# AMORE++

pre-alpha (active development aiming to release a beta version this summer (2011) )

Generated by Doxygen 1.7.4

Mon Jun 20 2011 15:56:11

# Contents

Chapter 1

# The AMORE++ package

## 1.1 Introduction

Here you will find the documentation of the C++ component of the AMORE++ R package.

The AMORE++ package is a new version of the publicly available AMORE package for neural network training and simulation under R

## 1.2 Motivation

Since the release of the previous version of the AMORE many things have changed in the R programming world.

The advent of the Reference Classes and of packages like Rcpp, inline and RUnit compel us to write a better version of the package in order to provide a more useful framework for neural network training and simulation.

## 1.3 Road Map

This project is currently very active and the development team intends to provide a beta version as soon as this summer (2011)

# Chapter 2

# Class Index

## 2.1 Class Hierarchy

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

# Chapter 3

# Class Index

## 3.1 Class List

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

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all files with brief descriptions:

# Chapter 5

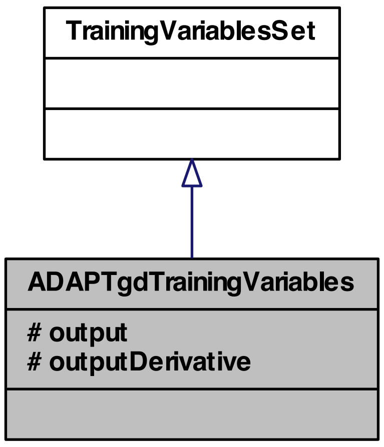
# Class Documentation

## 5.1 ADAPTgdTrainingVariables Class Reference

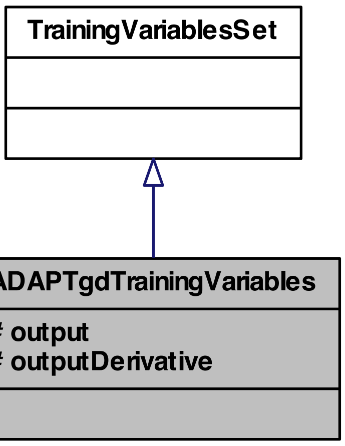
class ADAPTgdTrainingVariables -

`#include <ADAPTgdTrainingVariables.h>`

Inheritance diagram for ADAPTgdTrainingVariables:



Collaboration diagram for ADAPTgdTrainingVariables:



## Protected Attributes

- double output
- double outputDerivative

### 5.1.1 Detailed Description

class ADAPTgdTrainingVariables -

Definition at line 5 of file ADAPTgdTrainingVariables.h.

### 5.1.2 Member Data Documentation

#### 5.1.2.1 double **ADAPTgdTrainingVariables::output** `[protected]`

Definition at line 8 of file ADAPTgdTrainingVariables.h.

#### 5.1.2.2 double **ADAPTgdTrainingVariables::outputDerivative** `[protected]`

Definition at line 9 of file ADAPTgdTrainingVariables.h.

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

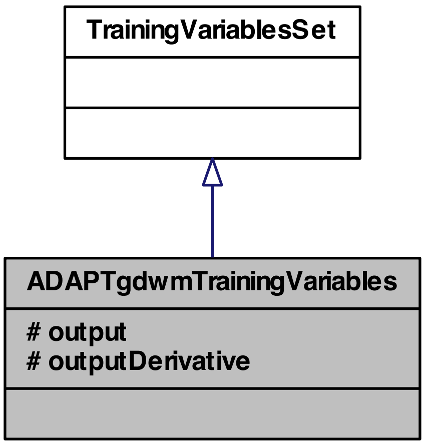
- pkg/AMORE/src/dia/ADAPTgdTrainingVariables.h

## 5.2 ADAPTgdwmTrainingVariables Class Reference

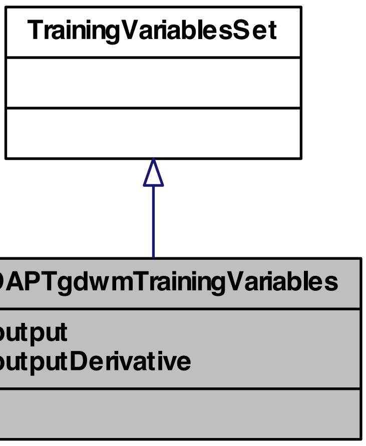
class ADAPTgdwmTrainingVariables -

`#include <ADAPTgdwmTrainingVariables.h>`

Inheritance diagram for ADAPTgdwmTrainingVariables:



Collaboration diagram for ADAPTgdwmTrainingVariables:



## Protected Attributes

- double output
- double outputDerivative

## 5.2.1 Detailed Description

class ADAPTgdwmTrainingVariables -

Definition at line 5 of file ADAPTgdwmTrainingVariables.h.

## 5.2.2 Member Data Documentation

### 5.2.2.1 double **ADAPTgdwmTrainingVariables::output** `[protected]`

Definition at line 8 of file ADAPTgdwmTrainingVariables.h.

### 5.2.2.2 double **ADAPTgdwmTrainingVariables::outputDerivative** `[protected]`

Definition at line 9 of file ADAPTgdwmTrainingVariables.h.

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

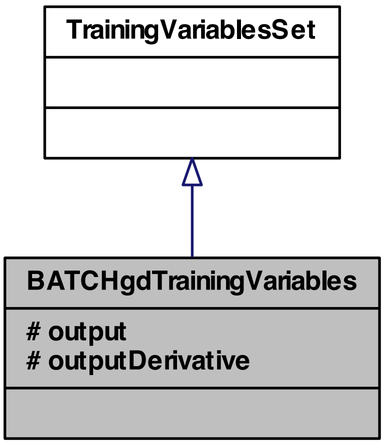
- pkg/AMORE/src/dia/ADAPTgdwmTrainingVariables.h

## 5.3 BATCHgdTrainingVariables Class Reference

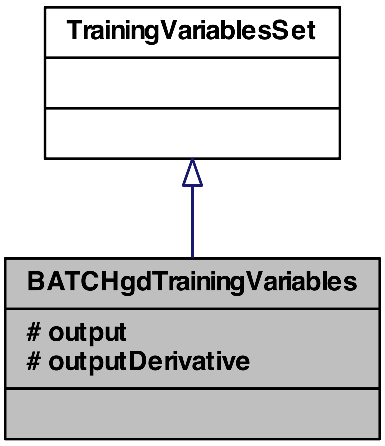
class BATCHgdTrainingVariables -

`#include <BATCHgdTrainingVariables.h>`

Inheritance diagram for BATCHgdTrainingVariables:



Collaboration diagram for BATCHgdTrainingVariables:



## Protected Attributes

- double output
- double outputDerivative

### 5.3.1 Detailed Description

class BATCHgdTrainingVariables -

Definition at line 5 of file BATCHgdTrainingVariables.h.

### 5.3.2 Member Data Documentation

#### 5.3.2.1 double BATCHgdTrainingVariables::output `[protected]`

Definition at line 8 of file BATCHgdTrainingVariables.h.

#### 5.3.2.2 double BATCHgdTrainingVariables::outputDerivative `[protected]`

Definition at line 9 of file BATCHgdTrainingVariables.h.

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

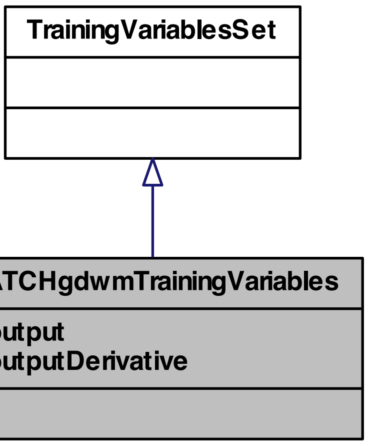
- pkg/AMORE/src/dia/BATCHgdTrainingVariables.h

## 5.4 BATCHgdwmTrainingVariables Class Reference

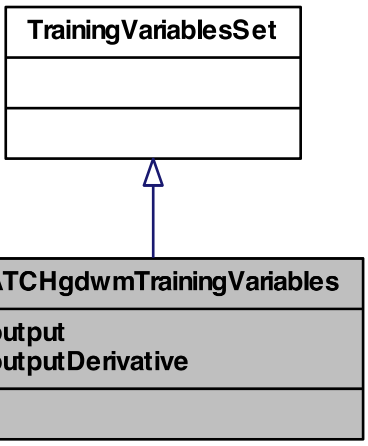
class BATCHgdwmTrainingVariables -

`#include <BATCHgdwmTrainingVariables.h>`

Inheritance diagram for BATCHgdwmTrainingVariables:



Collaboration diagram for BATCHgdwmTrainingVariables:



### Protected Attributes

- double output
- double outputDerivative

## 5.4.1 Detailed Description

class BATCHgdwmTrainingVariables -

Definition at line 5 of file BATCHgdwmTrainingVariables.h.

## 5.4.2 Member Data Documentation

### 5.4.2.1 double **BATCHgdwmTrainingVariables::output** `[protected]`

Definition at line 8 of file BATCHgdwmTrainingVariables.h.

### 5.4.2.2 double **BATCHgdwmTrainingVariables::outputDerivative** `[protected]`

Definition at line 9 of file BATCHgdwmTrainingVariables.h.

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

- pkg/AMORE/src/dia/BATCHgdwmTrainingVariables.h

## 5.5 CompareId Struct Reference

### Public Member Functions

- bool operator() (const ConPtr a, const ConPtr b)
- bool operator() (const ConPtr a, const int b)
- bool operator() (const int a, const ConPtr b)
- bool operator() (const int a, const int b)

### 5.5.1 Detailed Description

Definition at line 352 of file ConContainer.cpp.

### 5.5.2 Member Function Documentation

#### 5.5.2.1 bool CompareId::operator() ( const **ConPtr** *a*, const **ConPtr** *b* ) [inline]

Definition at line 356 of file ConContainer.cpp.

```
{
  return a->getId() < b->getId();
}
```

#### 5.5.2.2 bool CompareId::operator() ( const int *a*, const int *b* ) [inline]

Definition at line 377 of file ConContainer.cpp.

```
{
  return a < b;
}
```

#### 5.5.2.3 bool CompareId::operator() ( const int *a*, const **ConPtr** *b* ) [inline]

Definition at line 370 of file ConContainer.cpp.

```
{
  return a < b->getId();
}
```

#### 5.5.2.4 bool CompareId::operator() ( const ConPtr *a*, const int *b* ) [inline]

Definition at line 363 of file ConContainer.cpp.

```
{
  return a->getId() < b;
}
```

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

- pkg/AMORE/src/old/ConContainer.cpp

## 5.6 Con Class Reference

class Con -

`#include <Con.h>`

### Public Member Functions

- Con (Neuron &neuron)

  *Constructor.*
- Con (Neuron &neuron, double weight)

  *Constructor.*
- Neuron & neuron ()

  *from field accessor.*
- int Id ()

  *A getter of the Id of the Neuron pointed by the from field.*
- double & weight ()
- void show ()

  *Pretty print of the Con information.*
- bool validate ()

  *Object validator.*
- Con ()

  *Default Constructor.*
- Con (NeuronPtr neuronPtr)

  *Constructor.*
- Con (NeuronPtr neuronPtr, double value)

  *Constructor.*
- ∼Con ()

  *Default Destructor.*
- NeuronPtr getFrom ()

  *from field accessor.*
- void setFrom (NeuronPtr neuronPtr)

*from field accessor.*

- int getId ()

  *A getter of the Id of the Neuron pointed by the from field.*
- double getWeight ()

  *weight field accessor.*
- void setWeight (double value)

  *weight field accessor.*
- bool show ()
- bool validate ()

## Protected Attributes

- NeuronRef d_neuron
- double d_weight

## Private Attributes

- NeuronWeakPtr from

  *A smart pointer to the Neuron used as input during simulation or training.*
- double weight

  *A double variable that contains the weight of the connection.*

### 5.6.1 Detailed Description

class Con -

A class to handle the information needed to describe an input connection.

The Con class provides a simple class for a connection described by a pair of values: a pointer to a Neuron object used as the from field and the weight used to propagate the value of that Neuron object.

Definition at line 3 of file Con.h.

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 Con::Con ( **Neuron & *neuron*** )

Constructor.

Definition at line 17 of file Con.cpp.

```
                          :
  d_neuron( boost::ref(neuron) ), d_weight(0)
{
}
```

#### 5.6.2.2 Con::Con ( Neuron & *neuron*, double *weight* )

Constructor.

Definition at line 28 of file Con.cpp.

```
                                                     :
  d_neuron(boost::ref(neuron)), d_weight(weight)
{
}
```

#### 5.6.2.3 Con::Con ( )

Default Constructor.

Definition at line 17 of file Con.cpp.

```
         :
  weight(0), from()
{
}
```

#### 5.6.2.4 Con::Con ( NeuronPtr *neuronPtr* )

Constructor.

Definition at line 40 of file Con.cpp.

```
                              :
  from(neuronPtr), weight(0)
{
}
```

#### 5.6.2.5 Con::Con ( NeuronPtr *neuronPtr*, double *value* )

Constructor.

Definition at line 29 of file Con.cpp.

```
                                          :
  from(neuronPtr), weight(value)
{
}
```

#### 5.6.2.6 Con::$\sim$Con ( )

Default Destructor.

Definition at line 46 of file Con.cpp.

```
{
}
```

### 5.6.3 Member Function Documentation

#### 5.6.3.1 NeuronPtr Con::getFrom ( )

from field accessor.

This method allows access to the address stored in the private from field (a pointer to a Neuron object).*

**Returns**

A pointer to the Neuron object referred to by the from field.

```
        //================
        //Usage example:
        //================
        // Data set up
                        NeuronPtr ptShNeuron ( new Neuron(1) );          // Neuron
       Id is set 1
                        ConPtr ptShCon( new Con(ptShNeuron) );           // from p
      oints to ptShNeuron and weight is set to 0
        // Test
                        ptShNeuron = ptShCon->getFrom() ;
                        int result = ptShNeuron->getId();

        // Now, result is equal to 1.
```

**See also**

getId and the unit test files, e.g., runit.Cpp.Con.R, for further examples.

Definition at line 71 of file Con.cpp.

References from.

```
{
  return (from.lock());
}
```

#### 5.6.3.2 int Con::getId ( )

A getter of the Id of the Neuron pointed by the from field.

This method gets the Id of the Neuron referred to by the from field

**Returns**

The value of the Id (an integer).

```
        //================
        //Usage example:
        //================
        // Data set up
                        NeuronPtr ptShNeuron ( new Neuron(16) );         // Neuron
       Id is set to 16
```

```
                        ConPtr ptShCon( new Con(ptShNeuron) );          // from p
      oints to ptShNeuron and weight is set to 0
        // Test
                        int result = ptShCon->getId();

        // Now, result is equal to 16.
```

**See also**

getFrom, setFrom and the unit test files, e.g., runit.Cpp.Con.R, for further examples.

Definition at line 123 of file Con.cpp.

References from.

```
{
  if (from.use_count() > 0)
    {
      NeuronPtr neuronPtr(from);
      return (neuronPtr->getId());
    }
  else
    {
      return (NA_INTEGER);
    }
}
```

#### 5.6.3.3 double Con::getWeight ( )

weight field accessor.

This method allows access to the value stored in the private field weight

**Returns**

The value of weight (double)

```
  //================
  //Usage example:
  //================
  // Data set up
                        std::vector<double> result;
                        NeuronPtr ptShNeuron ( new Neuron(16) );                /
      / Neuron Id is set to 16
                        ConPtr ptShCon( new Con(ptShNeuron, 12.4) );  // from poi
      nts to ptShNeuron and weight is set to 12.4
        // Test
                        result.push_back( ptShCon->getWeight() );
                        ptShCon->setWeight(2.2);
                        result.push_back( ptShCon->getWeight() );

        // Now, result is a numeric vector that contains the values 12.4 and 2.2
      .
```

**See also**

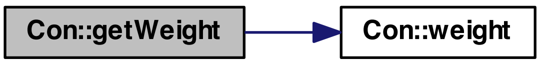
setWeight and the unit test files, e.g., runit.Cpp.Con.R, for further examples.

Definition at line 158 of file Con.cpp.

References weight().

```
{
  return (weight);
}
```

Here is the call graph for this function:



#### 5.6.3.4 int Con::Id ( )

A getter of the Id of the Neuron pointed by the from field.

This method gets the Id of the Neuron referred to by the from field

**Returns**

The value of the Id (an integer).

```
//================
//Usage example:
//================
// Data set up
                NeuronPtr ptShNeuron ( new Neuron(16) );        // Neuron I
d is set to 16
                ConPtr ptShCon( new Con(ptShNeuron) );          // from poi
nts to ptShNeuron and weight is set to 0
// Test
                int result = ptShCon->getId();

// Now, result is equal to 16.
```

**See also**

getFrom, setFrom and the unit test files, e.g., runit.Cpp.Con.R, for further examples.

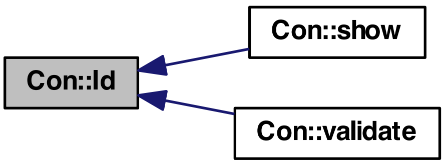
Definition at line 79 of file Con.cpp.

References d_neuron.

Referenced by show(), and validate().

```
{
  return d_neuron.get().Id();
}
```

Here is the caller graph for this function:



### 5.6.3.5 Neuron & Con::neuron ( )

from field accessor.

