- **algorithmTask* m_iniRun**

  algorithmTask (p. 12) *for the initialization of the run.*

- **algorithmTask* m_endRun**

  algorithmTask (p. 12) *for the end of the run.*

- **eventProcess* m_runEvent**

  algorithmTask (p. 12) *for process an event.*

- std::string **m_eventProcessName**

  dummy string to **optionVI** (p. 114)*.*

- std::string **m_algorithmTaskName**

  dummy string to **optionVI** (p. 114)*.*

Static Private Attributes

- processManager* **m_instance** = 0

  Singleton: pointer to itself.

Friends

- class **eventProcess**
- class **algorithmTask**

4.47.1 Detailed Description

Handle the algorithms and the multialgorithms.

ProcessManager stores the system and user declared algorithms.

ProcessManager returns the algorithms via the template method **getAlgorithm()** (p. 122).

- The system and user algorithms are declared in the class algorithmUserServer.
- The method **getAlgorithm()** (p. 122) returns an **algorithmVI** (p. 18) called name from the protected server of algorithms.
- The method **addAlgorithm()** (p. 122) add an algorithm into the server.
- In addition, the run("name") will execute the run() method of the algorith name. The algorithms are stores into a protected server (**serverVI** (p. 152)) of **algorithmVI** (p. 18).
- It has a second protected server (**serverVI** (p. 152)) of **eventProcess** (p. 68). An **eventProcess** (p. 68) is an **eventComposite** (p. 66) algorithm with services provided by **serviceI** (p. 156). These services allows an external user (**optionManager** (p. 109)) to create new **eventProcess** (p. 68) or add algorithms to the generation, reconstruction and analysis task of a given **eventProcess** (p. 68). There is not public access to **eventProcess** (p. 68), the only acces is via the getAlgorithm(name) method.
- It has a third protected server (**serverVI** (p. 152)) of **algorithmTask** (p. 12). An **algorithmTask** (p. 12) is an **algorithmComposite** (p. 10) with services **optionVI** (p. 114) and **messageVI** (p. 99) provided by the **serviceI** (p. 156) base class. These services allow the possibility of an external user to add algorithms to the **algorithmComposite** (p. 10) (see **algorithmTask** (p. 12) for more information). There is not public access to **algorithmTask** (p. 12), the only access is via **getAlgorithm**() (p. 122) method.
- The *Centella* implementation of processManager has two already created **algorithmTask** (p. 12) named "*iniOfRun*" and "*endOfRun*", which should contain algorithms to be run at the start and end of the run.
- This implementation also has a main **eventProcess** (p. 68) called "*runEvent*" where the **eventProcess** (p. 68) defined by the user can be add to. The "runEvent" is the process called by the **runEventAlg** (p. 140) algorithm that executes one event.
- Note that either **eventProcess** (p. 68) and **algorithmTask** (p. 12) are **algorithmVI** (p. 18) their selves. So the getAlgorithm(name) or run(name) methods will search on their server and return their **algorithmVI** (p. 18) base.

Definition at line 50 of file processManager.h.

4.47.2 Constructor & Destructor Documentation

4.47.2.1 processManager::processManager ()

constructor - creates the algorithmUserServer.

Definition at line 9 of file processManager.cpp.

4.47.2.2 processManager::~processManager ()

destructor - delete the algorithmUserServer.

add a new **algorithmTask** (p. 12) into the server

Definition at line 60 of file processManager.cpp.

4.47.3 Member Function Documentation

4.47.3.1 void processManager::addAlgorithm (std::string *name*, algorithmVI $*$ *alg*) `[inline]`

add algorithms into the server.

Definition at line 69 of file processManager.h.

4.47.3.2 void processManager::defineOption () `[inline, protected, virtual]`

define the options.

Reimplemented from **optionVI** (p. 116).

Definition at line 92 of file processManager.cpp.

4.47.3.3 algorithmVI $*$ processManager::getAlgorithm (std::string *name*) const

returns an algorithm with an name.

Definition at line 65 of file processManager.cpp.

4.47.3.4 algorithmTask $*$ processManager::getAlgorithmTask (std::string *name*) const `[protected]`

returns a **algorithmTask** (p. 12) with a name.

Definition at line 83 of file processManager.cpp.

4.47.3.5 eventProcess $*$ processManager::getEventProcess (std::string *name*) const `[protected]`

returns a **eventProcess** (p. 68) with a name.

Definition at line 75 of file processManager.cpp.

4.47.3.6 processManager $*$ processManager::instance () `[inline, static]`

Singleton, pointer to the class.

Definition at line 64 of file processManager.h.

4.47.3.7 void processManager::setOption (std::string *type*, std::string *name*) `[inline, protected, virtual]`

create and include the evtProcess and endRunProcess that the User wants to execute.

Always create a "endOfRun" **algorithmTask** (p. 12) for execute the end of run algorithms

Reimplemented from **optionVI** (p. 117).

Definition at line 43 of file processManager.cpp.

4.47.3.8 void processManager::writeOut () const `[inline, virtual]`

write the information of this class.

Reimplemented from **serviceI** (p. 158).

Definition at line 99 of file processManager.cpp.

4.47.4 Friends And Related Function Documentation

4.47.4.1 class algorithmTask `[friend]`

Definition at line 54 of file processManager.h.

4.47.4.2 class eventProcess `[friend]`

Definition at line 53 of file processManager.h.

4.47.5 Member Data Documentation

4.47.5.1 std::string processManager::m_algorithmTaskName `[private]`

dummy string to **optionVI** (p. 114).

Definition at line 108 of file processManager.h.

4.47.5.2 serverVI<algorithmTask>* processManager::m_algorithmTasks `[private]`

pointer to the server of endRunProcess.

Definition at line 96 of file processManager.h.

4.47.5.3 serverVI<algorithmVI>* processManager::m_algorithms `[private]`

pointer to the server of algorithmUserServer.

Definition at line 92 of file processManager.h.

4.47.5.4 algorithmTask * processManager::m_endRun `[private]`

algorithmTask (p. 12) for the end of the run.

Definition at line 101 of file processManager.h.

4.47.5.5 std::string processManager::m_eventProcessName `[private]`

dummy string to **optionVI** (p. 114).

Definition at line 106 of file processManager.h.

4.47.5.6 serverVI<eventProcess>* processManager::m_eventProcesses `[private]`

pointer to the evtProcess Server.

Definition at line 94 of file processManager.h.

4.47.5.7 algorithmTask * processManager::m_iniRun `[private]`

algorithmTask (p. 12) for the initialization of the run.

Definition at line 99 of file processManager.h.

4.47.5.8 processManager * processManager::m_instance = 0 `[static, private]`

Singleton: pointer to itself.

Definition at line 89 of file processManager.h.

4.47.5.9 eventProcess * processManager::m_runEvent `[private]`

algorithmTask (p. 12) for process an event.

Definition at line 103 of file processManager.h.

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

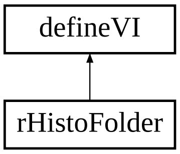
- **processManager.h**
- **processManager.cpp**

4.48 rHistoFolder Class Reference

base class to define, store and fill a group of root histograms and ntuples.

`#include <rHistoFolder.h>`

Inheritance diagram for rHistoFolder:



Public Methods

- **rHistoFolder** ()

  constructor.

- ~**rHistoFolder** ()

  destructor.

- TFile∗ **getFile** () const

  return the pointer to the file where the histogram/ntuples are going to be stored.

- virtual void **define** () = 0

  pure virtual method defines (book) histograms and ntuples.

- virtual void **fill** () = 0

  A pure virtual method to fill the histograms and ntuples.

Protected Methods

- virtual void **setFile** (TFile∗ file)

  set the root-file where the histogram/ntuples are stores.

Private Attributes

- TFile∗ **m_RFile**

  pointer to the output root-file.

Friends

- class **rHistoFolderServer**

4.48.1 Detailed Description

base class to define, store and fill a group of root histograms and ntuples.

- An TFile should be indicated to every HistoFolder.
- The **define()** (p. 126) method should BOOK the histograms and ntuples
- The `fille()` method should fill the histograms and ntuples

Definition at line 20 of file rHistoFolder.h.

4.48.2 Constructor & Destructor Documentation

4.48.2.1 rHistoFolder::rHistoFolder () `[inline]`

constructor.

a "name" is asociated witch every "rHistoFolder" object

Definition at line 31 of file rHistoFolder.h.

4.48.2.2 rHistoFolder::~rHistoFolder () `[inline]`

destructor.

Definition at line 33 of file rHistoFolder.h.

4.48.3 Member Function Documentation

4.48.3.1 void rHistoFolder::define () `[inline, pure virtual]`

pure virtual method defines (book) histograms and ntuples.

Reimplemented from **defineVI** (p. 65).

4.48.3.2 void rHistoFolder::fill () `[inline, pure virtual]`

A pure virtual method to fill the histograms and ntuples.