This method allows access to the address stored in the private from field (a pointer to a Neuron object).*

**Returns**

A pointer to the Neuron object referred to by the from field.

```
        //================
        //Usage example:
        //================
        // Data set up
                        NeuronPtr ptShNeuron ( new Neuron(1) );         // Neuron
       Id is set 1
                        ConPtr ptShCon( new Con(ptShNeuron) );          // from p
      oints to ptShNeuron and weight is set to 0
        // Test
                        ptShNeuron = ptShCon->getFrom() ;
                        int result = ptShNeuron->getId();

        // Now, result is equal to 1.
```

**See also**

getId and the unit test files, e.g., runit.Cpp.Con.R, for further examples.

Definition at line 54 of file Con.cpp.

References d_neuron.

```
{
  return d_neuron;
}
```

#### 5.6.3.6 void Con::setFrom ( NeuronPtr *neuronPtr* )

from field accessor.

This method sets the value of the from field with the address used as parameter.

**Parameters**

| | |
|---|---|
| *f* | A pointer to the neuron that is to be inserted in the from field. |

```
        //================
        //Usage example:
        //================
        // Data set up
                        NeuronPtr ptShNeuron ( new Neuron(1) );         // Neuron
       Id is set to 1
                        ConPtr ptShCon( new Con() );
                        ptShCon->setFrom( ptShNeuron );
        // Test
                        ptShNeuron = ptShCon->getFrom() ;
                        int result = ptShNeuron->getId();

        // Now, result is equal to 1
```

**See also**

getFrom and getId contain usage examples. For further examples see the unit test files, e.g., runit.Cpp.Con.R

Definition at line 98 of file Con.cpp.

References from.

```
{
  from = neuronPtr;
}
```

#### 5.6.3.7 void Con::setWeight ( double *value* )

weight field accessor.

This method sets the value of the weight field.

**Parameters**

| | |
|---|---|
| *w* | The new value (double) to be set in the weight field. |

```
  //================
  //Usage example:
```

```
  //================
  // Data set up
                      std::vector<double> result;
                      NeuronPtr ptShNeuron ( new Neuron(16) );                /
      / Neuron Id is set to 16
                      ConPtr ptShCon( new Con(ptShNeuron, 12.4) );  // from poi
      nts to ptShNeuron and weight is set to 12.4
                      result.push_back(ptShCon->getWeight());
        // Test
                      ptShCon->setWeight(2.2);
                      result.push_back(ptShCon->getWeight());

        // Now, result is a numeric vector that contains the values 12.4 and 2.2
      .
```

**See also**

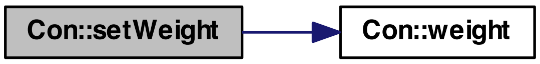
getWeight and the unit test files (e.g. runit.Cpp.Con.R)

Definition at line 186 of file Con.cpp.

References weight().

```
{
  weight = value;
}
```

Here is the call graph for this function:



#### 5.6.3.8 bool Con::show ( )

#### 5.6.3.9 bool Con::show ( )

Pretty print of the Con information.

This method outputs in the R terminal the contents of the Con fields.

**Returns**

true in case everything works without throwing an exception

**See also**

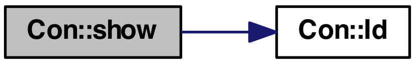
setWeight and the unit test files, e.g., runit.Cpp.Con.R, for usage examples.

Definition at line 118 of file Con.cpp.

References d_weight, and Id().

```
{
  int id = Id();
  if (id == NA_INTEGER)
    {
      Rprintf("From: NA\t Invalid Connection \n");
    }
  else
    {
      Rprintf("From:\t %d \t Weight= \t %lf \n", id, d_weight);
    }
}
```

Here is the call graph for this function:



#### 5.6.3.10 bool Con::validate ( )

Object validator.

This method checks the object for internal coherence. A try / catch mechanism exits normal execution and returns control to the R terminal in case the contents of the Con object are identified as corrupted.

**Returns**

true in case the checks are Ok.

**Exceptions**

| | |
|---|---|
| *An* | std::range error if weight or from are not finite. |

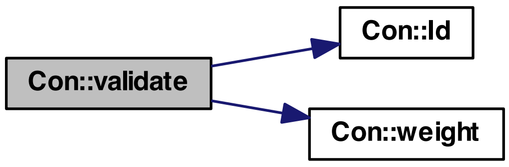
Definition at line 138 of file Con.cpp.

References Id(), and weight().

```
{
  BEGIN_RCPP
  if (! R_FINITE(weight()) ) throw std::range_error("weight is not finite.");
  if (Id() == NA_INTEGER)
```

```
    throw std::range_error("fromId is not finite.");
  return (true);
END_RCPP}
```

Here is the call graph for this function:

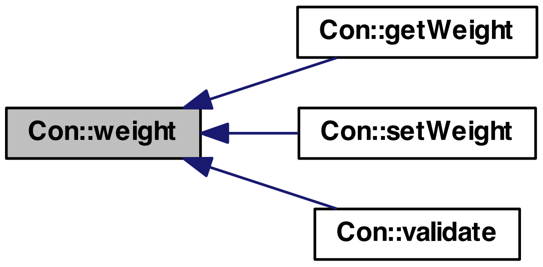


#### 5.6.3.11 bool Con::validate ( )

#### 5.6.3.12 double& Con::weight ( )

Referenced by getWeight(), setWeight(), and validate().

Here is the caller graph for this function:



### 5.6.4 Member Data Documentation

#### 5.6.4.1 NeuronRef Con::d_neuron [protected]

Definition at line 6 of file Con.h.

Referenced by Id(), and neuron().

#### 5.6.4.2 double Con::d_weight [protected]

Definition at line 7 of file Con.h.

Referenced by show().

#### 5.6.4.3 NeuronWeakPtr Con::from [private]

A smart pointer to the Neuron used as input during simulation or training.

The from field contains the address of the Neuron whose output will be used as input by the Neuron containing the Con object.

Definition at line 22 of file Con.h.

Referenced by getFrom(), getId(), and setFrom().

#### 5.6.4.4 double & Con::weight [private]

A double variable that contains the weight of the connection.

weight field accessor.

The weight field contains the factor by which the output value of the Neuron addressed by the from field is multiplied during simulation or training.

This method allows access to the value stored in the private field weight

**Returns**

The value of weight (double)

```
//================
//Usage example:
//================
// Data set up
                      std::vector<double> result;
                      NeuronPtr ptShNeuron ( new Neuron(16) );                 /
    / Neuron Id is set to 16
                      ConPtr ptShCon( new Con(ptShNeuron, 12.4) );  // from poi
    nts to ptShNeuron and weight is set to 12.4
      // Test
                      result.push_back( ptShCon->getWeight() );
                      ptShCon->setWeight(2.2);
                      result.push_back( ptShCon->getWeight() );

      // Now, result is a numeric vector that contains the values 12.4 and 2.2
    .
```

**See also**

setWeight and the unit test files, e.g., runit.Cpp.Con.R, for further examples.

Definition at line 27 of file Con.h.

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

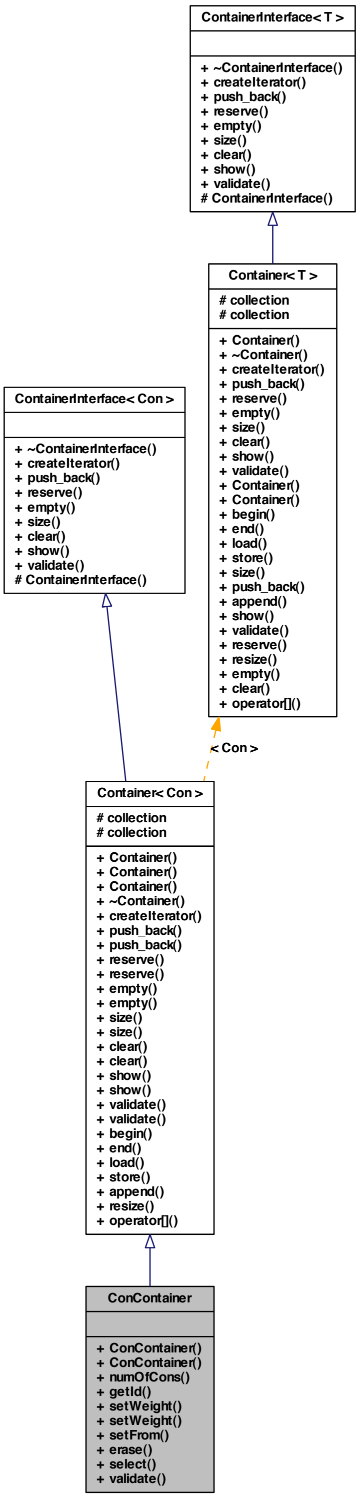
- pkg/AMORE/src/dia/Con.h
- pkg/AMORE/src/old/Con.h
- pkg/AMORE/src/Con.cpp
- pkg/AMORE/src/old/Con.cpp

## 5.7 ConContainer Class Reference

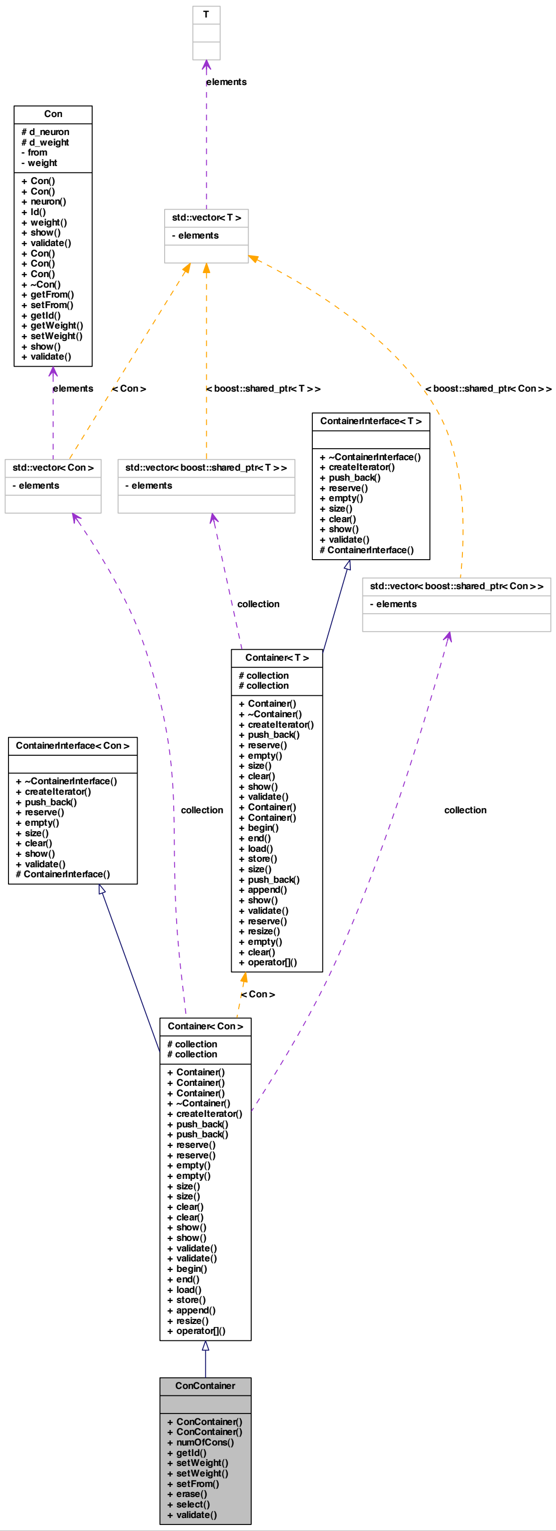
A vector of connections.

```
#include <ConContainer.h>
```

Inheritance diagram for ConContainer:



Collaboration diagram for ConContainer:



## Public Types

- typedef std::vector< boost::shared_ptr< Con > >::iterator iterator
- typedef std::vector< boost::shared_ptr< Con > >::const_iterator const_iterator
- typedef boost::shared_ptr< Con > value_type
- typedef value_type const & const_reference

## Public Member Functions

- ConContainer ()
- ConContainer (std::vector< ConPtr > collection)
- int numOfCons ()

  *Size of the ConContainer object.*
- std::vector< int > getId ()

  *Getter of the Id values of the vector of Cons.*
- bool setWeight (std::vector< double > nWeights)

  *Setter of the weight field of the Con objects related to ConContainer.*
- bool setWeight (std::vector< double > nWeights, std::vector< int > nIds)

  *Setter of the weights of the specified elements from the ConContainer object.*
- bool setFrom (NeuronContainer neuronContainer)

  *Setter of the from fields of the Con objects related to ConContainer.*
- void erase (std::vector< int > nIds)

  *Erase the specified elements from the vecCom object.*
- ConContainerPtr select (std::vector< int > nIds)

  *Selects the specified elements from the vecCom object.*
- bool validate ()

  *Object validator.*

### 5.7.1 Detailed Description

A vector of connections.

The ConContainer class provides a simple class for a vector of connections. It's named after the R equivalent Reference Class.

Definition at line 16 of file ConContainer.h.

### 5.7.2 Member Typedef Documentation

#### 5.7.2.1 typedef std::vector<boost::shared_ptr<Con> >::const_iterator ConContainer::const_iterator

Reimplemented from Container< Con >.

Definition at line 23 of file ConContainer.h.

#### 5.7.2.2 typedef value_type const& ConContainer::const_reference

Reimplemented from Container< Con >.

Definition at line 27 of file ConContainer.h.

#### 5.7.2.3 typedef std::vector<boost::shared_ptr<Con> >::iterator ConContainer::iterator

Reimplemented from Container< Con >.

Definition at line 21 of file ConContainer.h.

#### 5.7.2.4 typedef boost::shared_ptr<Con> ConContainer::value_type

Reimplemented from Container< Con >.

Definition at line 25 of file ConContainer.h.

### 5.7.3 Constructor & Destructor Documentation

#### 5.7.3.1 ConContainer::ConContainer ( )

Definition at line 8 of file ConContainer.cpp.

```
{
}
```

#### 5.7.3.2 ConContainer::ConContainer ( std::vector< ConPtr > *collection* )

Definition at line 12 of file ConContainer.cpp.

```
                                                           :
  Container<Con> (collection) // Call to Base constructor
{
}
```

### 5.7.4 Member Function Documentation

#### 5.7.4.1 void ConContainer::erase ( std::vector< int > *nIds* )

Erase the specified elements from the vecCom object.

Provides a convenient way of removing some Con objects from the collection field of the ConContainer object.

**Parameters**

| | |
|---|---|
| *vFrom* | An std::vector<int> with the Ids of the connections to remove. |

```
        //================
        //Usage example:
        //================

        // Data set up
                        std::vector<int> result;
                        std::vector<NeuronPtr> neuronContainer;
                        ConContainerPtr conContainerPtr( new ConContainer() );
                        ConContainerPtr vErased;
                        ConPtr  ptC;
                        NeuronPtr ptN;
                        int ids[]= {11, 10, 9, 3, 4, 5, 6, 7, 8, 2, 1};
                        std::vector<double> nWeights;
                        nWeights.push_back(11.32);
                        nWeights.push_back(1.26);
                        nWeights.push_back(2.14);
                        nWeights.push_back(3.16);
                        nWeights.push_back(4.14);
                        nWeights.push_back(5.19);
                        nWeights.push_back(6.18);
                        nWeights.push_back(7.16);
                        nWeights.push_back(8.14);
                        nWeights.push_back(9.12);
                        nWeights.push_back(10.31);

                        for (int i=0; i<nWeights.size() ; i++) {                                /
      / Let's create a vector with three neurons
                                ptN.reset( new Neuron( ids[i] ) );
                                neuronContainer.push_back(ptN);
                        }
                        conContainerPtr->buildAndAppend(neuronContainer, nWeights
      );

                        // Test

                        std::vector<int> toRemove;
                        toRemove.push_back(1);
                        toRemove.push_back(3);
                        toRemove.push_back(5);
                        toRemove.push_back(7);

                        conContainerPtr->erase(toRemove);
                        conContainerPtr->show();
                        result=conContainerPtr->getId();

                // The output at the R terminal would display :
                //
                // From:         2       Weight=         9.120000
                // From:         4       Weight=         4.140000
                // From:         6       Weight=         6.180000
                // From:         8       Weight=         8.140000
                // From:         9       Weight=         2.140000
                // From:         10   Weight=     1.260000
                // From:         11   Weight=     11.320000
```

**See also**

select and the unit test files, e.g. runit.Cpp.ConContainer.R, for further examples.

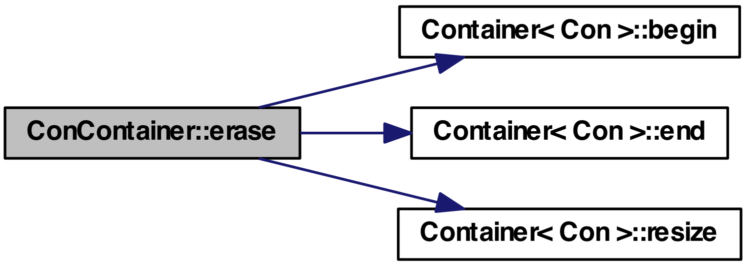
Definition at line 450 of file ConContainer.cpp.

References Container< Con >::begin(), Container< Con >::end(), and Container<

Con >::resize().

```
{
  std::vector<ConPtr>::iterator itr;
  sort(begin(), end(), CompareId());
  sort(nIds.begin(), nIds.end());
  itr = set_difference(begin(), end(), nIds.begin(), nIds.end(), begin(),
      CompareId());
  resize(itr - begin());
}
```

Here is the call graph for this function:



#### 5.7.4.2 std::vector< int > ConContainer::getId ( )

Getter of the Id values of the vector of Cons.

This function returns the Id's of the neurons referred to by the vector of Cons.

**Returns**

An std::vector<int> that contains the Ids

```
  //================
  //Usage example:
  //================
        // Data set up
                        Neuron N1, N2, N3;
                        ConContainer conContainer;
                        std::vector<int> result;

                        N1.setId(10);
                        N2.setId(20);
                        N3.setId(30);

                        ConPtr ptCon( new Con(&N1, 1.13) );     // Create new Con
```

```
       and initialize ptCon
                        conContainer.push_back(ptCon);                          /
      / push_back
                        ptCon.reset(  new Con(&N2, 2.22) );             // create
       new Con and assign to ptCon
                        conContainer.push_back(ptCon);                          /
      / push_back
                        ptCon.reset(  new Con(&N3, 3.33) );             // create
       new Con and assign to ptCon
                        conContainer.push_back(ptCon);                          /
      / push_back

        // Test
                        conContainer.show() ;
                        conContainer.validate();
                        result=conContainer.getId();

        // Now result is a vector that contains the values 10, 20 and 30.
```

**See also**

getWeight and the unit test files, e.g. runit.Cpp.ConContainer.R, for further examples.

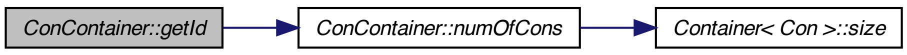
Definition at line 93 of file ConContainer.cpp.

References numOfCons().

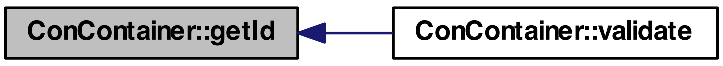
Referenced by validate().

```
{
  std::vector<int> result;
  result.reserve(numOfCons());
  foreach (ConPtr itr, *this)
    {
      result.push_back(itr->getId());
    }
  return result;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.7.4.3 int ConContainer::numOfCons ( )

Size of the ConContainer object.

This function returns the size of the ConContainer object, that is to say, the number of Con objects it contains.

**Returns**

The size of the vector

```
//================
//Usage example:
//================
      // Data set up
                        std::vector<int> result;
                        std::vector<ConPtr> vcA, vcB;
                        ContainerNeuronPtr      neuronContainerPtr( new
    Container<Neuron>() );
                        ConContainerPtr conContainerPtr( new
    ConContainer() );
                        ConPtr  ptC;
                        NeuronPtr ptN;
                        int ids[]= {10, 20, 30};
                        double weights[] = {1.13, 2.22, 3.33 };
                        for (int i=0; i<=2 ; i++) {                    /
    / Let's create a vector with three neurons
                                ptN.reset( new Neuron( ids[i] ) );
                                neuronContainerPtr->push_back(ptN);
                        }
      // Test
                        for (int i=0; i<=2 ; i++) {                    /
    / and a vector with three connections
                                result.push_back(conContainerPtr->numOfCo
    ns());          // Append numOfCons to result, create new Con and push_back into
    conContainer
                                ptC.reset( new Con( neuronContainerPtr->l
    oad().at(i), weights[i]) );
                                conContainerPtr->push_back(ptC);
                        }

      // Now, result contains a numeric vector with values 0, 1, 2, and 3.
```

**See also**

Container::size (alias)

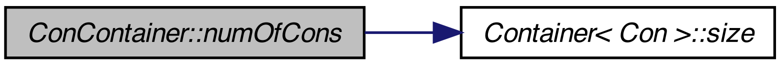
Definition at line 52 of file ConContainer.cpp.

References Container< Con >::size().

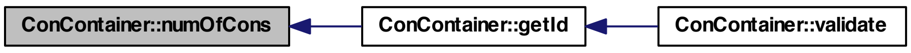
Referenced by getId().

```
{
  return size();
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.7.4.4 ConContainerPtr ConContainer::select ( std::vector< int > *nIds* )

Selects the specified elements from the vecCom object.

Provides a convenient way of selecting some Con objects from the collection field of the ConContainer object.

**Parameters**

| | |
|---|---|
| *vFrom* | An std::vector<int> with the Ids of the connections to select. |

```
        //================
        //Usage example:
```

```
        //================

        // Data set up
                std::vector<int> result;
                std::vector<NeuronPtr> neuronContainer;
                ConContainerPtr conContainerPtr( new ConContainer() );
                ConPtr  ptC;
                NeuronPtr ptN;
                int ids[]= {11, 10, 9, 3, 4, 5, 6, 7, 8, 2, 1};
                double weights[]={11.32, 1.26, 2.14, 3.16, 4.14, 5.19, 6.18, 7.16
      , 8.14, 9.12, 10.31};
                std::vector<double> nWeights;
                for (int i=0; i<11; i++) {
                        nWeights.push_back(weights[i]);
                }
                for (int i=0; i<nWeights.size() ; i++) {                                /
      / Let's create a vector with three neurons
                        ptN.reset( new Neuron( ids[i] ) );
                        neuronContainer.push_back(ptN);
                }
                conContainerPtr->buildAndAppend(neuronContainer, nWeights);
                // Test
                std::vector<int> toSelect;
                toSelect.push_back(1);
                toSelect.push_back(3);
                toSelect.push_back(5);
                toSelect.push_back(7);

                ConContainerPtr  vSelect (  conContainerPtr->select(toSelect)  );

                result=vSelect->getId();

                // Now, result is a numeric vector with the values 1, 3, 5 and 7.
```

**See also**

erase and the unit test files, e.g. runit.Cpp.ConContainer.R, for further examples.

Definition at line 505 of file ConContainer.cpp.

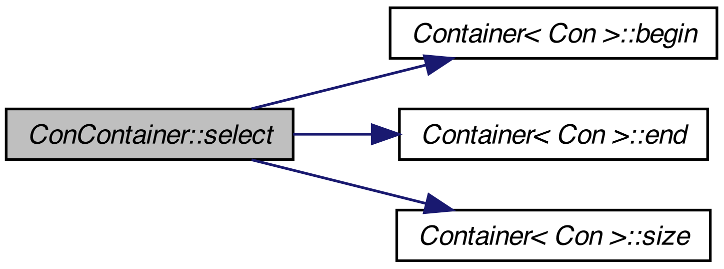
References Container< Con >::begin(), Container< Con >::end(), and Container< Con >::size().

Referenced by setWeight().

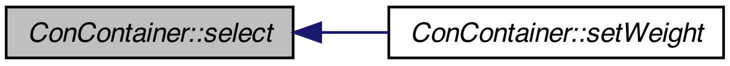
```
{
  ConContainerPtr result(new ConContainer);
  result->reserve(size());
  sort(begin(), end(), CompareId());
  sort(nIds.begin(), nIds.end());
  set_intersection(begin(), end(), nIds.begin(), nIds.end(),
      std::back_inserter(*result), CompareId());

  return result;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.7.4.5 bool ConContainer::setFrom ( **NeuronContainer** *neuronContainer* )

Setter of the from fields of the Con objects related to ConContainer.

This function provides a convenient way of getting the values of the weight field of those Con object pointed to by the smart pointer stored in the ConContainer object.

**Parameters**

| | |
|---|---|
| *vFrom* | An std::vector<NeuronPtr> with the pointers to be set in the from fields of the ConContainer object. |

**Returns**

true if not exception is thrown

```
//================
//Usage example:
        //================

        // Data set up
                std::vector<int> result;
                ContainerNeuronPtr      neuronContainerPtr( new
      Container<Neuron>() );
                ConContainerPtr conContainerPtr( new ConContainer() );
                ConPtr  ptC;
                NeuronPtr ptN;

                int ids[]= {10, 20, 30};
                double weights[] = {1.13, 2.22, 3.33 };

                for (int i=0; i<=2 ; i++) {                             // Let's
      create a vector with three neurons
                        ptN.reset( new Neuron( ids[i] ) );
                        neuronContainerPtr->push_back(ptN);
                }
                for (int i=0; i<=2 ; i++) {                             // and a
      vector with three connections
                        ptC.reset( new Con() );
                        conContainerPtr->push_back(ptC);
                }
        // Test
                conContainerPtr->setFrom(neuronContainerPtr->load()) ;
                conContainerPtr->show();
                result=conContainerPtr->getId();

        // Now result is a vector that contains the values 10, 20 and 30.
```

**See also**

getFrom and the unit test files, e.g. runit.Cpp.ConContainer.R, for further examples.

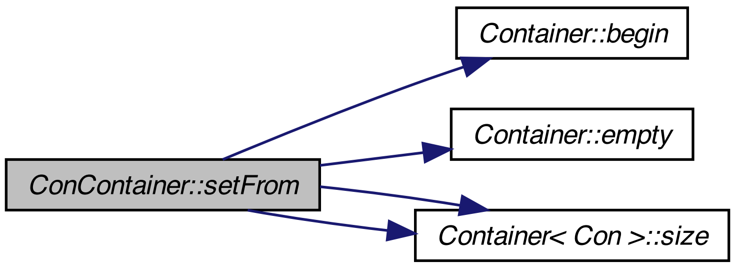
Definition at line 333 of file ConContainer.cpp.

References Container< T >::begin(), Container< T >::empty(), Container< Con >::size(), and Container< T >::size().

```
{
  BEGIN_RCPP
  if (neuronContainer.empty())
    { throw std::range_error("[ C++ ConContainer::setFrom]: Error, w is empty");}

  if (neuronContainer.size() != size())
    {
      throw std::range_error(
          "[C++ ConContainer::setFrom]: Error, neuronContainer.size() != collecti
      on.size()");
    }
  std::vector<NeuronPtr>::iterator itrNeuron = neuronContainer.begin();
  foreach(ConPtr itr , *this)
    {
      itr->setFrom( *itrNeuron );
      itrNeuron++;
    }
  return true;
END_RCPP}
```

Here is the call graph for this function:



#### 5.7.4.6 bool ConContainer::setWeight ( std::vector< double > *nWeights*, std::vector< int > *nIds* )

Setter of the weights of the specified elements from the ConContainer object.

Provides a convenient way of setting the weights of some Con objects from the collection field of the ConContainer object.

**Parameters**

| | |
|---|---|
| *nWeights* | A numeric (double) vector with the weights to be set in the Con objects contained in the ConContainer object. |
| *vFrom* | An std::vector<int> with the Ids of the connections to select |

**Returns**

true in case no exception is thrown

```
//================
//Usage example:
//================

// Data set up
        std::vector<double> result;
                std::vector<NeuronPtr> neuronContainer;
                ConContainerPtr conContainerPtr( new ConContainer() );
                ConPtr  ptC;
                NeuronPtr ptN;
                int ids[]= {11, 10, 9, 3, 4, 5, 6, 7, 8, 2, 1};
                double weights[]={11.32, 1.26, 2.14, 3.16, 4.14, 5.19, 6.
18, 7.16, 8.14, 9.12, 10.31};
                std::vector<double> nWeights;
                for (int i=0; i<11; i++) {
                nWeights.push_back(weights[i]);
```

```
                        }
                        for (int i=0; i<nWeights.size() ; i++) {
      / Let's create a vector with three neurons
                        ptN.reset( new Neuron( ids[i] ) );
                        neuronContainer.push_back(ptN);
                        }
                        conContainerPtr->buildAndAppend(neuronContainer, nWeights
      );

                        std::vector<int> toSelect;
                        std::vector<double> vNewWeights;
                        toSelect.push_back(1);
                        toSelect.push_back(3);
                        toSelect.push_back(5);
                        toSelect.push_back(7);
                        vNewWeights.push_back(1000.1);
                        vNewWeights.push_back(3000.3);
                        vNewWeights.push_back(5000.5);
                        vNewWeights.push_back(7000.7);
                        conContainerPtr->setWeight(vNewWeights, toSelect);

        // Test
                        result = conContainerPtr->getWeight();
                        return wrap(result);

        // Now, result is a numeric vector with the values  1000.10, 9.12, 3000.3
      0, 4.14, 5000.50, 6.18, 7000.70, 8.14, 2.14, 1.26 and 11.32 .
```

**See also**

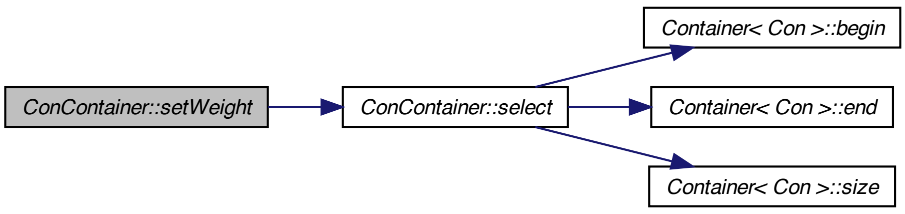
getWeigth and the unit test files, e.g. runit.Cpp.ConContainer.R, for further examples.

Definition at line 627 of file ConContainer.cpp.

References select().

```
{
BEGIN_RCPP return select(nIds)->setWeight(nWeights);
END_RCPP
}
```

Here is the call graph for this function:



#### 5.7.4.7 bool ConContainer::setWeight ( std::vector< double > *nWeights* )

Setter of the weight field of the Con objects related to ConContainer.

This function provides a convenient way of setting the values of the weight field of those Con objects pointed to by the smart pointer stored in the ConContainer object.

**Parameters**

| | |
|---|---|
| *nWeights* | A numeric (double) vector with the weights to be set in the Con objects contained in the ConContainer object. |

**Returns**

true in case no exception is thrown

```
    //================
    //Usage example:
    //================
    // Data set up
            std::vector<double> result;
                    int ids[]= {1, 2, 3};
                    double weights[] = {12.3, 1.2, 2.1 };
                    ConContainer conContainer;
                    std::vector<NeuronPtr> neuronContainer;
                    std::vector<double> nWeights;
                    NeuronPtr ptNeuron;

                    for (int i=0; i<=2; i++) {
                    ptNeuron.reset( new Neuron(ids[1]) );
                    neuronContainer.push_back(ptNeuron);
                    nWeights.push_back(0);                                  /
  / weights are set to 0
                    }
                    conContainer.buildAndAppend(neuronContainer, nWeights);
                    conContainer.show();

                    for (int i=0; i<=2; i++) {
                            nWeights.at(i)=weights[i];
                    }
    // Test
                    conContainer.setWeight(nWeights);                       /
  / weights are set to 12.3, 1.2 and 2.1
                    result=conContainer.getWeight();

    // Now result is a vector that contains the values 12.3, 1.2 and 2.1 .
```

**See also**

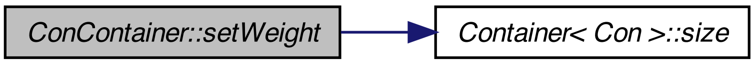
getWeight and the unit test files, e.g. runit.Cpp.ConContainer.R, for further examples.

Definition at line 270 of file ConContainer.cpp.

References Container< Con >::size().

```
{
  BEGIN_RCPP
  if (nWeights.empty())
    { throw std::range_error("[ C++ ConContainer::setWeight]: Error, nWeights is
      empty");}
  if (nWeights.size() != size())
    {
      throw std::range_error(
          "[C++ ConContainer::setWeight]: Error, nWeights.size() != collection.si
      ze()");
    }
  std::vector<double>::iterator itrWeight = nWeights.begin();
  foreach (ConPtr itr, *this)
    {
      itr->setWeight( *itrWeight );
      itrWeight++;
    }
  return true;
END_RCPP}
```

Here is the call graph for this function:



#### 5.7.4.8 bool ConContainer::validate ( ) [virtual]

Object validator.

This method checks the object for internal coherence. A try / catch mechanism exits normal execution and returns control to the R terminal in case the contents of the ConContainer object are identified as corrupted.

**Returns**

true in case the checks are Ok.

**Exceptions**

| | |
|---|---|
| *An* | std::range error if weight or from are not finite. |

**See also**

The unit test files, e.g., runit.Cpp.ConContainer.R, for usage examples.

Reimplemented from Container< Con >.

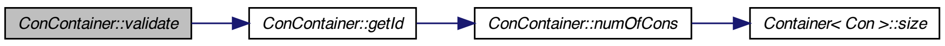
Definition at line 645 of file ConContainer.cpp.

References getId().