4.48.3.3 TFile * rHistoFolder::getFile () const `[inline]`

return the pointer to the file where the histogram/ntuples are going to be stored.

Definition at line 36 of file rHistoFolder.h.

4.48.3.4 void rHistoFolder::setFile (TFile * *file*) `[inline, protected, virtual]`

set the root-file where the histogram/ntuples are stores.

It allows derived classes to especify the root-file

Definition at line 47 of file rHistoFolder.h.

4.48.4 Friends And Related Function Documentation

4.48.4.1 class rHistoFolderServer `[friend]`

Definition at line 25 of file rHistoFolder.h.

4.48.5 Member Data Documentation

4.48.5.1 TFile ∗ rHistoFolder::m_RFile `[private]`

pointer to the output root-file.

Definition at line 52 of file rHistoFolder.h.

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

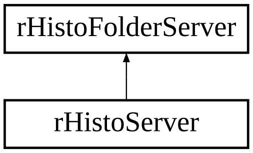
- **rHistoFolder.h**

4.49 rHistoFolderServer Class Reference

It handle a server with **rHistoFolder** (p. 125).

`#include <rHistoFolderServer.h>`

Inheritance diagram for rHistoFolderServer:



Public Methods

- **rHistoFolderServer** ()

  constructor.

- **~rHistoFolderServer** ()

  destructor.

- void **define** (std::string name)

  execute **define()** (p. 129) *the* **rHistoFolder** (p. 125) *with name.*

- void **fill** (std::string name)

  execute **fill()** (p. 129) *in the* **rHistoFolder** (p. 125) *with name.*

Protected Methods

- **rHistoFolder∗ getHistoFolder** (std::string name) const

  returns the **rHistoFolder** (p. 125) *named with "name".*

- virtual void **defineHistoFolders** () = 0

  pure virtual method. The user should define the folders.

- virtual void **makeAlgorithms** ()

  converts the user defined rHistoFolders into algorithms.

- virtual void **setFile** (TFile∗ file)

  *set the root-file where histos/ntuple are writen for all "***rHistoFolder** (p. 125)*" in the Server.*

Private Attributes

- **serverVI<rHistoFolder>∗ m_server**

  serverVI (p. 152) *of the "***rHistoFolder** (p. 125)*" in the Server.*

4.49.1 Detailed Description

It handle a server with **rHistoFolder** (p. 125).

- It provides access to a rHistofolder by its name (method `getHistoFolder()`).
- The **define()** (p. 129) and **fill()** (p. 129) methods are executed in a named **rHistoFolder** (p. 125).
- The user should implement their **rHistoFolder** (p. 125) for different task and added into the constructor of this server via the **defineHistoFolders()** (p. 129).
- In the **rHistoServer** (p. 131), an algorithm fill+rHistoServer_name will be created automatically. This is made calling **makeAlgorithms()** (p. 130) from the constructor. The constructor also call **defineHistoFolders()** (p. 129).

Definition at line 22 of file rHistoFolderServer.h.

4.49.2 Constructor & Destructor Documentation

4.49.2.1 rHistoFolderServer::rHistoFolderServer ()

constructor.

Definition at line 8 of file rHistoFolderServer.cpp.

4.49.2.2 rHistoFolderServer::~rHistoFolderServer ()

destructor.

Definition at line 17 of file rHistoFolderServer.cpp.

4.49.3 Member Function Documentation

4.49.3.1 void rHistoFolderServer::define (std::string *name*) `[inline]`

execute **define()** (p. 129) the **rHistoFolder** (p. 125) with name.

Definition at line 33 of file rHistoFolderServer.h.

4.49.3.2 void rHistoFolderServer::defineHistoFolders () `[inline, protected, pure virtual]`

pure virtual method. The user should define the folders.

Reimplemented in **rHistoServer** (p. 132).

4.49.3.3 void rHistoFolderServer::fill (std::string *name*) `[inline]`

execute **fill()** (p. 129) in the **rHistoFolder** (p. 125) with name.

Definition at line 35 of file rHistoFolderServer.h.

4.49.3.4 rHistoFolder ∗ rHistoFolderServer::getHistoFolder (std::string *name*) const `[protected]`

returns the **rHistoFolder** (p. 125) named with "name".

Definition at line 33 of file rHistoFolderServer.cpp.

4.49.3.5 void rHistoFolderServer::makeAlgorithms () `[inline, protected, virtual]`

converts the user defined rHistoFolders into algorithms.

Definition at line 22 of file rHistoFolderServer.cpp.

4.49.3.6 void rHistoFolderServer::setFile (TFile ∗ *file*) `[inline, protected, virtual]`

set the root-file where histos/ntuple are writen for all "**rHistoFolder** (p. 125)" in the Server.

Definition at line 41 of file rHistoFolderServer.cpp.

4.49.4 Member Data Documentation

4.49.4.1 serverVI<rHistoFolder>∗ rHistoFolderServer::m_server `[private]`

serverVI (p. 152) of the "**rHistoFolder** (p. 125)" in the Server.

Definition at line 54 of file rHistoFolderServer.h.

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

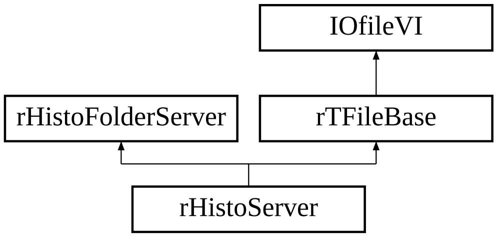
- **rHistoFolderServer.h**
- **rHistoFolderServer.cpp**

4.50 rHistoServer Class Reference

User implementation of histogram/ntuples.

`#include <rHistoServer.h>`

Inheritance diagram for rHistoServer:



Protected Methods

- **rHistoServer** ()

  generic constructor.

- ~**rHistoServer** ()

  destructor.

- virtual void **defineHistoFolders** () = 0

  the user should define the folders.

- virtual void **open** ()

  Open a root-file with a given name.

4.50.1 Detailed Description

User implementation of histogram/ntuples.

a HistoServer has all **IOfileVI** (p. 84) operations defined via **rTFileBase** (p. 133) a HistoServer handles a collection or **rHistoFolder** (p. 125) via the **rHistoFolderServer** (p. 128).

- It modifies the **`open()`** (p. 132) of **rTFileBase** (p. 133) to set the TFile into the rHistoFolders of the **rHistoFolderServer** (p. 128).
- The User should declare all the rHistoFolders in the `defineHistoFolder()` method.

Definition at line 24 of file rHistoServer.h.

4.50.2 Constructor & Destructor Documentation

4.50.2.1 rHistoServer::rHistoServer () `[protected]`

generic constructor.

A list of User defined "**rHistoFolder** (p. 125)" are created and stored

4.50.2.2 rHistoServer::~rHistoServer () `[protected]`

destructor.

4.50.3 Member Function Documentation

4.50.3.1 void rHistoServer::defineHistoFolders () `[inline, protected, pure virtual]`

the user should define the folders.

Reimplemented from **rHistoFolderServer** (p. 129).

4.50.3.2 void rHistoServer::open () `[inline, protected, virtual]`

Open a root-file with a given name.

overwriten to set the TFile into the rHistoFolders

Reimplemented from **rTFileBase** (p. 134).

Definition at line 4 of file rHistoServer.cpp.

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

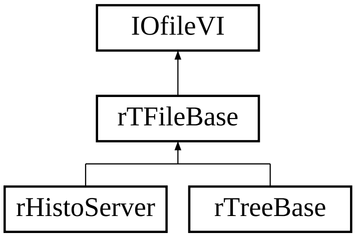
- **rHistoServer.h**
- **rHistoServer.cpp**

4.51 rTFileBase Class Reference

It provides a interface to open/close and access ROOT files (TFile).

`#include <rTFileBase.h>`

Inheritance diagram for rTFileBase:



Public Methods

- **rTFileBase** ()

  constructor.

- ~**rTFileBase** ()

  destructor.

- virtual void **open** ()

  open a file "filename" in Read/Write mode.

- virtual void **close** ()

  close the root-file.

- virtual void **clear** ()

  clear the the member variables.

- TFile∗ **getFile** ()

  return the pointer to the root-file.

Protected Attributes

- TFile∗ **m_RFile**

  pointer to the ROOT file.

4.51.1 Detailed Description

It provides a interface to open/close and access ROOT files (TFile).

The classes that used a ROOT file are derived from rTFileBase

Definition at line 13 of file rTFileBase.h.

4.51.2 Constructor & Destructor Documentation

4.51.2.1 rTFileBase::rTFileBase () `[inline]`

constructor.

Definition at line 19 of file rTFileBase.h.

4.51.2.2 rTFileBase::~rTFileBase () `[inline]`

destructor.

Definition at line 21 of file rTFileBase.h.