```
{
  BEGIN_RCPP

  std::vector<int>::iterator itr;
  std::vector<int> vIds = getId();
  sort(vIds.begin(), vIds.end());
  itr = adjacent_find(vIds.begin(), vIds.end());
  if (itr != vIds.end())
    throw std::range_error(
        "[C++ ConContainer::validate]: Error, duplicated Id.");
  Container<Con>::validate();
  return (true);
END_RCPP};
```

Here is the call graph for this function:



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

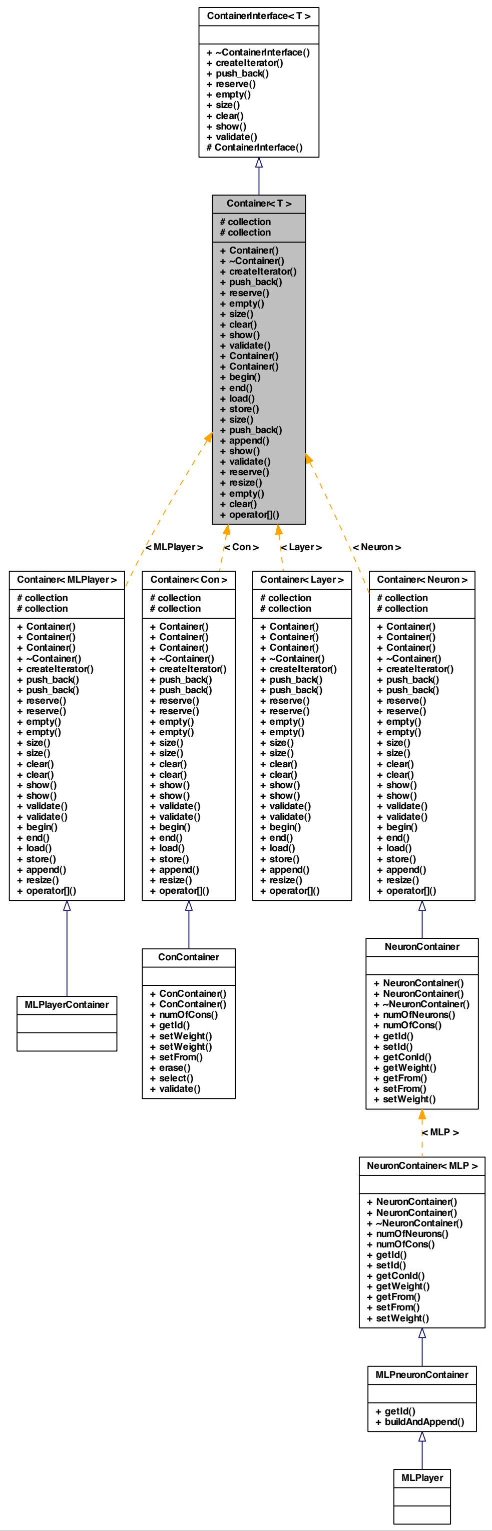
- pkg/AMORE/src/old/ConContainer.h
- pkg/AMORE/src/old/ConContainer.cpp

## 5.8 Container< T > Class Template Reference

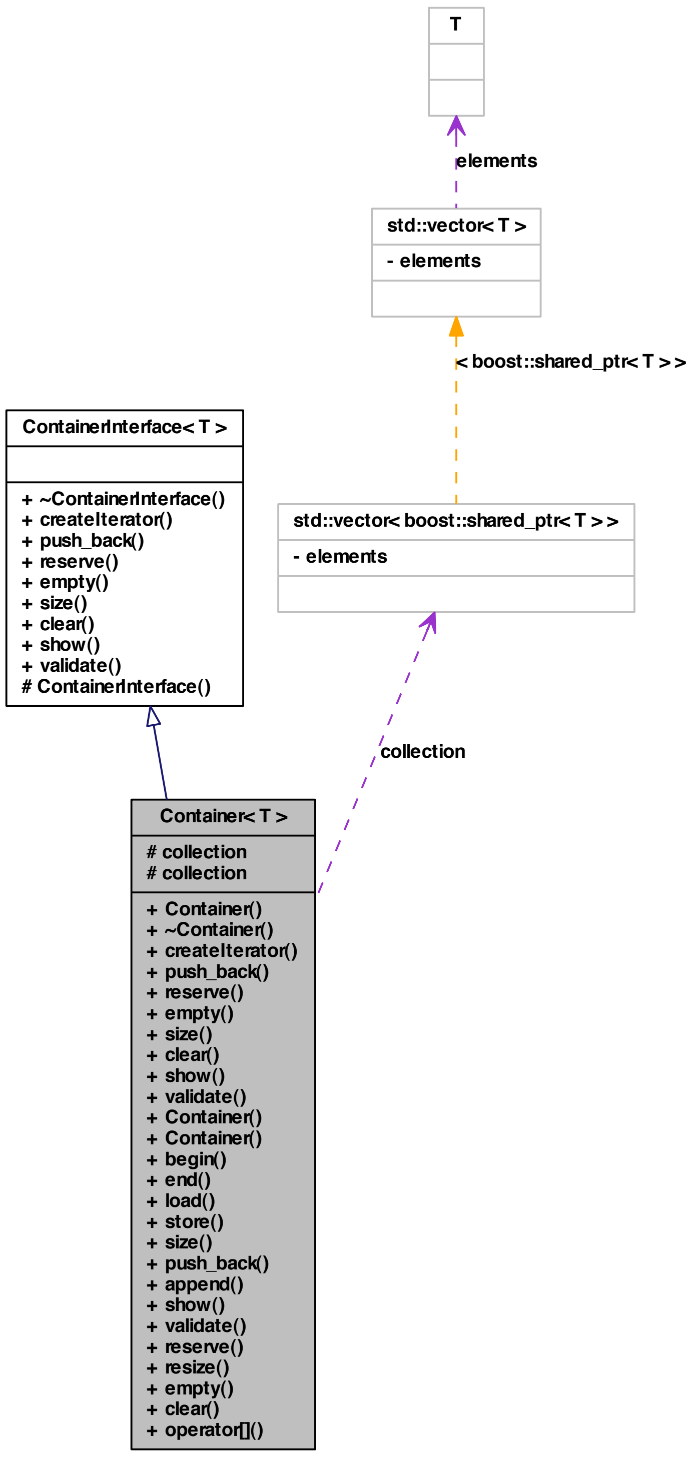
class Container -

`#include <Container.h>`

Inheritance diagram for Container< T >:



Collaboration diagram for Container< T >:



## Public Types

- typedef std::vector< boost::shared_ptr< T > >::iterator iterator
- typedef std::vector< boost::shared_ptr< T > >::const_iterator const_iterator
- typedef boost::shared_ptr< T > value_type
- typedef value_type const & const_reference

## Public Member Functions

- Container ()
- ~Container ()
- boost::shared_ptr< IteratorInterface< T > > createIterator ()
- void push_back (T const &const_reference)

  *Append a shared_ptr at the end of collection.*
- void reserve (int n)
- bool empty ()
- size_type size ()

  *Returns the size or length of the vector.*
- void clear ()
- void show ()

  *Pretty print of the Container<T>*
- bool validate ()

  *Object validator.*
- Container ()
- Container (std::vector< value > first, std::vector< value > last)
- iterator begin ()
- iterator end ()
- std::vector< boost::shared_ptr< T > > load ()

  *collection field accessor function*
- void store (typename std::vector< boost::shared_ptr< T > > collectionT)

  *collection field accessor function*
- size_type size ()
- void push_back (boost::shared_ptr< T > const &const_reference)

  *Append a shared_ptr at the end of collection.*
- void append (Container< T > containerT)

  *Appends a Container<T> object.*
- bool show ()
- bool validate ()
- void reserve (int n)
- void resize (int n)
- bool empty ()
- void clear ()
- boost::shared_ptr< T > & operator[] (size_type offset)

### Protected Attributes

- std::vector< T > collection
- std::vector< boost::shared_ptr< T > > collection

### Friends

- class ContainerIterator< T >

## 5.8.1 Detailed Description

**template<typename T>class Container< T >**

class Container -

Definition at line 6 of file Container.h.

## 5.8.2 Member Typedef Documentation

#### 5.8.2.1 template<typename T> typedef std::vector<boost::shared_ptr<T> >::const_iterator Container< T >::const_iterator

Reimplemented in ConContainer, and NeuronContainer.

Definition at line 22 of file Container.h.

#### 5.8.2.2 template<typename T> typedef value_type const& Container< T >::const_reference

Reimplemented in ConContainer, and NeuronContainer.

Definition at line 26 of file Container.h.

#### 5.8.2.3 template<typename T> typedef std::vector<boost::shared_ptr<T> >::iterator Container< T >::iterator

Reimplemented in ConContainer, and NeuronContainer.

Definition at line 19 of file Container.h.

#### 5.8.2.4 template<typename T> typedef boost::shared_ptr<T> Container< T >::value_type

Reimplemented in ConContainer, and NeuronContainer.

Definition at line 24 of file Container.h.

### 5.8.3 Constructor & Destructor Documentation

#### 5.8.3.1 template<typename T > Container< T >::Container ( )

Definition at line 11 of file Container.cpp.

```
{

}
```

#### 5.8.3.2 template<typename T > Container< T >::∼Container ( )

Definition at line 17 of file Container.cpp.

```
{
}
```

#### 5.8.3.3 template<typename T> Container< T >::Container ( )

#### 5.8.3.4 template<typename T> Container< T >::Container ( std::vector< value > *first*, std::vector< value > *last* )

### 5.8.4 Member Function Documentation

#### 5.8.4.1 template<typename T> void Container< T >::append ( Container< T > *v* )

Appends a Container<T> object.

This method inserts the collection field of a second object at the end of the collection field of the calling object.

**Parameters**

| | |
|---|---|
| *v* | The Container<T> object to be added to the current one |

**See also**

The unit test files, e.g., runit.Cpp.Container.R, for usage examples.

```
    //================
    //Usage example:
    //================
    // Data set up
                            std::vector<int> result;
                            std::vector<ConPtr> vcA, vcB;
                            ContainerNeuronPtr      neuronContainerPtr( new
  Container<Neuron>() );
                            ContainerConPtr conContainerPtrA( new
  Container<Con>() );
                            ContainerConPtr conContainerPtrB( new
  Container<Con>() );
```

```
                              ConPtr  ptC;
                              NeuronPtr ptN;
                              int ids[]= {1, 2, 3, 4, 5, 6};
                              double weights[] = {1.13, 2.22, 3.33, 5.6, 4.2, 3
.6 };
                              for (int i=0; i<=5 ; i++) {                      /
/ Let's create a vector with six neurons
                                      ptN.reset( new Neuron( ids[i] ) );
                                      neuronContainerPtr->push_back(ptN);
                              }
                              for (int i=0; i<=2 ; i++) {                      /
/ A vector with three connections
                                      ptC.reset( new Con( neuronContainerPtr->l
oad().at(i), weights[i]) );
                                      conContainerPtrA->push_back(ptC);
                              }
                              for (int i=3; i<=5 ; i++) {                      /
/ Another vector with three connections
                                      ptC.reset( new Con( neuronContainerPtr->l
oad().at(i), weights[i]) );
                                      conContainerPtrB->push_back(ptC);
                              }
  // Test
                              conContainerPtrA->append(*conContainerPtrB);
                              conContainerPtrA->validate();
                              conContainerPtrA->show() ;

  // After execution of the code above, the output at the R terminal would
display:
  //
  //   From:        1       Weight=         1.130000
  //       From:    2       Weight=         2.220000
  //       From:    3       Weight=         3.330000
  //       From:    4       Weight=         5.600000
  //       From:    5       Weight=         4.200000
  //       From:    6       Weight=         3.600000
```

**See also**

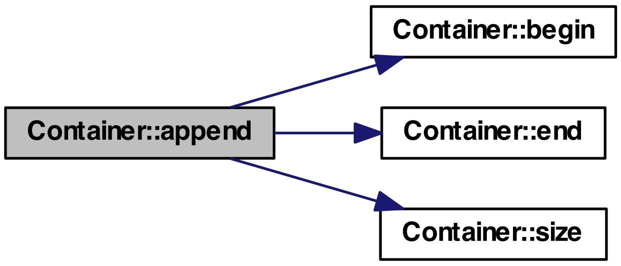
Container::store , Container::push_back and the unit test files, e.g., runit.Cpp.Container.R, for usage examples.

Definition at line 207 of file Container.cpp.

References Container< T >::begin(), Container< T >::end(), and Container< T >::size().

```
{
  reserve(size() + v.size());
  collection.insert(end(), v.begin(), v.end());
}
```

Here is the call graph for this function:



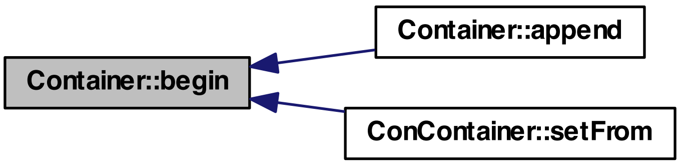
### 5.8.4.2 template< typename T > std::vector< boost::shared_ptr< T > >::iterator Container< T >::begin ( )

Definition at line 22 of file Container.cpp.

Referenced by Container< T >::append(), and ConContainer::setFrom().

```
{
  return collection.begin();
}
```

Here is the caller graph for this function:



### 5.8.4.3 template< typename T > void Container< T >::clear ( ) [virtual]

Implements ContainerInterface< T >.

Definition at line 177 of file Container.cpp.

```
{
  collection.clear();
}
```

#### 5.8.4.4 template<typename T> void Container< T >::clear ( ) [virtual]

Implements ContainerInterface< T >.

#### 5.8.4.5 template<typename T > boost::shared_ptr< IteratorInterface< T > > Container< T >::createIterator ( ) [virtual]

Implements ContainerInterface< T >.

Definition at line 23 of file Container.cpp.

```
{
  boost::shared_ptr< ContainerIterator<T> > containerIteratorPtr( new
    ContainerIterator<T> ());
  containerIteratorPtr->d_container = this;
  containerIteratorPtr->d_iterator = collection.begin();
  return containerIteratorPtr;
}
```

#### 5.8.4.6 template<typename T > bool Container< T >::empty ( ) [virtual]

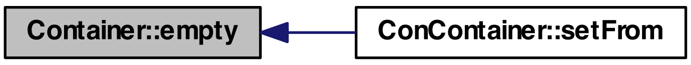
Implements ContainerInterface< T >.

Definition at line 163 of file Container.cpp.

Referenced by ConContainer::setFrom().

```
{
  return (collection.empty());
}
```

Here is the caller graph for this function:



#### 5.8.4.7 template<typename T> bool Container< T >::empty ( ) [virtual]

Implements ContainerInterface< T >.

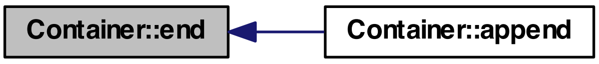
#### 5.8.4.8 template<typename T > std::vector< boost::shared_ptr< T > >::iterator Container< T >::end ( )

Definition at line 29 of file Container.cpp.

Referenced by Container< T >::append().

```
{
  return collection.end();
}
```

Here is the caller graph for this function:



#### 5.8.4.9 template<typename T > std::vector< boost::shared_ptr< T > > Container< T >::load ( )

collection field accessor function

This method allows access to the data stored in the collection field.

**Returns**

The collection vector.

```
    //================
    //Usage example:
    //================
            // Data set up
                            std::vector<int> result;
                            std::vector<ConPtr> vcA, vcB;
                            ContainerNeuronPtr      neuronContainerPtr( new
  Container<Neuron>() );
                            ContainerConPtr conContainerPtr( new
  Container<Con>() );
                            ConPtr  ptC;
                            NeuronPtr ptN;
                            int ids[]= {10, 20, 30};
```

```
                                double weights[] = {1.13, 2.22, 3.33 };
                                for (int i=0; i<=2 ; i++) {                        /
    / Let's create a vector with three neurons
                                        ptN.reset( new Neuron( ids[i] ) );
                                        neuronContainerPtr->push_back(ptN);
                                }
                                for (int i=0; i<=2 ; i++) {                        /
    / and a vector with three connections
                                        ptC.reset( new Con( neuronContainerPtr->l
    oad().at(i), weights[i]) );
                                        vcA.push_back(ptC);
                                }
              // Test
                        conContainerPtr->store(vcA);
                        vcB = conContainerPtr->load();
                        for (int i=0; i<=2 ; i++) {                                /
    / get Ids. Container does not have getId defined
                                        result.push_back( vcB.at(i)->getId());
                        }

              // Now, result is an integer vector with values 10, 20, 30.
```

**See also**

store and the unit test files, e.g., runit.Cpp.Container.R, for usage examples.

Definition at line 254 of file Container.cpp.

```
{
  return collection;
}
```

### 5.8.4.10 template<typename T > boost::shared_ptr< T > & Container< T >::operator[] ( size_type *offset* )

Definition at line 317 of file Container.cpp.

```
{
  return collection[offset];
}
```

### 5.8.4.11 template<typename T> void Container< T >::push_back ( T const & *reference* ) [virtual]

Append a shared_ptr at the end of collection.

Implements push_back for the Container class

**Parameters**

| | |
|---|---|
| *TsharedPtr* | A shared_ptr pointer to be inserted at the end of collection |

```
              //================
                //Usage example:
                //================
                // Data set up
                        Neuron N1, N2, N3;
                        Container<Con> conContainer;
                        std::vector<ConPtr> vc;
                        std::vector<int> result;
                        N1.setId(10);
                        N2.setId(20);
                        N3.setId(30);
                // Test
                        ConPtr ptCon( new Con(&N1, 1.13) );     // Create new Con
         and initialize ptCon
                        conContainer.push_back(ptCon);                          /
        / push_back
                        ptCon.reset(  new Con(&N2, 2.22) );             // create
         new Con and assign to ptCon
                        conContainer.push_back(ptCon);                          /
        / push_back
                        ptCon.reset(  new Con(&N3, 3.33) );             // create
         new Con and assign to ptCon
                        conContainer.push_back(ptCon);                          /
        / push_back

                        vc = conContainer.load();

                        result.push_back(vc.at(0)->getId());
                        result.push_back(vc.at(1)->getId());
                        result.push_back(vc.at(2)->getId());
          // After execution of this code, result contains a numeric vector with va
        lues 10, 20 and 30.
```

**See also**

C++ documentation for std::vector::push_back and the unit test files, e.g., runit.Cpp.Container.R, for usage examples.

Implements ContainerInterface< T >.

Definition at line 68 of file Container.cpp.

```
{
  collection.push_back(reference);
}
```

#### 5.8.4.12 template<typename T> void Container< T >::push_back ( boost::shared_ptr< T > const & *const_reference* )

Append a shared_ptr at the end of collection.

Implements push_back for the Container class

**Parameters**

| | |
|---|---|
| *TsharedPtr* | A shared_ptr pointer to be inserted at the end of collection |

```
                //================
                //Usage example:
                //================
                // Data set up
                        Neuron N1, N2, N3;
                        Container<Con> conContainer;
                        std::vector<ConPtr> vc;
                        std::vector<int> result;
                        N1.setId(10);
                        N2.setId(20);
                        N3.setId(30);
                // Test
                        ConPtr ptCon( new Con(&N1, 1.13) );     // Create new Con
         and initialize ptCon
                        conContainer.push_back(ptCon);                          /
        / push_back
                        ptCon.reset(  new Con(&N2, 2.22) );             // create
         new Con and assign to ptCon
                        conContainer.push_back(ptCon);                          /
        / push_back
                        ptCon.reset(  new Con(&N3, 3.33) );             // create
         new Con and assign to ptCon
                        conContainer.push_back(ptCon);                          /
        / push_back

                        vc = conContainer.load();

                        result.push_back(vc.at(0)->getId());
                        result.push_back(vc.at(1)->getId());
                        result.push_back(vc.at(2)->getId());
         // After execution of this code, result contains a numeric vector with va
        lues 10, 20 and 30.
```

**See also**

C++ documentation for std::vector::push_back and the unit test files, e.g., runit.Cpp.Container.R, for usage examples.

Definition at line 71 of file Container.cpp.

```
{
  collection.push_back(const_reference);
}
```

#### 5.8.4.13 template<typename T> void Container< T >::reserve ( int *n* ) [virtual]

Implements ContainerInterface< T >.

#### 5.8.4.14 template<typename T > void Container< T >::reserve ( int *n* ) [virtual]

Implements ContainerInterface< T >.

Definition at line 170 of file Container.cpp.

```
{
  collection.reserve(n);
}
```

#### 5.8.4.15 template<typename T > void Container< T >::resize ( int *n* )

Definition at line 289 of file Container.cpp.

```
{
  collection.resize(n);
}
```

#### 5.8.4.16 template<typename T> bool Container< T >::show ( ) [virtual]

Implements ContainerInterface< T >.

#### 5.8.4.17 template<typename T > bool Container< T >::show ( ) [virtual]

Pretty print of the Container<T>

This method outputs in the R terminal the contents of Container::collection.

**Returns**

true in case everything works without throwing an exception

*

```
                //================
                //Usage example:
                //================
                // Data set up
                        ContainerNeuronPtr      neuronContainerPtr( new
      Container<Neuron>() );
                        ContainerConPtr conContainerPtr( new Container<Con>() );
                        ConPtr  ptC;
                        NeuronPtr ptN;
                        int ids[]= {10, 20, 30};
                        double weights[] = {1.13, 2.22, 3.33 };

                        for (int i=0; i<=2 ; i++) {                          /
      / Let's create a vector with three neurons
                                ptN.reset( new Neuron( ids[i] ) );
                                neuronContainerPtr->push_back(ptN);
                        }

                        for (int i=0; i<=2 ; i++) {                          /
      / and a vector with three connections
                                ptC.reset( new Con( neuronContainerPtr->load().at
      (i), weights[i]) );
                                conContainerPtr->push_back(ptC);
                        }

                // Test
                        conContainerPtr->show() ;

                // The output at the R terminal would display:
                //
                //      # From:  10      Weight=         1.130000
                //      # From:  20      Weight=         2.220000
                //      # From:  30      Weight=         3.330000
                //
```

**See also**

The unit test files, e.g., runit.Cpp.Container.R, for usage examples.

Implements ContainerInterface< T >.

Definition at line 118 of file Container.cpp.

```
{

  for (typename std::vector<T>::iterator itr(collection.begin()); itr
      != collection.end(); ++itr)
    {
      itr->show();
    }

}
```

#### 5.8.4.18 template<typename T> size_type Container< T >::size ( ) [virtual]

Implements ContainerInterface< T >.

#### 5.8.4.19 template<typename T > size_type Container< T >::size ( ) [virtual]

Returns the size or length of the vector.

This method returns the size of the vector. In the classes derived from Container<T> this is aliased as numOfCons, numOfNeurons and numOfLayers. The unit test files, e.g., runit.Cpp.Container.R, for usage examples.

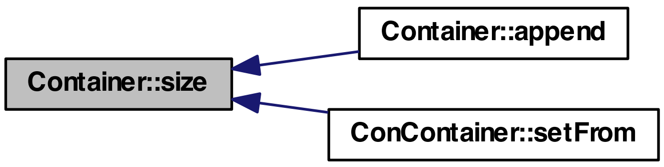
Implements ContainerInterface< T >.

Definition at line 155 of file Container.cpp.

Referenced by Container< T >::append(), and ConContainer::setFrom().

```
{
  return collection.size();
}
```

Here is the caller graph for this function:



#### 5.8.4.20 template<typename T> void Container< T >::store ( typename std::vector< boost::shared_ptr< T > > *collectionT* )

collection field accessor function

This method sets the value of the data stored in the collection field.

**Parameters**

| | |
|---|---|
| *v* | The vector of smart pointers to be stored in the collection field |

**See also**

load and the unit test files, e.g., runit.Cpp.Container.R, for usage examples.

Definition at line 268 of file Container.cpp.

```
{
  collection = collectionT;
}
```

#### 5.8.4.21 template<typename T> bool Container< T >::validate ( ) [virtual]

Implements ContainerInterface< T >.

Reimplemented in ConContainer.

#### 5.8.4.22 template<typename T > bool Container< T >::validate ( ) [virtual]

Object validator.

This method checks the object for internal coherence. This method calls the validate method for each element in collection,

**See also**

The unit test files, e.g., runit.Cpp.Container.R, for usage examples.

Implements ContainerInterface< T >.

Reimplemented in ConContainer.

Definition at line 136 of file Container.cpp.

```
{
  for (typename std::vector<T>::iterator itr(collection.begin()); itr
       != collection.end(); ++itr)
    {
      itr->validate();
    }
  return true;
}
```

### 5.8.5 Friends And Related Function Documentation

#### 5.8.5.1 template<typename T> friend class ContainerIterator< T > `[friend]`

Definition at line 12 of file Container.h.

### 5.8.6 Member Data Documentation

#### 5.8.6.1 template<typename T> std::vector< T > Container< T >::collection `[protected]`

Definition at line 9 of file Container.h.

#### 5.8.6.2 template<typename T> std::vector<boost::shared_ptr<T> > Container< T >::collection `[protected]`

Definition at line 15 of file Container.h.

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

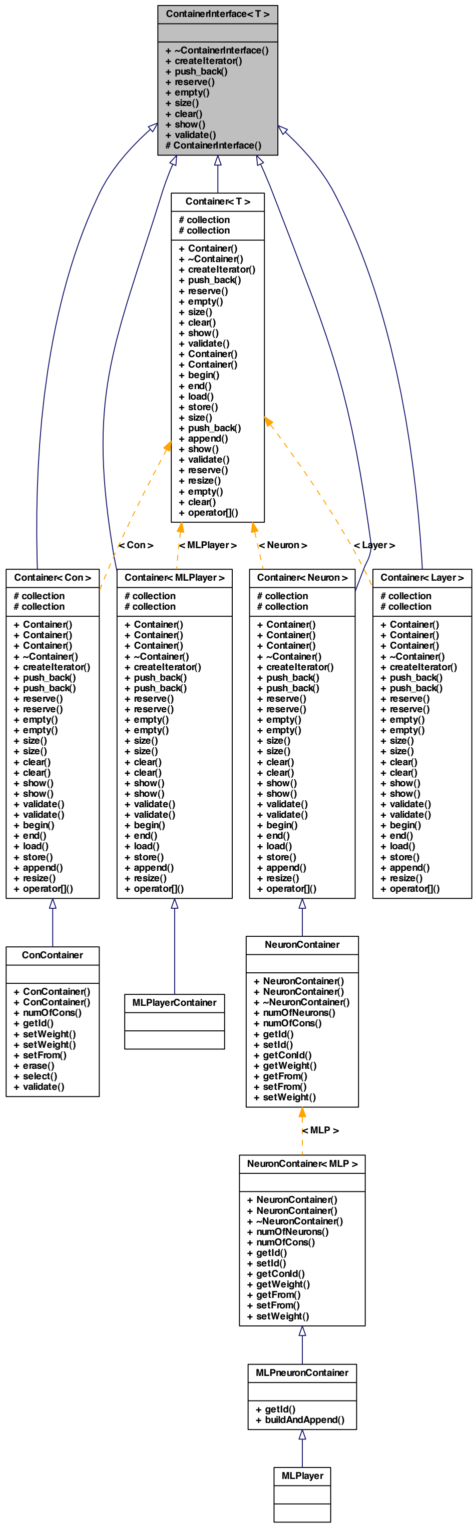
- pkg/AMORE/src/dia/Container.h
- pkg/AMORE/src/old/Container.h
- pkg/AMORE/src/Container.cpp
- pkg/AMORE/src/old/Container.cpp

## 5.9 ContainerInterface< T > Class Template Reference

class ContainerInterface -

`#include <ContainerInterface.h>`

Inheritance diagram for ContainerInterface< T >:



### Public Member Functions

- virtual ~ContainerInterface ()
- virtual boost::shared_ptr< IteratorInterface< T > > createIterator ()=0
- virtual void push_back (T const &const_reference)=0
- virtual void reserve (int n)=0
- virtual bool empty ()=0
- virtual size_type size ()=0
- virtual void clear ()=0
- virtual void show ()=0
- virtual bool validate ()=0

### Protected Member Functions

- ContainerInterface ()

### 5.9.1 Detailed Description

**template<typename T>class ContainerInterface< T >**

class ContainerInterface -

Definition at line 4 of file ContainerInterface.h.

### 5.9.2 Constructor & Destructor Documentation

#### 5.9.2.1 template<typename T > ContainerInterface< T >::~ContainerInterface ( ) [virtual]

Definition at line 10 of file containerInterface.cpp.

```
{
}
```

#### 5.9.2.2 template<typename T > ContainerInterface< T >::ContainerInterface ( ) [protected]

Definition at line 4 of file containerInterface.cpp.

```
{
}
```

### 5.9.3 Member Function Documentation

#### 5.9.3.1 template<typename T > virtual void ContainerInterface< T >::clear ( ) [pure virtual]

Implemented in Container< T >, Container< T >, Container< MLPlayer >, Container< MLPlayer >, Container< Con >, Container< Con >, Container< Layer >, Container< Layer >, Container< Neuron >, and Container< Neuron >.

#### 5.9.3.2 template<typename T > virtual boost::shared_ptr< IteratorInterface<T> > ContainerInterface< T >::createIterator ( ) [pure virtual]

Implemented in Container< T >, Container< MLPlayer >, Container< Con >, Container< Layer >, and Container< Neuron >.

#### 5.9.3.3 template<typename T > virtual bool ContainerInterface< T >::empty ( ) [pure virtual]

Implemented in Container< T >, Container< T >, Container< MLPlayer >, Container< MLPlayer >, Container< Con >, Container< Con >, Container< Layer >, Container< Layer >, Container< Neuron >, and Container< Neuron >.

#### 5.9.3.4 template<typename T > virtual void ContainerInterface< T >::push_back ( T const & *const_reference* ) [pure virtual]

Implemented in Container< T >.

#### 5.9.3.5 template<typename T > virtual void ContainerInterface< T >::reserve ( int *n* ) [pure virtual]

Implemented in Container< T >, Container< T >, Container< MLPlayer >, Container< MLPlayer >, Container< Con >, Container< Con >, Container< Layer >, Container< Layer >, Container< Neuron >, and Container< Neuron >.

#### 5.9.3.6 template<typename T > virtual void ContainerInterface< T >::show ( ) [pure virtual]

Implemented in Container< T >, Container< T >, Container< MLPlayer >, Container< MLPlayer >, Container< Con >, Container< Con >, Container< Layer >, Container< Layer >, Container< Neuron >, and Container< Neuron >.

#### 5.9.3.7 template<typename T > virtual size_type ContainerInterface< T >::size ( ) [pure virtual]

Implemented in Container< T >, Container< T >, Container< MLPlayer >, Container< MLPlayer >, Container< Con >, Container< Con >, Container< Layer >, Container< Layer >, Container< Neuron >, and Container< Neuron >.

#### 5.9.3.8 template<typename T > virtual bool ContainerInterface< T >::validate ( ) [pure virtual]

Implemented in Container< T >, ConContainer, Container< T >, Container< MLPlayer >, Container< MLPlayer >, Container< Con >, Container< Con >, Container< Layer >, Container< Layer >, Container< Neuron >, and Container< Neuron >.

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

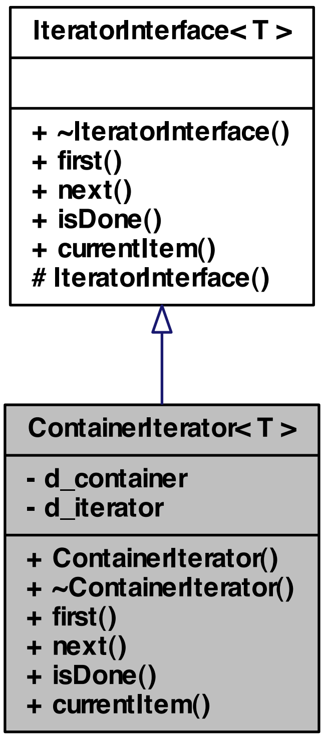
- pkg/AMORE/src/dia/ContainerInterface.h
- pkg/AMORE/src/containerInterface.cpp

## 5.10 ContainerIterator< T > Class Template Reference

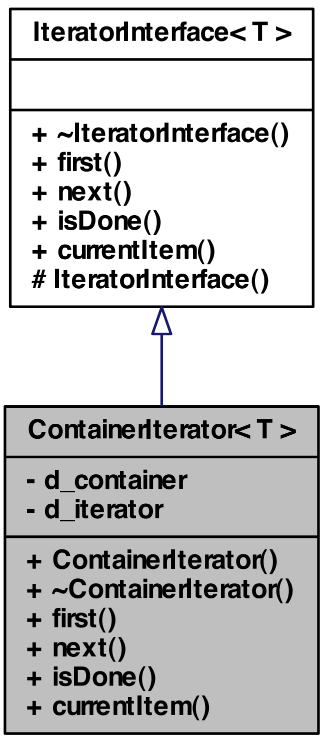
class ContainerIterator -

`#include <ContainerIterator.h>`

Inheritance diagram for ContainerIterator< T >:



Collaboration diagram for ContainerIterator< T >:



## Public Member Functions

- ContainerIterator ()
- ∼ContainerIterator ()
- void first ()
- void next ()
- bool isDone ()
- T currentItem ()

## Private Attributes

- Container< T > ∗ d_container
- std::vector< T >::iterator d_iterator

## Friends

- class Container< T >

### 5.10.1 Detailed Description

**template<typename T>class ContainerIterator< T >**

class ContainerIterator -

Definition at line 6 of file ContainerIterator.h.

### 5.10.2 Constructor & Destructor Documentation

#### 5.10.2.1 template<typename T > ContainerIterator< T >::ContainerIterator ( )

Definition at line 4 of file ContainerIterator.cpp.

```
{
}
```

#### 5.10.2.2 template<typename T > ContainerIterator< T >::~ContainerIterator ( )

Definition at line 9 of file ContainerIterator.cpp.

```
{
}
```

### 5.10.3 Member Function Documentation

#### 5.10.3.1 template<typename T > T ContainerIterator< T >::currentItem ( ) [virtual]

Implements IteratorInterface< T >.

Definition at line 37 of file ContainerIterator.cpp.

References ContainerIterator< T >::d_iterator.

```
{
  return *d_iterator;
}
```

#### 5.10.3.2 template<typename T > void ContainerIterator< T >::first ( ) [virtual]

Implements IteratorInterface< T >.

Definition at line 15 of file ContainerIterator.cpp.

References ContainerIterator< T >::d_container, and ContainerIterator< T >::d_iterator.

```
{
  d_iterator = d_container->collection.begin();
}
```

#### 5.10.3.3 template<typename T > bool ContainerIterator< T >::isDone ( ) [virtual]

Implements IteratorInterface< T >.

Definition at line 29 of file ContainerIterator.cpp.

References ContainerIterator< T >::d_container, and ContainerIterator< T >::d_iterator.