4.51.3 Member Function Documentation

4.51.3.1 void rTFileBase::clear () `[inline, virtual]`

clear the the member variables.

Reimplemented from **IOfileVI** (p. 86).

Reimplemented in **rTreeBase** (p. 137).

Definition at line 32 of file rTFileBase.cpp.

4.51.3.2 void rTFileBase::close () `[inline, virtual]`

close the root-file.

Reimplemented from **IOfileVI** (p. 86).

Definition at line 22 of file rTFileBase.cpp.

4.51.3.3 TFile * rTFileBase::getFile () `[inline]`

return the pointer to the root-file.

To active this ROOT file the derived methods sould call "**getFile**() (p. 134)->cd();"

Definition at line 32 of file rTFileBase.h.

4.51.3.4 void rTFileBase::open () `[inline, virtual]`

open a file "filename" in Read/Write mode.

Reimplemented from **IOfileVI** (p. 87).

Reimplemented in **rHistoServer** (p. 132), and **rTreeBase** (p. 138).

Definition at line 6 of file rTFileBase.cpp.

4.51.4 Member Data Documentation

4.51.4.1 TFile $*$ rTFileBase::m_RFile `[protected]`

pointer to the ROOT file.

Definition at line 37 of file rTFileBase.h.

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

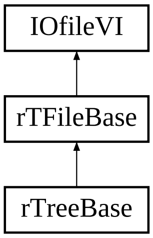
- **rTFileBase.h**
- **rTFileBase.cpp**

4.52 rTreeBase Class Reference

Provide an interface to lead/create and use a ROOT-Tree.

`#include <rTreeBase.h>`

Inheritance diagram for rTreeBase:



Public Methods

- **rTreeBase** (rCore* r, char* name)

  constructor: it need the main branch class "rEvent" template.

- **~rTreeBase** ()

  destructor.

- virtual void **open** ()

  open a (read or write) root-tree file and (load or create) the tree.

- virtual void **clear** ()

  clear the member variables.

- rCore* **core** ()

  returns the main branch class.

- char* **className** ()
- void **writeEvent** ()

  write current event from the main branch into the root-tree.

- void **readEvent** ()

  read current event data from the root-tree into the main branch.

- void **skipEvent** ()

  skip some events when reading.

Protected Attributes

- TTree* **m_RTree**

  pointer to the root-tree.

- rCore* **m_core**

main branch class of the tree (template to be instanciated by the user).

- UInt_t **bufsize**

  root file buffer size.

- Int_t **split**

  root file split level.

- char* **m_class**

4.52.1 Detailed Description

template<class rCore> class rTreeBase

Provide an interface to lead/create and use a ROOT-Tree.

- It provides the method to `read()`, `write()` from/to a ROOT-tree.
- The Tree should have only one main branch, the main branch is defined using the template "rCore".
- That class should have a **className()** (p. 137) method with the name of the class (needed by ROOT).
- The ROOT file is handle by the base class "**rTFileBase** (p. 133)".

Definition at line 21 of file rTreeBase.h.

4.52.2 Constructor & Destructor Documentation

4.52.2.1 template<class rCore> rTreeBase<rCore>::rTreeBase<rCore> (rCore * *r*, char * *name*)

constructor: it need the main branch class "rEvent" template.

Definition at line 28 of file rTreeBase.h.

4.52.2.2 template<class rCore> rTreeBase<rCore>::~rTreeBase<rCore> () `[inline]`

destructor.

Definition at line 30 of file rTreeBase.h.

4.52.3 Member Function Documentation

4.52.3.1 template<class rCore> char * rTreeBase<rCore>::className () `[inline]`

Definition at line 41 of file rTreeBase.h.

4.52.3.2 template<class rCore> void rTreeBase<rCore>::clear () `[inline, virtual]`

clear the member variables.

Reimplemented from **rTFileBase** (p. 134).

Definition at line 62 of file rTreeBase.cpp.

4.52.3.3 template<class rCore> rCore ∗ rTreeBase<rCore>::core () `[inline]`

returns the main branch class.

Definition at line 39 of file rTreeBase.h.

4.52.3.4 template<class rCore> void rTreeBase<rCore>::open () `[inline, virtual]`

open a (read or write) root-tree file and (load or create) the tree.

It set the branchAddess of the tree to the main branch class "rEvent"

Reimplemented from **rTFileBase** (p. 134).

Definition at line 7 of file rTreeBase.cpp.

4.52.3.5 template<class rCore> void rTreeBase<rCore>::readEvent () `[virtual]`

read current event data from the root-tree into the main branch.

Reimplemented from **IOfileVI** (p. 87).

Definition at line 27 of file rTreeBase.cpp.

4.52.3.6 template<class rCore> void rTreeBase<rCore>::skipEvent () `[inline, virtual]`

skip some events when reading.

Reimplemented from **IOfileVI** (p. 88).

Definition at line 49 of file rTreeBase.h.

4.52.3.7 template<class rCore> void rTreeBase<rCore>::writeEvent () `[virtual]`

write current event from the main branch into the root-tree.

Reimplemented from **IOfileVI** (p. 88).

Definition at line 45 of file rTreeBase.cpp.

4.52.4 Member Data Documentation

4.52.4.1 template<class rCore> UInt_t rTreeBase<rCore>::bufsize `[protected]`

root file buffer size.

Definition at line 60 of file rTreeBase.h.

4.52.4.2 template<class rCore> TTree ∗ rTreeBase<rCore>::m_RTree `[protected]`

pointer to the root-tree.

Definition at line 54 of file rTreeBase.h.

4.52.4.3 template<class rCore> char ∗ rTreeBase<rCore>::m_class `[protected]`

Definition at line 64 of file rTreeBase.h.

4.52.4.4 template<class rCore> rCore ∗ rTreeBase<rCore>::m_core `[protected]`

main branch class of the tree (template to be instanciated by the user).

Definition at line 57 of file rTreeBase.h.

4.52.4.5 template<class rCore> Int_t rTreeBase<rCore>::split `[protected]`

root file split level.

Definition at line 62 of file rTreeBase.h.

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

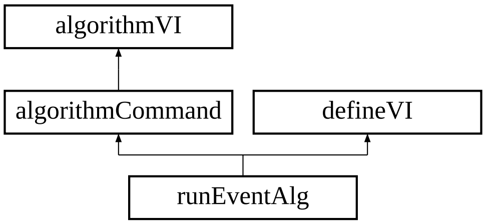
- **rTreeBase.h**
- **rTreeBase.cpp**

4.53 runEventAlg Class Reference

an algorithm to runs all the algorithms of one Event.

`#include <runEventAlg.h>`

Inheritance diagram for runEventAlg:



Public Methods

- **runEventAlg** ()
  default constructor.
- **~runEventAlg** ()
  defautl destrutor.
- virtual void **execute** ()
  execute the algorithm.
- virtual void **define** ()
  define the algorithm using other algorithms.

Private Attributes

- **cutVI* m_cReadEvent**
  cutVI (p. 49) *should we read the next event?*
- **cutVI* m_cWriteEvent**
  cutVI (p. 49) *should we write this event?;.*
- **algorithmVI* m_aReadEvent**
  algorithmVI (p. 18) *to read the event;.*
- **algorithmVI* m_aProcessEvent**
  algorithmVI (p. 18) *to process the event;.*
- **algorithmVI* m_aWriteEvent**
  algorithmVI (p. 18) *to write and event;.*
- **algorithmVI* m_aSkipEvent**
  algorithmVI (p. 18) *to skip and event.*

4.53.1 Detailed Description

an algorithm to runs all the algorithms of one Event.

- It is an **algorithmVI** (p. 18). Runs all the process for one event.
- The logic of the algorithm is implemented in the **execute()** (p. 141) method.
- It has two selection: "*selRead*" and "*selAna*" that indicates if the event should be read and if the event should be put in persistency after been processed.
- It also has the **eventProcess** (p. 68) algorithm called "runEvent" that has in fact all the list of algorithms to be executed in the event.

Definition at line 23 of file runEventAlg.h.

4.53.2 Constructor & Destructor Documentation

4.53.2.1 runEventAlg::runEventAlg () `[inline]`

default constructor.

Definition at line 29 of file runEventAlg.h.

4.53.2.2 runEventAlg::~runEventAlg () `[inline]`

defautl destrutor.

Definition at line 31 of file runEventAlg.h.

4.53.3 Member Function Documentation

4.53.3.1 void runEventAlg::define () `[inline, virtual]`

define the algorithm using other algorithms.

Reimplemented from **defineVI** (p. 65).

Definition at line 26 of file runEventAlg.cpp.

4.53.3.2 void runEventAlg::execute () `[inline, virtual]`

execute the algorithm.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 8 of file runEventAlg.cpp.

4.53.4 Member Data Documentation

4.53.4.1 algorithmVI * runEventAlg::m_aProcessEvent `[private]`

algorithmVI (p. 18) to process the event;.