```
{
  bool IteratorIsDone(d_iterator == d_container->collection.end());
  return IteratorIsDone;
}
```

#### 5.10.3.4 template<typename T > void ContainerIterator< T >::next ( ) [virtual]

Implements IteratorInterface< T >.

Definition at line 22 of file ContainerIterator.cpp.

References ContainerIterator< T >::d_iterator.

```
{
  ++d_iterator;
}
```

### 5.10.4 Friends And Related Function Documentation

#### 5.10.4.1 template<typename T > friend class Container< T > [friend]

Definition at line 13 of file ContainerIterator.h.

### 5.10.5 Member Data Documentation

#### 5.10.5.1 template<typename T > Container<T>* ContainerIterator< T >::d_container [private]

Definition at line 9 of file ContainerIterator.h.

Referenced by ContainerIterator< T >::first(), and ContainerIterator< T >::isDone().

#### 5.10.5.2 template<typename T > std::vector<T>::iterator ContainerIterator< T >::d_iterator [private]

Definition at line 10 of file ContainerIterator.h.

Referenced by ContainerIterator< T >::currentItem(), ContainerIterator< T >::first(), ContainerIterator< T >::isDone(), and ContainerIterator< T >::next().

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

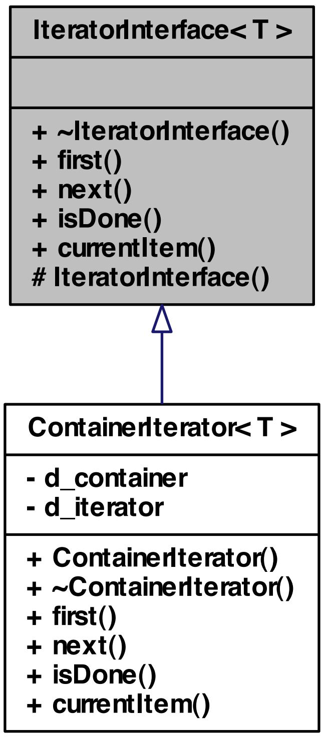
- pkg/AMORE/src/dia/ContainerIterator.h
- pkg/AMORE/src/ContainerIterator.cpp

## 5.11 IteratorInterface< T > Class Template Reference

class IteratorInterface -

`#include <IteratorInterface.h>`

Inheritance diagram for IteratorInterface< T >:



## Public Member Functions

- virtual ~IteratorInterface ()
- virtual void first ()=0
- virtual void next ()=0
- virtual bool isDone ()=0
- virtual T currentItem ()=0

## Protected Member Functions

- IteratorInterface ()

### 5.11.1 Detailed Description

**template<typename T>class IteratorInterface< T >**

class IteratorInterface -

Definition at line 4 of file IteratorInterface.h.

## 5.11.2 Constructor & Destructor Documentation

### 5.11.2.1 template<typename T > IteratorInterface< T >::~IteratorInterface ( ) [virtual]

Definition at line 10 of file IteratorInterface.cpp.

```
{
}
```

### 5.11.2.2 template<typename T > IteratorInterface< T >::IteratorInterface ( ) [protected]

Definition at line 4 of file IteratorInterface.cpp.

```
{
}
```

## 5.11.3 Member Function Documentation

### 5.11.3.1 template<typename T > virtual T IteratorInterface< T >::currentItem ( ) [pure virtual]

Implemented in ContainerIterator< T >.

### 5.11.3.2 template<typename T > virtual void IteratorInterface< T >::first ( ) [pure virtual]

Implemented in ContainerIterator< T >.

### 5.11.3.3 template<typename T > virtual bool IteratorInterface< T >::isDone ( ) [pure virtual]

Implemented in ContainerIterator< T >.

### 5.11.3.4 template<typename T > virtual void IteratorInterface< T >::next ( ) [pure virtual]

Implemented in ContainerIterator< T >.

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

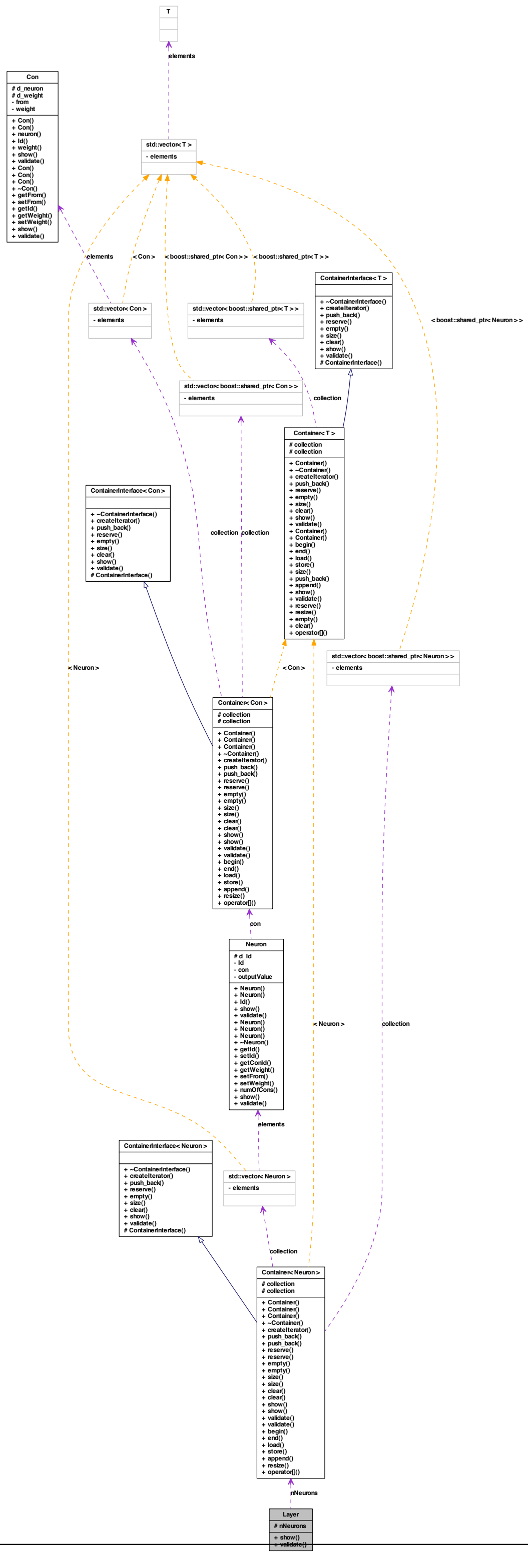
- pkg/AMORE/src/dia/IteratorInterface.h
- pkg/AMORE/src/IteratorInterface.cpp

## 5.12 Layer Class Reference

class Layer -

`#include <Layer.h>`

Collaboration diagram for Layer:



## Public Member Functions

- void show ()
- bool validate ()

## Protected Attributes

- Container< Neuron > nNeurons

### 5.12.1 Detailed Description

class Layer -

Definition at line 3 of file Layer.h.

### 5.12.2 Member Function Documentation

#### 5.12.2.1 void Layer::show ( )

#### 5.12.2.2 bool Layer::validate ( )

### 5.12.3 Member Data Documentation

#### 5.12.3.1 Container<Neuron> Layer::nNeurons [protected]

Definition at line 6 of file Layer.h.

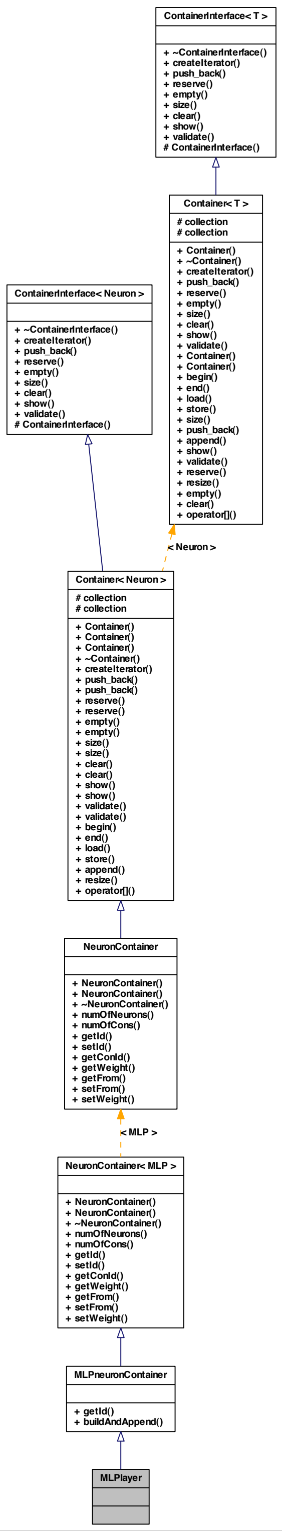
The documentation for this class was generated from the following file:

- pkg/AMORE/src/dia/Layer.h

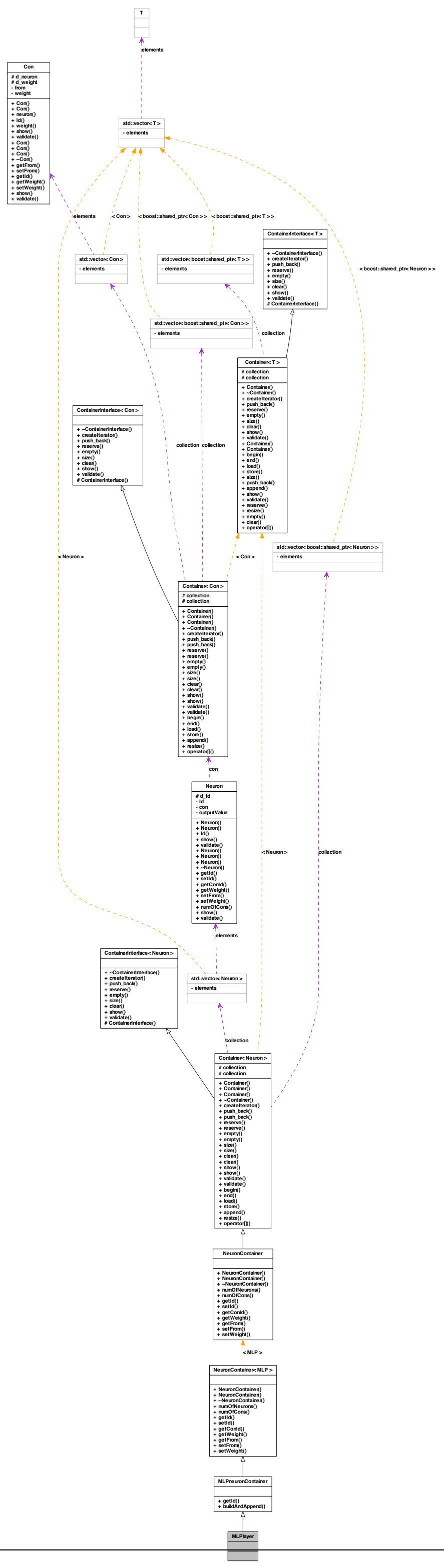
## 5.13 MLPlayer Class Reference

```
#include <MLPlayer.h>
```

Inheritance diagram for MLPlayer:



Collaboration diagram for MLPlayer:



### 5.13.1 Detailed Description

Definition at line 1 of file MLPlayer.h.

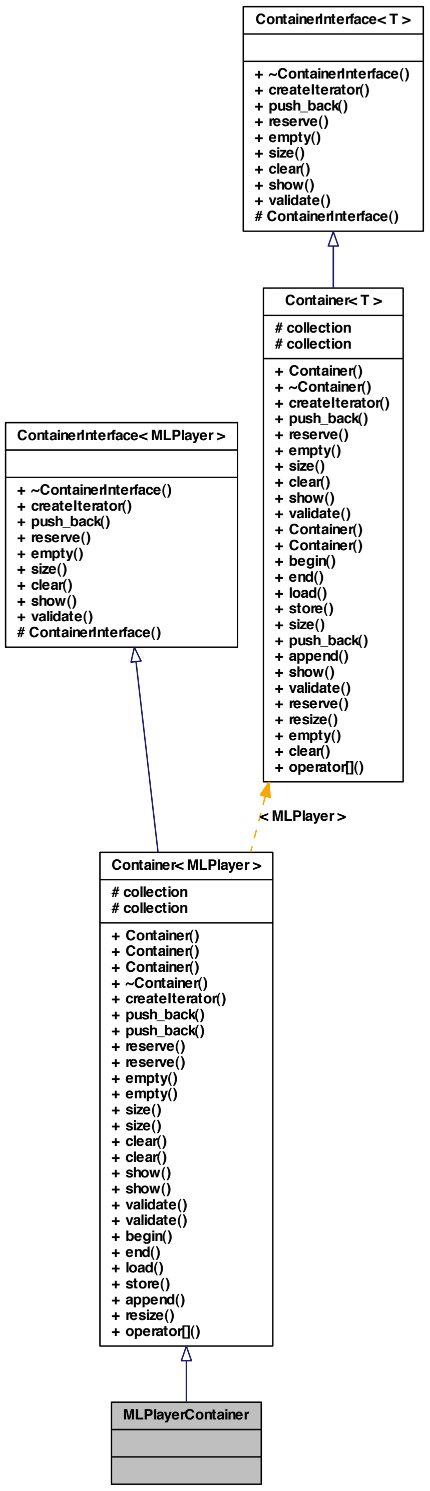
The documentation for this class was generated from the following file:

- pkg/AMORE/src/old/MLPlayer.h

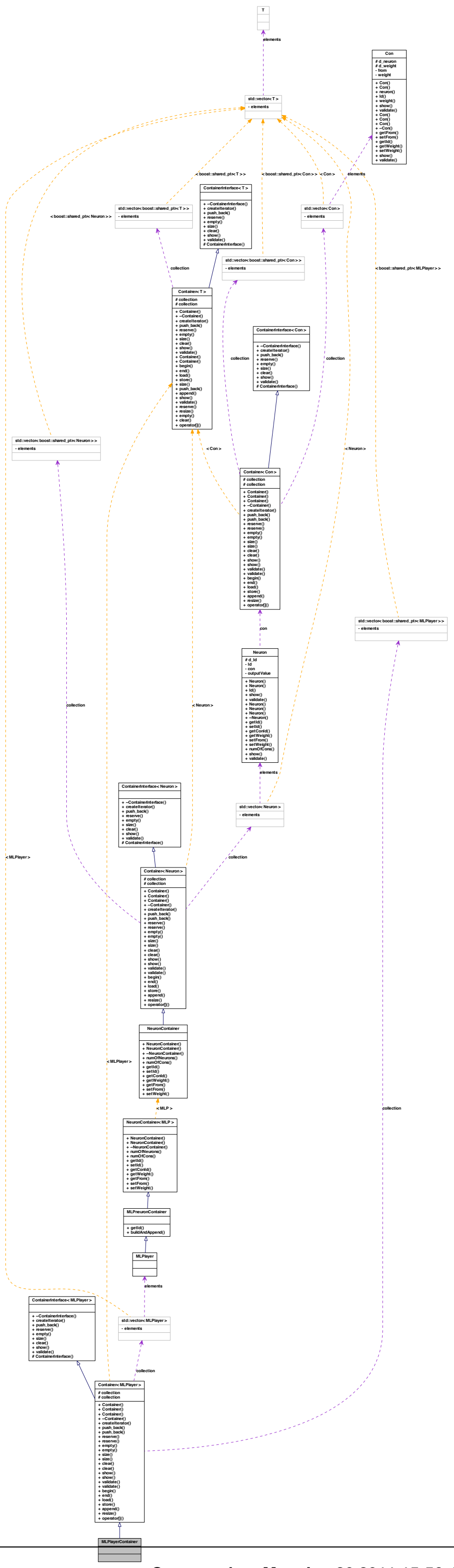
## 5.14 MLPlayerContainer Class Reference

```
#include <MLPlayerContainer.h>
```

Inheritance diagram for MLPlayerContainer:



Collaboration diagram for MLPlayerContainer:



### 5.14.1 Detailed Description

Definition at line 1 of file MLPlayerContainer.h.

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

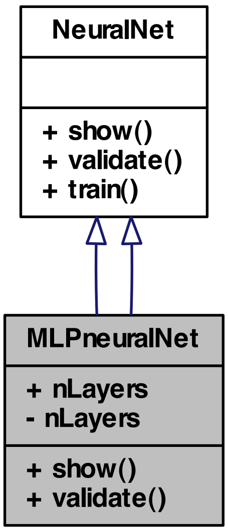
- pkg/AMORE/src/old/MLPlayerContainer.h

## 5.15 MLPneuralNet Class Reference

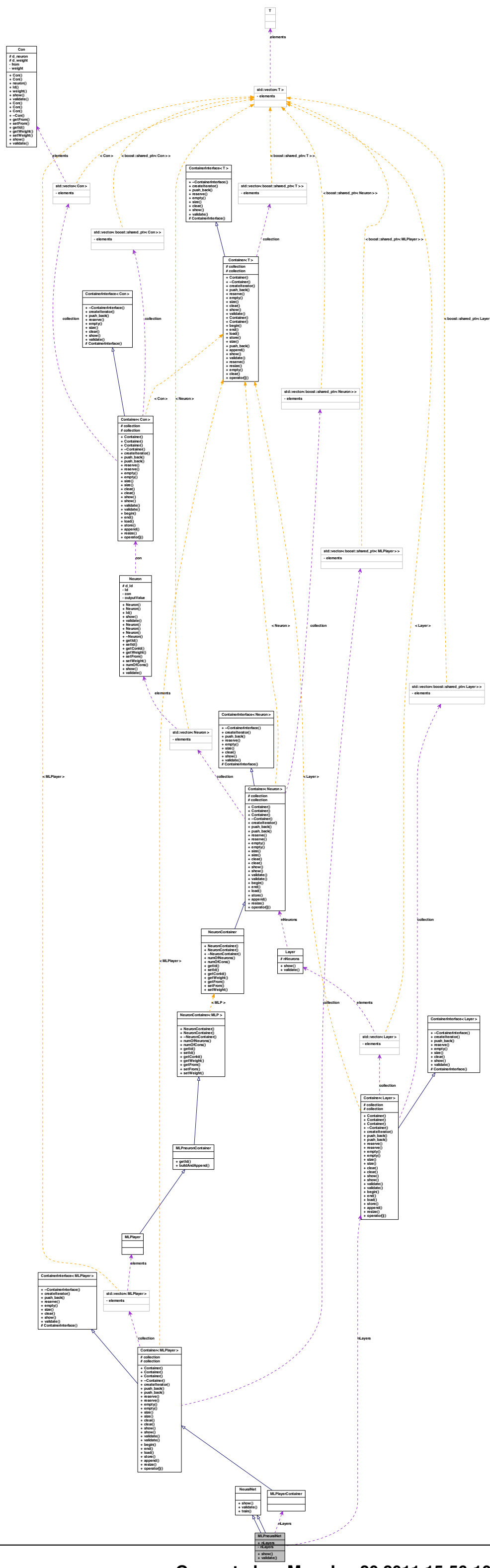
class MLPneuralNet -