Definition at line 47 of file runEventAlg.h.

4.53.4.2 algorithmVI * runEventAlg::m_aReadEvent `[private]`

algorithmVI (p. 18) to read the event;.

Definition at line 45 of file runEventAlg.h.

4.53.4.3 algorithmVI * runEventAlg::m_aSkipEvent `[private]`

algorithmVI (p. 18) to skip and event.

Definition at line 51 of file runEventAlg.h.

4.53.4.4 algorithmVI * runEventAlg::m_aWriteEvent `[private]`

algorithmVI (p. 18) to write and event;.

Definition at line 49 of file runEventAlg.h.

4.53.4.5 cutVI * runEventAlg::m_cReadEvent `[private]`

cutVI (p. 49) should we read the next event?

Definition at line 41 of file runEventAlg.h.

4.53.4.6 cutVI * runEventAlg::m_cWriteEvent `[private]`

cutVI (p. 49) should we write this event?;.

Definition at line 43 of file runEventAlg.h.

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

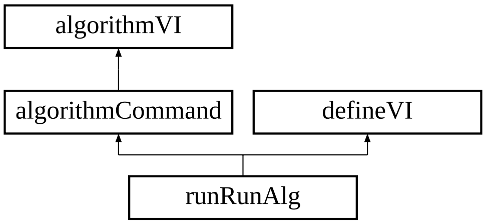
- **runEventAlg.h**
- **runEventAlg.cpp**

4.54 runRunAlg Class Reference

algorithm to run a RUN.

`#include <runRunAlg.h>`

Inheritance diagram for runRunAlg:



Public Methods

- **runRunAlg** ()

  default constructor.

- ~**runRunAlg** ()

  defautl destrutor.

- virtual void **execute** ()

  execute the algorithm.

- virtual void **define** ()

  define the algorithm using other algorithms.

Private Attributes

- **cutVI* m_cNextEvent**

  keep a pointer to the cuts and algorithms needed for this algorithm **cutVI** (p. 49) *is next event in persistency?*

- **algorithmVI* m_aRunEvent**

  algorithmVI (p. 18) *to run an event event;.*

4.54.1 Detailed Description

algorithm to run a RUN.

runRunAlg runs the **runEventAlg** (p. 140) in all the events in the persistent store.

Definition at line 16 of file runRunAlg.h.

4.54.2 Constructor & Destructor Documentation

4.54.2.1 runRunAlg::runRunAlg () `[inline]`

default constructor.

Definition at line 22 of file runRunAlg.h.

4.54.2.2 runRunAlg::~runRunAlg () `[inline]`

defautl destrutor.

Definition at line 24 of file runRunAlg.h.

4.54.3 Member Function Documentation

4.54.3.1 void runRunAlg::define () `[inline, virtual]`

define the algorithm using other algorithms.

Reimplemented from **defineVI** (p. 65).

Definition at line 19 of file runRunAlg.cpp.

4.54.3.2 void runRunAlg::execute () `[inline, virtual]`

execute the algorithm.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 8 of file runRunAlg.cpp.

4.54.4 Member Data Documentation

4.54.4.1 algorithmVI * runRunAlg::m_aRunEvent `[private]`

algorithmVI (p. 18) to run an event event;.

Definition at line 38 of file runRunAlg.h.

4.54.4.2 cutVI * runRunAlg::m_cNextEvent `[private]`

keep a pointer to the cuts and algorithms needed for this algorithm **cutVI** (p. 49) is next event in persistency?

Definition at line 36 of file runRunAlg.h.

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

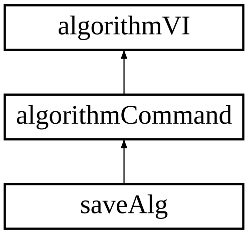
- **runRunAlg.h**
- **runRunAlg.cpp**

4.55 saveAlg Class Reference

transform a converter from transient to persistent into an algorithm.

`#include <saveAlg.h>`

Inheritance diagram for saveAlg:



Public Methods

- **saveAlg** (**converterVI**∗ conv)

  constructor - set the converter.

- ∼**saveAlg** ()

  destructor.

- void **execute** ()

  execute import() method.

Private Attributes

- **converterVI**∗ **m_conv**

  pointer to the converter.

4.55.1 Detailed Description

transform a converter from transient to persistent into an algorithm.

Definition at line 11 of file saveAlg.h.

4.55.2 Constructor & Destructor Documentation

4.55.2.1 saveAlg::saveAlg (converterVI ∗ *conv*)

constructor - set the converter.

Definition at line 16 of file saveAlg.h.

4.55.2.2 saveAlg::~saveAlg () `[inline]`

destructor.

Definition at line 18 of file saveAlg.h.

4.55.3 Member Function Documentation

4.55.3.1 void saveAlg::execute () `[inline, virtual]`

execute import() method.

Reimplemented from **algorithmVI** (p. 19).

Definition at line 21 of file saveAlg.h.

4.55.4 Member Data Documentation

4.55.4.1 converterVI * saveAlg::m_conv `[private]`

pointer to the converter.

Definition at line 26 of file saveAlg.h.

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

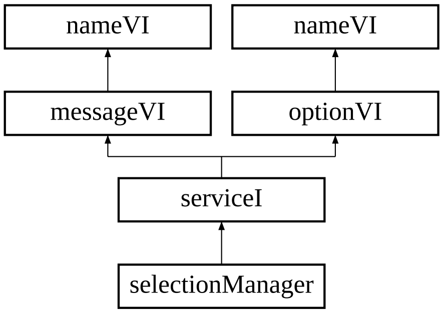
- **saveAlg.h**

4.56 selectionManager Class Reference

selection Manager.

`#include <selectionManager.h>`

Inheritance diagram for selectionManager:



Public Methods

- **selectionManager** ()

  default constructor - constructs the cutUserServer.

- **~selectionManager** ()

  destructor.

- bool **apply** (std::string name)

  apply the named **cutVI** (p. 49)*.*

- void **addCut** (std::string name, **cutVI**∗ cut)

  add a cut into the server.

- **cutVI**∗ **getCut** (std::string name) const

  returns a named **cutVI** (p. 49)*.*

- **cutComposite**∗ **getCutComposite** (std::string name) const

  returns a named **cutComposite** (p. 44)*.*

- virtual void **writeOut** () const

  info of the manager.

Static Public Methods

- selectionManager∗ **instance** ()

  Singleton: returns the Manager.

Protected Methods

- **cutSelection∗ getSelection** (std::string name) const

  return the selectionServer.

- virtual void **defineOption** ()

  define the options **optionVI** (p. 114)*.*

- virtual void **setOption** (std::string selectionName, std::string cutName)

  *the User define the selections (***cutComposite** (p. 44)*) via* **optionVI** (p. 114)*.*

Private Attributes

- **serverVI<cutVI>∗ m_serverCut**

  Pointer to the user defined cut in the Server.

- **serverVI<cutSelection>∗ m_serverSelection**

  Pointer to the selections in the Server.

- **cutSelection∗ m_selRead**

  Read event **cutSelection** (p. 46) *pointer.*

- **cutSelection∗ m_selAna**

  Analysis **cutSelection** (p. 46) *pointer.*

- std::string **m_selName**

  dummy string for the options.

Static Private Attributes

- selectionManager∗ **m_instance** = 0

  Singleton: pointer to the class.

4.56.1 Detailed Description

selection Manager.

selectionManager stores the system and user defined cuts.

selectionManager returns the cuts and **cutComposite** (p. 44) via the **`getCut()`** (p. 150) and **`getCutComposite()`** (p. 150) methods

- the system and user cuts should be defined in the cutUserServer class.
- the method `getCut(nameCut)` returns the a cut (**cutVI** (p. 49)) with name nameCut from the server of the selectionManager. These cuts are either simple cuts (**cutVI** (p. 49)) or multiple cuts (**cutComposite** (p. 44)). In addition, the method `apply(nameCut)` returns the result of applying the cut named nameCut.

- the method `getCutComposite(nameCuts)` returns the **cutComposite** (p. 44) with name nameCuts. the **cutComposite** (p. 44) is a multicut that allows access to everyone of the cuts by name or index. (see **cutComposite** (p. 44) for details). A **cutComposite** (p. 44) is also a **cutVI** (p. 49), the different of calling getCut(myCut) or getCutComposite(myCut), being myCut an implementation of **cutComposite** (p. 44), it that the 1st one only allow to execute the method **apply**() (p. 149), where the second one will allow to access the individual cuts on myCut.
- selectionManager has a protected server with **cutSelection** (p. 46). cutSelections are just a **cutComposite** (p. 44) with services declared in **serviceI** (p. 156) (**optionVI** (p. 114) and **messageVI** (p. 99)). cutSelections have a object name to refer to them, they hace the possibility of adding cuts into the **cutComposite** (p. 44), that implies that an user can run-time define a **cutSelection** (p. 46) (that is a **cutVI** (p. 49) or **cutComposite** (p. 44)) by adding particular cuts. There is not public access for **getSelection**() (p. 150), but when the `getComposite(name)` method is executed the manager searchs into the private **cutSelection** (p. 46) server.
- via the options an external user (**optionManager** (p. 109)) can define a new **cutSelection** (p. 46) and from there add cuts into the cut Selection.
- The *Centella* implementation of selectionServer have two already created **cutSelection** (p. 46): they are the "*selRead*" and the "*selAna*" selections. The "*selRead*" selection contains the selection criteria or cuts required to the event in order to be read it from persistency and the event algorithms (**runEventAlg** (p. 140)) executed. the "*selAna*" contains the selection criteria that a processed event should fulfill in order to be written into persistency.