```
#include <MLPneuralNet.h>
```

Inheritance diagram for MLPneuralNet:



Collaboration diagram for MLPneuralNet:



## Public Member Functions

- void show ()
- bool validate ()

## Public Attributes

- Container< Layer > nLayers

## Private Attributes

- MLPlayerContainer nLayers

### 5.15.1 Detailed Description

class MLPneuralNet -

Definition at line 5 of file MLPneuralNet.h.

### 5.15.2 Member Function Documentation

#### 5.15.2.1 void MLPneuralNet::show ( )

Reimplemented from NeuralNet.

#### 5.15.2.2 bool MLPneuralNet::validate ( )

Reimplemented from NeuralNet.

### 5.15.3 Member Data Documentation

#### 5.15.3.1 Container<Layer> MLPneuralNet::nLayers

Definition at line 8 of file MLPneuralNet.h.

#### 5.15.3.2 MLPlayerContainer MLPneuralNet::nLayers `[private]`

Definition at line 2 of file MLPneuralNet.h.

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

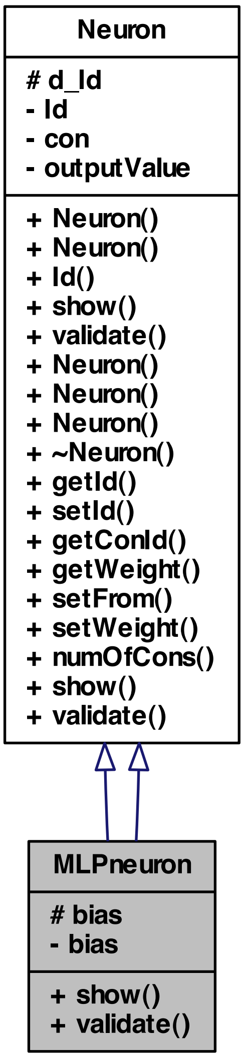
- pkg/AMORE/src/dia/MLPneuralNet.h
- pkg/AMORE/src/old/MLPneuralNet.h

## 5.16 MLPneuron Class Reference

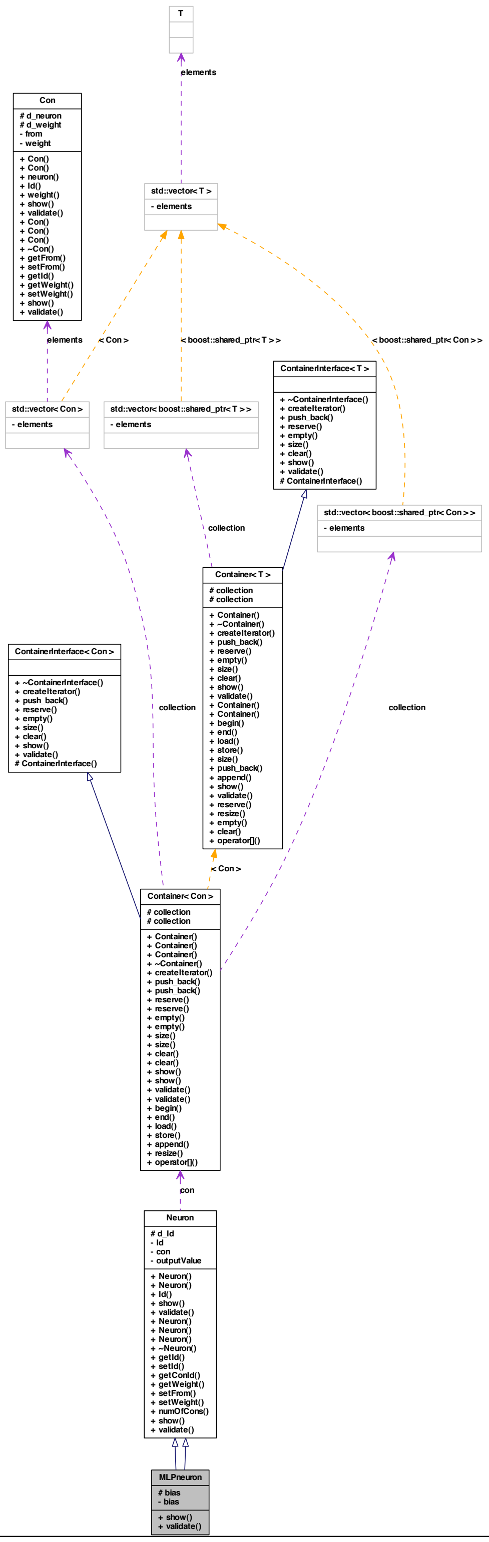
class MLPneuron -

```
#include <MLPneuron.h>
```

Inheritance diagram for MLPneuron:



Collaboration diagram for MLPneuron:



## Public Member Functions

- void show ()
- bool validate ()

## Protected Attributes

- double bias

## Private Attributes

- int bias

### 5.16.1 Detailed Description

class MLPneuron -

Definition at line 5 of file MLPneuron.h.

### 5.16.2 Member Function Documentation

#### 5.16.2.1 void MLPneuron::show ( )

Reimplemented from Neuron.

#### 5.16.2.2 bool MLPneuron::validate ( )

Reimplemented from Neuron.

### 5.16.3 Member Data Documentation

#### 5.16.3.1 double **MLPneuron::bias** `[protected]`

Definition at line 8 of file MLPneuron.h.

#### 5.16.3.2 int **MLPneuron::bias** `[private]`

Definition at line 2 of file MLPneuron.h.

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

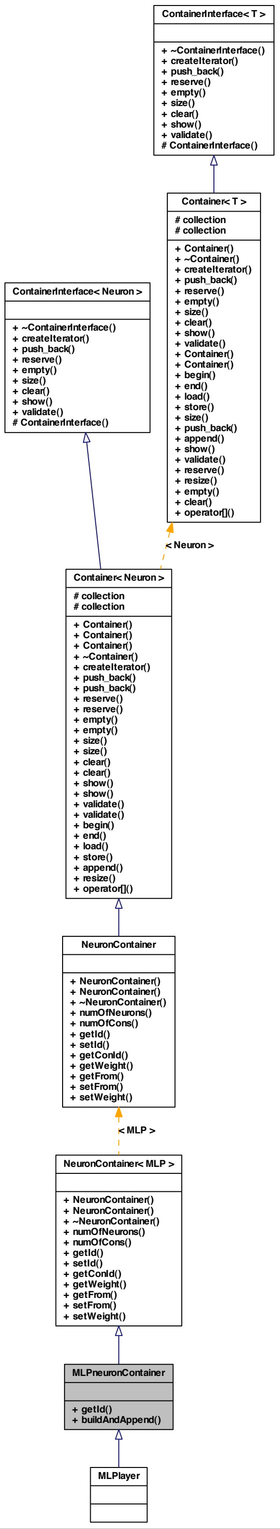
- pkg/AMORE/src/dia/MLPneuron.h
- pkg/AMORE/src/old/MLPneuron.h

## 5.17 MLPneuronContainer Class Reference

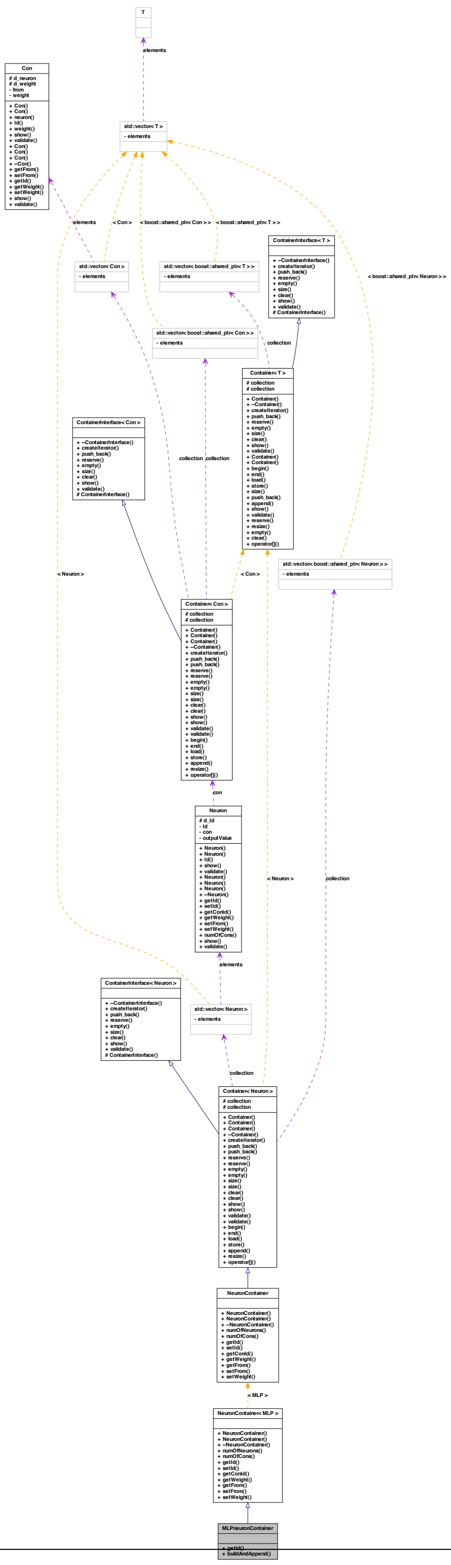
A vector of connections.

```
#include <MLPneuronContainer.h>
```

Inheritance diagram for MLPneuronContainer:



Collaboration diagram for MLPneuronContainer:



### Public Member Functions

- std::vector< int > getId ()
- bool buildAndAppend (std::vector< int > IDS, std::vector< int > BIAS, ConContainer VC)

### 5.17.1 Detailed Description

A vector of connections.

The ConContainer class provides a simple class for a vector of connections. It's named after the R equivalent Reference Class.

Definition at line 16 of file MLPneuronContainer.h.

### 5.17.2 Member Function Documentation

#### 5.17.2.1 bool MLPneuronContainer::buildAndAppend ( std::vector< int > *IDS*, std::vector< int > *BIAS*, ConContainer *VC* )

#### 5.17.2.2 std::vector<int> MLPneuronContainer::getId ( )

Reimplemented from NeuronContainer< MLP >.

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

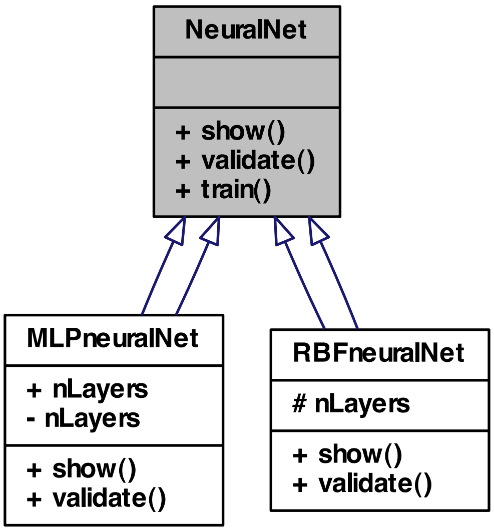
- pkg/AMORE/src/old/MLPneuronContainer.h

## 5.18 NeuralNet Class Reference

class NeuralNet -

```
#include <NeuralNet.h>
```

Inheritance diagram for NeuralNet:



## Public Member Functions

- void show ()
- bool validate ()
- virtual void train ()=0

## 5.18.1 Detailed Description

class NeuralNet -

Definition at line 3 of file NeuralNet.h.

## 5.18.2 Member Function Documentation

### 5.18.2.1 void NeuralNet::show ( )

Reimplemented in MLPneuralNet, and RBFneuralNet.

### 5.18.2.2 virtual void NeuralNet::train ( ) [pure virtual]

#### 5.18.2.3 bool NeuralNet::validate ( )

Reimplemented in MLPneuralNet, and RBFneuralNet.

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

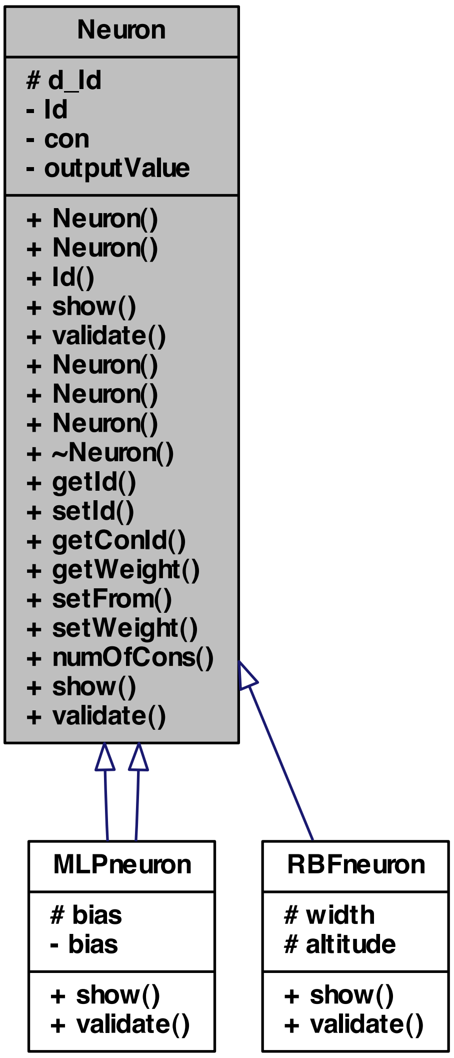
- pkg/AMORE/src/dia/NeuralNet.h
- pkg/AMORE/src/old/NeuralNet.h

## 5.19 Neuron Class Reference

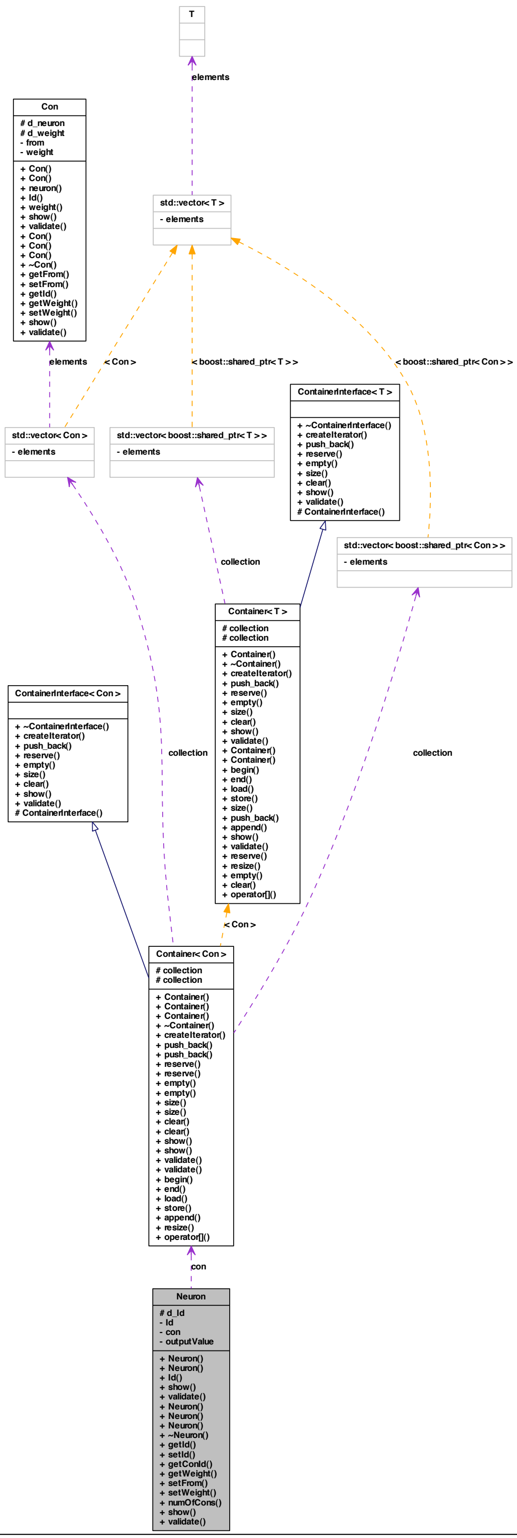
class Neuron -

```
#include <Neuron.h>
```

Inheritance diagram for Neuron:



Collaboration diagram for Neuron:



## Public Member Functions

- Neuron ()
- Neuron (int Id)
- int Id ()
- void show ()
- bool validate ()
- Neuron ()
- Neuron (int Id)
- Neuron (int Id, ConContainer con)
- ~Neuron ()
- int getId ()
- void setId (int value)
- std::vector< int > getConId ()
- std::vector< double > getWeight ()
- bool setFrom (NeuronContainer neuronContainer)
- bool setWeight (std::vector< double > nWeights)
- int numOfCons ()
- bool show ()
- bool validate ()

## Protected Attributes

- int d_Id

## Private Attributes

- int Id

  *An integer variable with the Neuron Id.*
- ConContainer con

  *A vector of input connections.*
- double outputValue

### 5.19.1 Detailed Description

class Neuron -

A class to handle the information contained in a general Neuron.

A general class for neurons. The MLPneuron and RBFneuron classes will specialize this general class

Definition at line 3 of file Neuron.h.

### 5.19.2 Constructor & Destructor Documentation

#### 5.19.2.1 Neuron::Neuron ( )

Definition at line 10 of file Neuron.cpp.