Definition at line 53 of file selectionManager.h.

4.56.2 Constructor & Destructor Documentation

4.56.2.1 selectionManager::selectionManager ()

default constructor - constructs the cutUserServer.

Definition at line 12 of file selectionManager.cpp.

4.56.2.2 selectionManager::~selectionManager ()

destructor.

create always two selections. selRead to select the events to read selAna to select the events to analyse

Definition at line 40 of file selectionManager.cpp.

4.56.3 Member Function Documentation

4.56.3.1 void selectionManager::addCut (std::string *name*, cutVI $*$ *cut*) `[inline]`

add a cut into the server.

Definition at line 71 of file selectionManager.h.

4.56.3.2 bool selectionManager::apply (std::string *name*) `[inline]`

apply the named **cutVI** (p. 49).

Definition at line 68 of file selectionManager.h.

4.56.3.3 void selectionManager::defineOption () `[inline, protected, virtual]`

define the options **optionVI** (p. 114).

Reimplemented from **optionVI** (p. 116).

Definition at line 84 of file selectionManager.cpp.

4.56.3.4 cutVI ∗ selectionManager::getCut (std::string *name*) const

returns a named **cutVI** (p. 49).

Definition at line 47 of file selectionManager.cpp.

4.56.3.5 cutComposite ∗ selectionManager::getCutComposite (std::string *name*) const

returns a named **cutComposite** (p. 44).

Definition at line 56 of file selectionManager.cpp.

4.56.3.6 cutSelection ∗ selectionManager::getSelection (std::string *name*) const `[protected]`

return the selectionServer.

Definition at line 64 of file selectionManager.cpp.

4.56.3.7 selectionManager ∗ selectionManager::instance () `[inline, static]`

Singleton: returns the Manager.

Definition at line 65 of file selectionManager.h.

4.56.3.8 void selectionManager::setOption (std::string *selectionName*, std::string *cutName*) `[inline, protected, virtual]`

the User define the selections (**cutComposite** (p. 44)) via **optionVI** (p. 114).

Reimplemented from **optionVI** (p. 117).

Definition at line 72 of file selectionManager.cpp.

4.56.3.9 void selectionManager::writeOut () const `[inline, virtual]`

info of the manager.

Reimplemented from **serviceI** (p. 158).

Definition at line 90 of file selectionManager.cpp.

4.56.4 Member Data Documentation

4.56.4.1 selectionManager $*$ selectionManager::m_instance = 0 `[static, private]`

Singleton: pointer to the class.

Definition at line 96 of file selectionManager.h.

4.56.4.2 cutSelection $*$ selectionManager::m_selAna `[private]`

Analysis **cutSelection** (p. 46) pointer.

Definition at line 109 of file selectionManager.h.

4.56.4.3 std::string selectionManager::m_selName `[private]`

dummy string for the options.

Definition at line 112 of file selectionManager.h.

4.56.4.4 cutSelection $*$ selectionManager::m_selRead `[private]`

Read event **cutSelection** (p. 46) pointer.

Definition at line 107 of file selectionManager.h.

4.56.4.5 serverVI<cutVI>$*$ selectionManager::m_serverCut `[private]`

Pointer to the user defined cut in the Server.

Definition at line 99 of file selectionManager.h.

4.56.4.6 serverVI<cutSelection>$*$ selectionManager::m_serverSelection `[private]`

Pointer to the selections in the Server.

Definition at line 102 of file selectionManager.h.

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

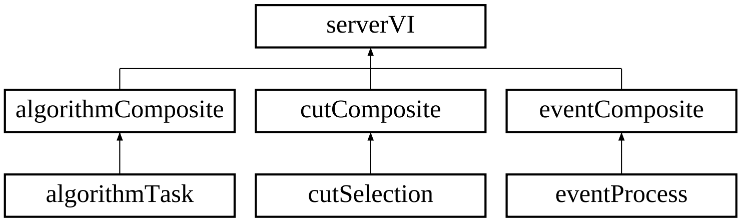
- **selectionManager.h**
- **selectionManager.cpp**

4.57 serverVI Class Reference

Server class: stores objects accesible by their name.

`#include <serverVI.h>`

Inheritance diagram for serverVI:



Public Methods

- **serverVI** ()

  default constructor.

- ~**serverVI** ()

  defualt destructor.

- void **add** (std::string n, T∗ t)

  add a new element, with its name and pointer to the template class.

- int **size** () const

  returns number of elements in to server.

- int **num** () const

  returns number of elements in the server.

- void **clear** ()

  add element no entries (no deletion of pointers).

- bool **search** (std::string name) const

  returns true if the object with name is in the server.

- std::string **getName** (T∗ t) const

  returns the name of the object, (otherwise "EMPTY").

- std::string **getName** (int i) const

  returns the name of the i object (otherwise "EMPTY").

- T∗ **get** (std::string name) const

  returns the pointer to the object with name.

- T∗ **get** (int i) const

  returns the pointer to the i element.

- std::string **nameList** () const

  returns a string with the list of all the names in the server.

- void **command** (std::string com) const

  executed a command com defined in the template class to all the elements in server.

Private Attributes

- std::vector<std::string> **m_names**
- std::vector<T*> **m_List**

4.57.1 Detailed Description

template<class T> class serverVI

Server class: stores objects accesible by their name.

- the server store POINTERS to the templates. It uses the method **add()** (p. 153);
- the objects are accesible by their names.
- every object in the server should have a unique name.
- it has the possibility of executing object method viapassing then with the **command()** (p. 154) method.
- Not possibility (yet) of removing objects from the server. Do not delete object that are in the server if the server is not destroyed or not used.
- future plan: use a map<string,T> class as base class.

Definition at line 21 of file serverVI.h.

4.57.2 Constructor & Destructor Documentation

4.57.2.1 template<class T> serverVI<T>::serverVI<T> () `[inline]`

default constructor.

Definition at line 28 of file serverVI.h.

4.57.2.2 template<class T> serverVI<T>::~serverVI<T> () `[inline]`

defualt destructor.

Definition at line 30 of file serverVI.h.

4.57.3 Member Function Documentation

4.57.3.1 template<class T> void serverVI<T>::add (std::string *n*, T * *t*) `[inline]`

add a new element, with its name and pointer to the template class.

Definition at line 33 of file serverVI.h.

4.57.3.2 template<class T> void serverVI<T>::clear () `[inline]`

add element no entries (no deletion of pointers).

Definition at line 48 of file serverVI.h.

4.57.3.3 template<class T> void serverVI<T>::command (std::string *com*) const `[inline]`

executed a command com defined in the template class to all the elements in server.

Definition at line 88 of file serverVI.h.

4.57.3.4 template<class T> T * serverVI<T>::get (int *i*) const `[inline]`

returns the pointer to the i element.

Definition at line 78 of file serverVI.h.

4.57.3.5 template<class T> T * serverVI<T>::get (std::string *name*) const `[inline]`

returns the pointer to the object with name.

Definition at line 70 of file serverVI.h.

4.57.3.6 template<class T> std::string serverVI<T>::getName (int *i*) const `[inline]`

returns the name of the i object (otherwise "EMPTY").

Definition at line 67 of file serverVI.h.

4.57.3.7 template<class T> std::string serverVI<T>::getName (T * *t*) const `[inline]`

returns the name of the object, (otherwise "EMPTY").

Definition at line 59 of file serverVI.h.

4.57.3.8 template<class T> std::string serverVI<T>::nameList () const `[inline]`

returns a string with the list of all the names in the server.

Definition at line 81 of file serverVI.h.

4.57.3.9 template<class T> int serverVI<T>::num () const `[inline]`

returns number of elements in the server.

Definition at line 44 of file serverVI.h.

4.57.3.10 template<class T> bool serverVI<T>::search (std::string *name*) const `[inline]`

returns true if the object with name is in the server.

Definition at line 51 of file serverVI.h.

4.57.3.11 template<class T> int serverVI<T>::size () const `[inline]`

returns number of elements in to server.

Definition at line 42 of file serverVI.h.

4.57.4 Member Data Documentation

4.57.4.1 template<class T> std::vector<T *> serverVI<T>::m_List `[private]`

Definition at line 95 of file serverVI.h.

4.57.4.2 template<class T> std::vector<std::string> serverVI<T>::m_names `[private]`

Definition at line 94 of file serverVI.h.

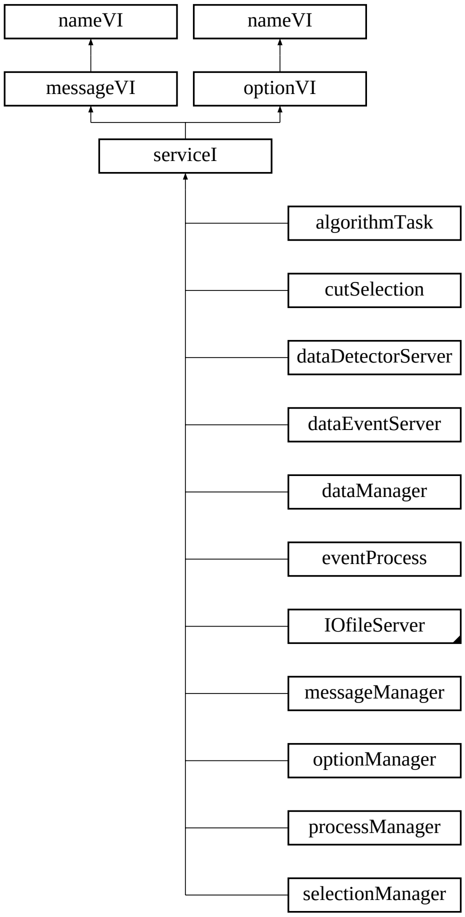
The documentation for this class was generated from the following file:

- **serverVI.h**

4.58 serviceI Class Reference

`#include <serviceI.h>`

Inheritance diagram for serviceI:



Public Methods

- **serviceI** ()

  constructor.

- ~**serviceI** ()

  default destructor.

- void **setName** (std::string name)

  Call this method to declare the class in the servers of **messageManager** (p. 93) *and* **optionManager** (p. 109)*.*

- virtual std::string **name** () const

  resolve the comflict between names in **optionVI** (p. 114) *and* **messageVI** (p. 99) *in favor of* **optionVI** (p. 114)*.*

- virtual void **defineMessage** ()

  define the message level of the serviceI classes.

- virtual void **writeOut** () const

  writeOut contains the option of the class.

Protected Methods

- void **writeOutOption** () const

  writeOut the defined options on serviceI.

4.58.1 Constructor & Destructor Documentation

4.58.1.1 serviceI::serviceI () `[inline]`

constructor.

the derived class can uses the services when it declares itself via the defineService method. There is no pure virtual methods to ovewrite. But the options should be defined overwriten the defineOption method and the messages using the defineMessage method. Both methods are called via the defineService method.

Definition at line 20 of file serviceI.h.

4.58.1.2 serviceI::~serviceI () `[inline]`

default destructor.

Definition at line 22 of file serviceI.h.

4.58.2 Member Function Documentation

4.58.2.1 void serviceI::defineMessage () `[inline, virtual]`

define the message level of the serviceI classes.

Reimplemented from **messageVI** (p. 101).

Reimplemented in **messageManager** (p. 96).

Definition at line 15 of file serviceI.cpp.

4.58.2.2 std::string serviceI::name () const `[inline, virtual]`

resolve the comflict between names in **optionVI** (p. 114) and **messageVI** (p. 99) in favor of **optionVI** (p. 114).

Reimplemented from **nameVI** (p. 105).

Definition at line 28 of file serviceI.h.

4.58.2.3 void serviceI::setName (std::string *name*) `[virtual]`

Call this method to declare the class in the servers of **messageManager** (p. 93) and **optionManager** (p. 109).

Reimplemented from **messageVI** (p. 102).

Definition at line 5 of file serviceI.cpp.

4.58.2.4 void serviceI::writeOut () const `[inline, virtual]`

writeOut contains the option of the class.

Reimplemented from **messageVI** (p. 102).

Reimplemented in **algorithmTask** (p. 13), **cutSelection** (p. 47), **dataDetectorServer** (p. 53), **dataEventServer** (p. 55), **eventProcess** (p. 69), **messageManager** (p. 97), **optionManager** (p. 112), **processManager** (p. 122), and **selectionManager** (p. 150).

Definition at line 21 of file serviceI.cpp.

4.58.2.5 void serviceI::writeOutOption () const `[protected]`

writeOut the defined options on serviceI.

Definition at line 29 of file serviceI.cpp.

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

- **serviceI.h**
- **serviceI.cpp**

4.59 trsDataVI Class Reference

bare class for the transient data.

`#include <trsDataVI.h>`

Public Methods

- **trsDataVI** ()
  default constructor.
- ~**trsDataVI** ()
  default destructor.
- virtual void **update** ()
  virtual method to clear and make/fill the transient data.
- virtual void **ini** () = 0
  pure virtual method to initialize the data (pointers if needed).
- virtual void **clear** () = 0
  pure virtual method to clear the data.
- virtual void **make** () = 0
  pure virtual method to make/fill the data.
- virtual void **writeOut** () const
  virtual function to writeOut the contents of the derived class.
- virtual void **command** (std::string com)
  virtual method to execute method of this class for composites.

4.59.1 Detailed Description

bare class for the transient data.

The transient data of the program should inherit from this base class.

- trsDataVI has a main virtual method **update**() (p. 160) that executes the pure virtual methods **clear**() (p. 160) and **make**() (p. 160). The **clear**() (p. 160) and **make**() (p. 160) should be implemented by the derived class. In principle, **clear**() (p. 160) clear the data and **make**() (p. 160) makes or fill the data. This method is usually null and replaced by the action of an **algorithmVI** (p. 18).
- There is an pure virtual method **ini**() (p. 160) to initialize the data. Most of the data is created only one, the pointer kept in a **serverVI** (p. 152), and the contents of the data replaced every **update**() (p. 160).
- For convenience we add a virtual method **writeOut**() (p. 160) to print out the information of the data. To be compatible with the **messageVI** (p. 99) **writeOut**() (p. 160) method.
- The **command**() (p. 160) method executed the method of this class via their names. It is useful to create composite classes. It is used for example in the template class **serverVI** (p. 152).

Definition at line 31 of file trsDataVI.h.

4.59.2 Constructor & Destructor Documentation

4.59.2.1 trsDataVI::trsDataVI () `[inline]`

default constructor.

Definition at line 39 of file trsDataVI.h.

4.59.2.2 trsDataVI::~trsDataVI () `[inline]`

default destructor.

Definition at line 41 of file trsDataVI.h.

4.59.3 Member Function Documentation

4.59.3.1 void trsDataVI::clear () `[inline, pure virtual]`

pure virtual method to clear the data.

4.59.3.2 void trsDataVI::command (std::string *com*) `[inline, virtual]`

virtual method to execute method of this class for composites.

It can be use for example in **serverVI** (p. 152) <trsData> classes

Definition at line 11 of file trsDataVI.cpp.

4.59.3.3 void trsDataVI::ini () `[inline, pure virtual]`

pure virtual method to initialize the data (pointers if needed).

4.59.3.4 void trsDataVI::make () `[inline, pure virtual]`

pure virtual method to make/fill the data.

usually this method is replaced by the action of an **algorithmVI** (p. 18)

4.59.3.5 void trsDataVI::update () `[inline, virtual]`

virtual method to clear and make/fill the transient data.

Definition at line 4 of file trsDataVI.cpp.

4.59.3.6 void trsDataVI::writeOut () const `[inline, virtual]`

virtual function to writeOut the contents of the derived class.

note that has the same name that the method in **messageVI** (p. 99)

Definition at line 56 of file trsDataVI.h.

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

- **trsDataVI.h**
- **trsDataVI.cpp**

4.60 userAlgorithms Class Reference

Server with the **system** and **user** declared algorithms.

`#include <userAlgorithms.h>`

Public Methods

- **userAlgorithms** ()

  constructor - creates the algorithms.

- **~userAlgorithms** ()

  destructor - destroies the algorithms.

4.60.1 Detailed Description

Server with the **system** and **user** declared algorithms.

All the static algorithms served by the **processManager** (p. 119) should be included here.

There are two **system** and **user** static algorithms.

The algorithms should exits untill the end of the program. They are not implicetely destroyed.

Definition at line 11 of file userAlgorithms.h.

4.60.2 Constructor & Destructor Documentation

4.60.2.1 userAlgorithms::userAlgorithms () `[inline]`

constructor - creates the algorithms.

Definition at line 17 of file userAlgorithms.h.

4.60.2.2 userAlgorithms::~userAlgorithms () `[inline]`

destructor - destroies the algorithms.

Definition at line 19 of file userAlgorithms.h.

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

- **userAlgorithms.h**

4.61 userCuts Class Reference

Class where the USER define his/her cuts.

`#include <userCuts.h>`

Public Methods

- **userCuts** ()
  contructors - the user define the cuts.
- **~userCuts** ()

4.61.1 Detailed Description

Class where the USER define his/her cuts.

Definition at line 7 of file userCuts.h.

4.61.2 Constructor & Destructor Documentation

4.61.2.1 userCuts::userCuts () `[inline]`

contructors - the user define the cuts.

see as reference **centellaCuts** (p. 26)

Definition at line 13 of file userCuts.h.

4.61.2.2 userCuts::~userCuts () `[inline]`

Definition at line 14 of file userCuts.h.

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

- **userCuts.h**

4.62 userDataDetector Class Reference

`#include <userDataDetector.h>`

Public Methods

- **userDataDetector** ()
- ~**userDataDetector** ()

4.62.1 Constructor & Destructor Documentation

4.62.1.1 userDataDetector::userDataDetector () `[inline]`

Definition at line 10 of file userDataDetector.h.

4.62.1.2 userDataDetector::~userDataDetector () `[inline]`

Definition at line 11 of file userDataDetector.h.

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

- **userDataDetector.h**

4.63 userDataEvent Class Reference

User defined event transient data.

`#include <userDataEvent.h>`

Public Methods

- **userDataEvent** ()

  constructor - the user should include the event data.

- **~userDataEvent** ()

  destructor.

4.63.1 Detailed Description

User defined event transient data.

Definition at line 6 of file userDataEvent.h.

4.63.2 Constructor & Destructor Documentation

4.63.2.1 userDataEvent::userDataEvent () `[inline]`

constructor - the user should include the event data.

Definition at line 11 of file userDataEvent.h.

4.63.2.2 userDataEvent::~userDataEvent () `[inline]`

destructor.

Definition at line 13 of file userDataEvent.h.

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

- **userDataEvent.h**

Chapter 5

Centella FrameWrok File Documentation

5.1 algorithmCommand.h File Reference

`#include "Event/algorithmVI.h"`

Compounds

- class **algorithmCommand**

5.2 algorithmComposite.cpp File Reference

5.3 algorithmComposite.h File Reference

```
#include "Event/algorithmVI.h"
#include "Event/serverVI.h"
```

Compounds

- class **algorithmComposite**

5.4 algorithmTask.cpp File Reference

`#include "Event/algorithmTask.h"`

`#include "Event/processManager.h"`

`#include "Event/messageManager.h"`

5.5 algorithmTask.h File Reference

```
#include "Event/algorithmComposite.h"
#include "Event/serviceI.h"
```

Compounds

- class **algorithmTask**

5.6 algorithmTree.cpp File Reference

```
#include "Event/algorithmTree.h"
```

5.7 algorithmTree.h File Reference

`#include "Event/algorithmVI.h"`

`#include "Event/algorithmComposite.h"`

Compounds

- class **algorithmTree**

5.8 algorithmVI.cpp File Reference

```
#include "Event/algorithmVI.h"
```

5.9 algorithmVI.h File Reference

`#include <string>`

Compounds

- class **algorithmVI**

5.10 centella.cpp File Reference

```
#include "Event/centella.h"
#include "Event/processManager.h"
#include "Event/dataManager.h"
#include "Event/dataIOROOTServer.h"
#include "Event/messageManager.h"
#include "Event/selectionManager.h"
#include "Event/optionManager.h"
```

5.11 centella.h File Reference

`#include "Event/algorithmTree.h"`

`#include "Event/defineVI.h"`

Compounds

- class **centella**

5.12 centellaAlgorithms.cpp File Reference

```
#include "Event/centellaAlgorithms.h"
#include "Event/processManager.h"
#include "Event/centellaCommands.h"
#include "Event/runEventAlg.h"
#include "Event/runRunAlg.h"
```

5.13 centellaAlgorithms.h File Reference

Compounds

- class **centellaAlgorithms**

5.14 centellaCommands.cpp File Reference

```
#include "Event/centellaCommands.h"
#include "Event/dataManager.h"
#include "Event/dataDetectorServer.h"
#include "Event/IOfileServer.h"
#include "Event/optionManager.h"
#include "Event/messageManager.h"
```

5.15 centellaCommands.h File Reference

```
#include "Event/algorithmCommand.h"
```

```
#include "Event/cutVI.h"
```

Compounds

- class **comApplyOptions**
- class **comCloseIO**
- class **comListOptions**
- class **comLoadCalibration**
- class **comOpenIO**
- class **comReadEvent**
- class **comSkipEvent**
- class **comWriteEvent**
- class **comWriteOutData**
- class **comWriteOutDetector**
- class **comWriteOutInfo**
- class **comWriteWelcome**
- class **conNextEvent**

5.16 centellaCuts.cpp File Reference

```
#include "Event/centellaCuts.h"
#include "Event/selectionManager.h"
#include "Event/centellaCommands.h"
```

5.17 centellaCuts.h File Reference

`#include "Event/cutVI.h"`

`#include "Event/serverVI.h"`

Compounds

- class **centellaCuts**

5.18 converterServer.cpp File Reference

```
#include "Event/converterServer.h"
#include "Event/processManager.h"
#include "Event/messageManager.h"
```

5.19 converterServer.h File Reference

```
#include "Event/serverVI.h"
#include "Event/converterVI.h"
#include "Event/loadAlg.h"
#include "Event/saveAlg.h"
```

Compounds

- class **converterServer**

5.20 converterVI.h File Reference

Compounds

- class **converterVI**

5.21 cutComposite.cpp File Reference

`#include "Event/cutComposite.h"`

5.22 cutComposite.h File Reference

`#include "Event/cutVI.h"`

`#include "Event/serverVI.h"`

Compounds

- class **cutComposite**

5.23 cutSelection.cpp File Reference

`#include "Event/cutSelection.h"`

`#include "Event/selectionManager.h"`

`#include "Event/messageManager.h"`

5.24 cutSelection.h File Reference

`#include "Event/cutComposite.h"`

`#include "Event/serviceI.h"`

Compounds

- class **cutSelection**

5.25 cutVI.h File Reference

Compounds

- class **cutVI**

5.26 dataDetectorServer.cpp File Reference

`#include "Event/dataDetectorServer.h"`

`#include "Event/messageManager.h"`

5.27 dataDetectorServer.h File Reference

```
#include "Event/trsDataVI.h"
#include "Event/serverVI.h"
#include "Event/serviceI.h"
```

Compounds

- class **dataDetectorServer**

5.28 dataEventServer.cpp File Reference

```
#include "Event/dataEventServer.h"
#include "Event/userDataEvent.h"
#include "Event/messageManager.h"
```

5.29 dataEventServer.h File Reference

```
#include "Event/trsDataVI.h"
#include "Event/serverVI.h"
#include "Event/serviceI.h"
```

Compounds

- class **dataEventServer**

Defines

- #define **dataEventServer_H** 1

5.29.1 Define Documentation

5.29.1.1 #define dataEventServer_H 1

Definition at line 8 of file dataEventServer.h.

5.30 dataIOROOTServer.cxx File Reference

```
#include "Event/dataIOROOTServer.h"
#include "Event/nullIOfile.h"
#include "Event/rTreeBase.cpp"
#include "RootTree/rHistoAnaServer.h"
#include "RootTree/rTreeReconServer.h"
#include "RootTree/Recon.h"
```

5.31 dataIOROOTServer.h File Reference

`#include "Event/IOfileServer.h"`

Compounds

- class **dataIOROOTServer**

Typedefs

- typedef rTreeReconServer **rTreeInServer**
- typedef rTreeReconServer **rTreeOutServer**
- typedef rHistoAnaServer **rHistoOutServer**

5.31.1 Typedef Documentation

5.31.1.1 typedef rHistoAnaServer rHistoOutServer

Definition at line 11 of file dataIOROOTServer.h.

5.31.1.2 typedef rTreeReconServer rTreeInServer

Definition at line 9 of file dataIOROOTServer.h.

5.31.1.3 typedef rTreeReconServer rTreeOutServer

Definition at line 10 of file dataIOROOTServer.h.

5.32 dataManager.cpp File Reference

`#include "Event/dataManager.h"`

`#include "Event/dataEventServer.h"`

`#include "Event/dataDetectorServer.h"`

`#include "Event/dataIOROOTServer.h"`

5.33 dataManager.h File Reference

```
#include <vector>
#include <string>
#include <iostream>
#include "Event/dataEventServer.h"
#include "Event/dataDetectorServer.h"
#include "Event/dataIOROOTServer.h"
```

Compounds

- class **dataManager**

Typedefs

- typedef **dataIOROOTServer dataIOServer**

  It stores and handles the Data.

5.33.1 Typedef Documentation

5.33.1.1 typedef dataIOROOTServer dataIOServer

It stores and handles the Data.

- It has a server with the detector data (dataUserServerDet) that contains the transient data for the calibration and the pointers to the geometrical data of the detectors.
- It has a second server with the event data (dataUserServerEvent) that contains a **serverVI** (p. 152) with the transient data of the event (data filled every event).
- It has a third server with the IO persistent data (dataUserServerROOT).
- The transiend data can be recovered using the `getData()` method.

Definition at line 31 of file dataManager.h.

5.34 defineVI.h File Reference

Compounds

- class **defineVI**

5.35 eventComposite.h File Reference

`#include "Event/eventVI.h"`

`#include "Event/serverVI.h"`

Compounds

- class **eventComposite**

5.36 eventProcess.cpp File Reference

```
#include "Event/eventProcess.h"
#include "Event/processManager.h"
#include "Event/messageManager.h"
```

5.37 eventProcess.h File Reference

```
#include "Event/eventComposite.h"
#include "Event/serviceI.h"
```

Compounds

- class **eventProcess**

5.38 eventVI.cpp File Reference

```
#include "Event/eventVI.h"
```

5.39 eventVI.h File Reference

`#include "Event/algorithmVI.h"`

`#include "Event/algorithmTree.h"`

Compounds

- class **eventVI**

5.40 fillAlg.h File Reference

```
#include "Event/algorithmCommand.h"
#include "Event/rHistoFolder.h"
```

Compounds

- class **fillAlg**

5.41 IOBase.h File Reference

Compounds

- class **IOBase**

5.42 IOfileServer.cpp File Reference

`#include "Event/IOfileServer.h"`

`#include "Event/messageManager.h"`

`#include "Event/optionManager.h"`

5.43 IOfileServer.h File Reference

`#include <vector>`

`#include <string>`

`#include "Event/serviceI.h"`

`#include "Event/serverVI.h"`

`#include "Event/IOBase.h"`

`#include "Event/IOfileVI.h"`

Compounds

- class **IOfileServer**

5.44 IOfileVI.cpp File Reference

```
#include "Event/IOfileVI.h"
```

5.45 IOfileVI.h File Reference

`#include "Event/IOBase.h"`

`#include <string>`

Compounds

- class **IOfileVI**

5.46 loadAlg.h File Reference

`#include "Event/algorithmCommand.h"`

`#include "Event/converterVI.h"`

Compounds

- class **loadAlg**

5.47 main.cpp File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include "TROOT.h"
#include "Event/centella.h"
#include "Event/optionManager.h"
```

Functions

- void **main** (int argc, char* argv[ ])

5.47.1 Function Documentation

5.47.1.1 void main (int *argc*, char * *argv*[ ])

Definition at line 15 of file main.cpp.

5.48 messageBase.cpp File Reference

```
#include "Event/messageBase.h"
```

5.49 messageBase.h File Reference

`#include <string>`

Compounds

- class **messageBase**

5.50 messageManager.cpp File Reference

`#include "Event/messageManager.h"`

`#include "Event/optionManager.h"`

5.51 messageManager.h File Reference

`#include <string>`

`#include <stdlib.h>`

`#include <iostream>`

`#include <fstream>`

`#include "Event/serviceI.h"`

Compounds

- class **messageManager**

Defines

- #define **messageMANAGER_H** 1

5.51.1 Define Documentation

5.51.1.1 #define messageMANAGER_H 1

Definition at line 3 of file messageManager.h.

5.52 messageVI.cpp File Reference

```
#include "Event/messageVI.h"
#include "Event/messageManager.h"
```

5.53 messageVI.h File Reference

```
#include "Event/nameVI.h"
```

```
#include "Event/messageBase.h"
```

Compounds

- class **messageVI**

5.54 nameVI.h File Reference

`#include <string>`

Compounds

- class **nameVI**

5.55 nullIOfile.h File Reference

`#include "Event/IOfileVI.h"`

Compounds

- class **nullIOfile**

5.56 optionManager.cpp File Reference

```
#include "Event/optionManager.h"
#include "Event/messageManager.h"
```

5.57 optionManager.h File Reference

`#include "Event/serviceI.h"`

`#include "Event/serverVI.h"`

Compounds

- class **optionManager**

5.58 optionVI.cpp File Reference

```
#include "Event/optionVI.h"
#include "Event/optionManager.h"
#include "Event/messageManager.h"
```

5.59 optionVI.h File Reference

`#include "Event/nameVI.h"`

`#include "Event/serverVI.h"`

`#include <string>`

Compounds

- class **optionVI**

5.60 processManager.cpp File Reference

```
#include "Event/processManager.h"
#include "Event/centellaAlgorithms.h"
#include "Event/userAlgorithms.h"
#include "Event/messageManager.h"
```

5.61 processManager.h File Reference

`#include "Event/eventProcess.h"`

`#include "Event/algorithmTask.h"`

`#include "Event/serviceI.h"`

`#include "Event/serverVI.h"`

`#include "Event/cutSelection.h"`

Compounds

- class **processManager**

5.62 rHistoFolder.cpp File Reference

```
#include "RootTree/rHistoFolder.h"
```

5.63 rHistoFolder.h File Reference

`#include <string>`

`#include "Event/defineVI.h"`

Compounds

- class **rHistoFolder**

5.64 rHistoFolderServer.cpp File Reference

```
#include "Event/rHistoFolderServer.h"
#include "Event/fillAlg.h"
#include "Event/processManager.h"
#include "Event/messageManager.h"
```

5.65 rHistoFolderServer.h File Reference

`#include "Event/rHistoFolder.h"`

`#include "Event/serverVI.h"`

`#include <vector>`

Compounds

- class **rHistoFolderServer**

5.66 rHistoServer.cpp File Reference

`#include "Event/rhistoServer.h"`

5.67 rHistoServer.h File Reference

`#include "Event/rTFileBase.h"`

`#include "Event/rHistoFolderServer.h"`

Compounds

- class **rHistoServer**

5.68 rTFileBase.cpp File Reference

```
#include "Event/rTFileBase.h"
#include "TFile.h"
#include "TROOT.h"
```

5.69 rTFileBase.h File Reference

`#include "Event/IOfileVI.h"`

Compounds

- class **rTFileBase**

5.70 rTreeBase.cpp File Reference

```
#include "Event/rTreeBase.h"
#include "TFile.h"
#include "TTree.h"
```

5.71 rTreeBase.h File Reference

`#include "rTFileBase.h"`

`#include "TROOT.h"`

Compounds

- class **rTreeBase**

5.72 runEventAlg.cpp File Reference

```
#include "Event/runEventAlg.h"
#include "Event/selectionManager.h"
#include "Event/processManager.h"
```

5.73 runEventAlg.h File Reference

```
#include "Event/algorithmCommand.h"
#include "Event/defineVI.h"
#include "Event/cutVI.h"
#include "Event/algorithmVI.h"
```

Compounds

- class **runEventAlg**

5.74 runRunAlg.cpp File Reference

```
#include "Event/runRunAlg.h"
#include "Event/selectionManager.h"
#include "Event/processManager.h"
```

5.75 runRunAlg.h File Reference

```
#include "Event/algorithmCommand.h"
#include "Event/defineVI.h"
#include "Event/cutVI.h"
#include "Event/algorithmVI.h"
```

Compounds

- class **runRunAlg**

5.76 saveAlg.h File Reference

`#include "Event/algorithmCommand.h"`

`#include "Event/converterVI.h"`

Compounds

- class **saveAlg**

5.77 selectionManager.cpp File Reference

```
#include "Event/selectionManager.h"
#include "Event/centellaCuts.h"
#include "Event/userCuts.h"
#include "Event/optionManager.h"
#include "Event/messageManager.h"
```

5.78 selectionManager.h File Reference

`#include "Event/cutVI.h"`

`#include "Event/cutSelection.h"`

`#include "Event/serviceI.h"`

`#include "Event/serverVI.h"`

Compounds

- class **selectionManager**

5.79 serverVI.h File Reference

`#include <vector>`

`#include <string>`

Compounds

- class **serverVI**

5.80 serviceI.cpp File Reference

```
#include "Event/serviceI.h"
#include "Event/messageManager.h"
```

5.81 serviceI.h File Reference

`#include "Event/optionVI.h"`

`#include "Event/messageVI.h"`

Compounds

- class **serviceI**

5.82 trsDataVI.cpp File Reference

`#include "Event/trsDataVI.h"`

5.83 trsDataVI.h File Reference

`#include <string>`

`#include <iostream>`

Compounds

- class **trsDataVI**

5.84 userAlgorithms.h File Reference

Compounds

- class **userAlgorithms**

5.85 userCuts.h File Reference

Compounds

- class **userCuts**

5.86 userDataDetector.h File Reference

Compounds

- class **userDataDetector**

5.87 userDataEvent.h File Reference

Compounds

- class **userDataEvent**