```
                :
  d_Id(NA_INTEGER) //, nCons()
{
}
```

#### 5.19.2.2 Neuron::Neuron ( int *Id* )

Definition at line 15 of file Neuron.cpp.

```
                       :
  d_Id(Id)//, nCons()
{
}
```

#### 5.19.2.3 Neuron::Neuron ( )

#### 5.19.2.4 Neuron::Neuron ( int *Id* )

#### 5.19.2.5 Neuron::Neuron ( int *Id,* ConContainer *con* )

#### 5.19.2.6 Neuron::∼Neuron ( )

### 5.19.3 Member Function Documentation

#### 5.19.3.1 std::vector<int> Neuron::getConId ( )

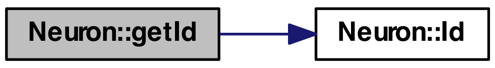
#### 5.19.3.2 int Neuron::getId ( )

Definition at line 26 of file Neuron.cpp.

References Id().

```
{
  return Id;
}
```

Here is the call graph for this function:

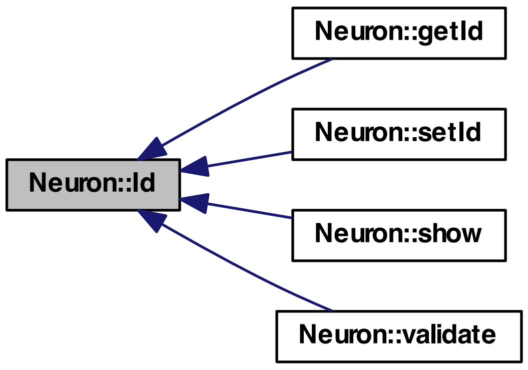


#### 5.19.3.3 std::vector<double> Neuron::getWeight ( )

#### 5.19.3.4 int Neuron::Id ( )

Referenced by getId(), setId(), show(), and validate().

Here is the caller graph for this function:



#### 5.19.3.5 int Neuron::numOfCons ( )

#### 5.19.3.6 bool Neuron::setFrom ( NeuronContainer *neuronContainer* )

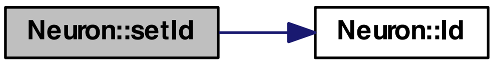
#### 5.19.3.7 void Neuron::setId ( int *value* )

Definition at line 32 of file Neuron.cpp.

References Id().

```
{
  Id = value;
}
```

Here is the call graph for this function:



#### 5.19.3.8 bool Neuron::setWeight ( std::vector< double > *nWeights* )

#### 5.19.3.9 bool Neuron::show ( )

Reimplemented in MLPneuron, and RBFneuron.

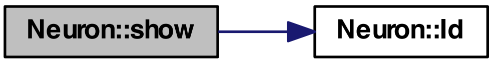
Definition at line 54 of file Neuron.cpp.

References Id().

```
{
  int id = Id();
  Rprintf("\n------------------------\n");
  if (id == NA_INTEGER)
    {
      Rprintf("\n Id: NA, Invalid neuron Id");
    }
  else
    {
      Rprintf("\n Id: %d", id);
    }
  Rprintf("\n------------------------\n");
#if 0

  if (nCons.size() == 0)
    {
      Rprintf("\n No connections defined");
    }
  else
    {
      nCons.show();
    }
  Rprintf("\n------------------------\n");
#endif
}
```

Here is the call graph for this function:



#### 5.19.3.10 bool Neuron::show ( )

Reimplemented in MLPneuron, and RBFneuron.

#### 5.19.3.11 bool Neuron::validate ( )

Reimplemented in MLPneuron, and RBFneuron.

#### 5.19.3.12 bool Neuron::validate ( )

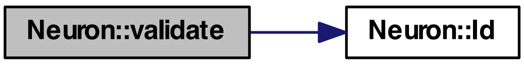
Reimplemented in MLPneuron, and RBFneuron.

Definition at line 82 of file Neuron.cpp.

References Id().

```
{
  BEGIN_RCPP
  if (Id() == NA_INTEGER ) throw std::range_error("[C++ Neuron::validate]: Error,
       Id is NA.");
 // nCons.validate();
  return (TRUE);
END_RCPP}
```

Here is the call graph for this function:



### 5.19.4 Member Data Documentation

#### 5.19.4.1 ConContainer Neuron::con `[private]`

A vector of input connections.

Definition at line 29 of file Neuron.h.

#### 5.19.4.2 int Neuron::d_Id `[protected]`

Definition at line 6 of file Neuron.h.

#### 5.19.4.3 int Neuron::Id `[private]`

An integer variable with the Neuron Id.

The Neuron Id provides a name to the neuron. This value is not expected to be used neither during simulation nor training but it provides an easy reference for human readers.

Definition at line 22 of file Neuron.h.

#### 5.19.4.4 double Neuron::outputValue `[private]`

Definition at line 30 of file Neuron.h.

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

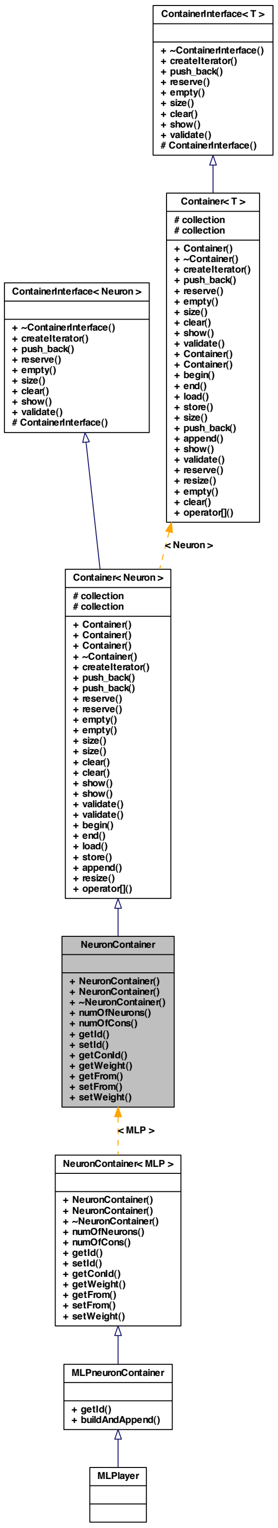
- pkg/AMORE/src/dia/Neuron.h
- pkg/AMORE/src/old/Neuron.h
- pkg/AMORE/src/Neuron.cpp
- pkg/AMORE/src/old/Neuron.cpp

## 5.20 NeuronContainer Class Reference

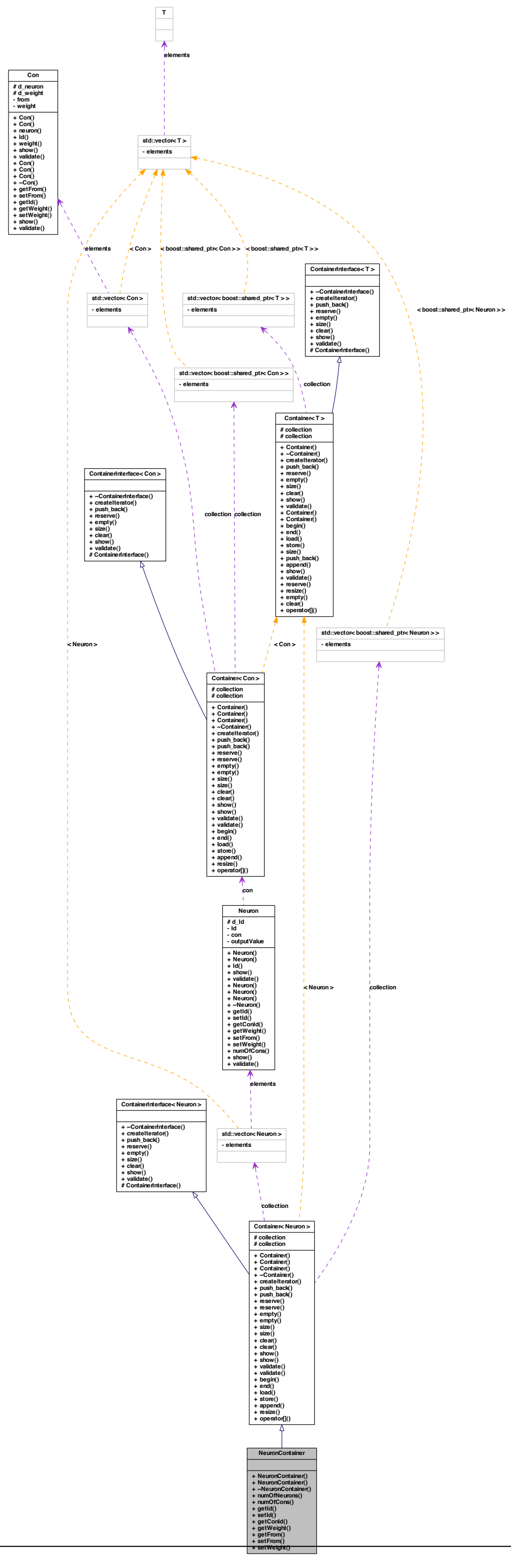
A vector of neurons.

```
#include <NeuronContainer.h>
```

Inheritance diagram for NeuronContainer:



Collaboration diagram for NeuronContainer:


T
elements
Con
std::vector< T >
< Con >
< boost::shared_ptr< Con > >
< boost::shared_ptr< T > >
ContainerInterface< T >
< boost::shared_ptr< Neuron > >
std::vector< Con >
std::vector< boost::shared_ptr< T > >
std::vector< boost::shared_ptr< Con > >
collection
Container< T >
ContainerInterface< Con >
std::vector< boost::shared_ptr< Neuron > >
< Neuron >
Container< Con >
con
Neuron
< Neuron >
ContainerInterface< Neuron >
std::vector< Neuron >
Container< Neuron >
NeuronContainer

## Public Types

- typedef NeuronContainer_iterator iterator
- typedef NeuronContainer_const_iterator const_iterator
- typedef boost::shared_ptr< Neuron > value_type
- typedef value_type const & const_reference

## Public Member Functions

- NeuronContainer ()
- NeuronContainer (std::vector< NeuronPtr > neuronContainer)
- ~NeuronContainer ()
- int numOfNeurons ()
- std::vector< int > numOfCons ()
- std::vector< int > getId ()
- void setId (std::vector< int > nIds)
- std::vector< std::vector< int > > getConId ()
- std::vector< std::vector< double > > getWeight ()
- std::vector< NeuronContainer > getFrom ()
- void setFrom (std::vector< NeuronContainer > neuronArray)
- void setWeight (std::vector< std::vector< double > > value)

### 5.20.1 Detailed Description

A vector of neurons.

The vecNeuron class provides a simple class for a vector of neurons. It's named after the R equivalent Reference Class.

Definition at line 17 of file NeuronContainer.h.

### 5.20.2 Member Typedef Documentation

#### 5.20.2.1 typedef NeuronContainer_const_iterator **NeuronContainer::const_iterator**

Reimplemented from Container< Neuron >.

Definition at line 23 of file NeuronContainer.h.

#### 5.20.2.2 typedef **value_type** const& **NeuronContainer::const_reference**

Reimplemented from Container< Neuron >.

Definition at line 27 of file NeuronContainer.h.

#### 5.20.2.3 typedef NeuronContainer_iterator NeuronContainer::iterator

Reimplemented from Container< Neuron >.

Definition at line 21 of file NeuronContainer.h.

#### 5.20.2.4 typedef boost::shared_ptr<Neuron> NeuronContainer::value_type

Reimplemented from Container< Neuron >.

Definition at line 25 of file NeuronContainer.h.

### 5.20.3 Constructor & Destructor Documentation

#### 5.20.3.1 NeuronContainer::NeuronContainer ( )

Definition at line 8 of file NeuronContainer.cpp.

```
{
}
```

#### 5.20.3.2 NeuronContainer::NeuronContainer ( std::vector< NeuronPtr > *neuronContainer* )

Definition at line 12 of file NeuronContainer.cpp.

```
                                                                                 :
  Container<Neuron> (collection)
{
}
```

#### 5.20.3.3 NeuronContainer::∼NeuronContainer ( )

Definition at line 17 of file NeuronContainer.cpp.

```
{
}
```

### 5.20.4 Member Function Documentation

#### 5.20.4.1 std::vector< std::vector< int > > NeuronContainer::getConId ( )

Definition at line 60 of file NeuronContainer.cpp.

```
{
  std::vector < std::vector<int> > result;
  foreach(NeuronPtr itrNeuron, *this)
```

```
    {
      result.push_back( itrNeuron->getConId() );
    }
  return result;
}
```

#### 5.20.4.2 std::vector<NeuronContainer> NeuronContainer::getFrom ( )

#### 5.20.4.3 std::vector< int > NeuronContainer::getId ( )

Reimplemented in MLPneuronContainer.

Definition at line 39 of file NeuronContainer.cpp.

```
{
  std::vector<int> nIds;
  foreach(NeuronPtr itrNeuron, *this)
    {
      nIds.push_back( itrNeuron->getId() );
    }
  return nIds;
}
```

#### 5.20.4.4 std::vector< std::vector< double > > NeuronContainer::getWeight ( )

Definition at line 71 of file NeuronContainer.cpp.

```
{
  std::vector < std::vector<double> > result;
  foreach(NeuronPtr itrNeuron, *this)
    {
      result.push_back( itrNeuron->getWeight() );
    }
  return result;
}
```

#### 5.20.4.5 std::vector< int > NeuronContainer::numOfCons ( )

Definition at line 28 of file NeuronContainer.cpp.

```
{
  std::vector<int> nIds;
  foreach(NeuronPtr itrNeuron, *this)
    {
      nIds.push_back( itrNeuron->numOfCons() );
    }
  return nIds;
}
```

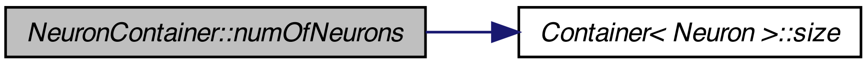
#### 5.20.4.6 int NeuronContainer::numOfNeurons ( )

Definition at line 22 of file NeuronContainer.cpp.

References Container< Neuron >::size().

```
{
  size();
}
```

Here is the call graph for this function:



#### 5.20.4.7 void NeuronContainer::setFrom ( std::vector< NeuronContainer > *neuronArray* )

Definition at line 83 of file NeuronContainer.cpp.

```
{
  std::vector<NeuronContainer>::iterator itrArray(neuronArray.begin());
foreach(NeuronPtr itrNeuron, *this)
  {
    itrNeuron->setFrom(*itrArray);
    itrArray++;
  }
}
```

#### 5.20.4.8 void NeuronContainer::setId ( std::vector< int > *nIds* )

Definition at line 50 of file NeuronContainer.cpp.

```
{
  std::vector<int>::iterator itrId(nIds.begin());
foreach(NeuronPtr itrNeuron, *this)
  {
    itrNeuron->setId(*itrId);
  }
}
```

#### 5.20.4.9 void NeuronContainer::setWeight ( std::vector< std::vector< double > > *value* )

Definition at line 94 of file NeuronContainer.cpp.

```
{
  std::vector<std::vector<double> >::iterator itrValue(value.begin());
foreach(NeuronPtr itrNeuron, *this)
  {
    itrNeuron->setWeight(*itrValue);
    itrValue++;
  }
}
```

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

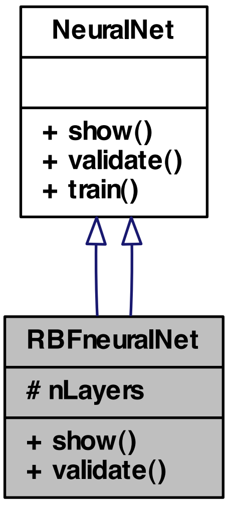
- pkg/AMORE/src/old/NeuronContainer.h
- pkg/AMORE/src/old/NeuronContainer.cpp

## 5.21 RBFneuralNet Class Reference

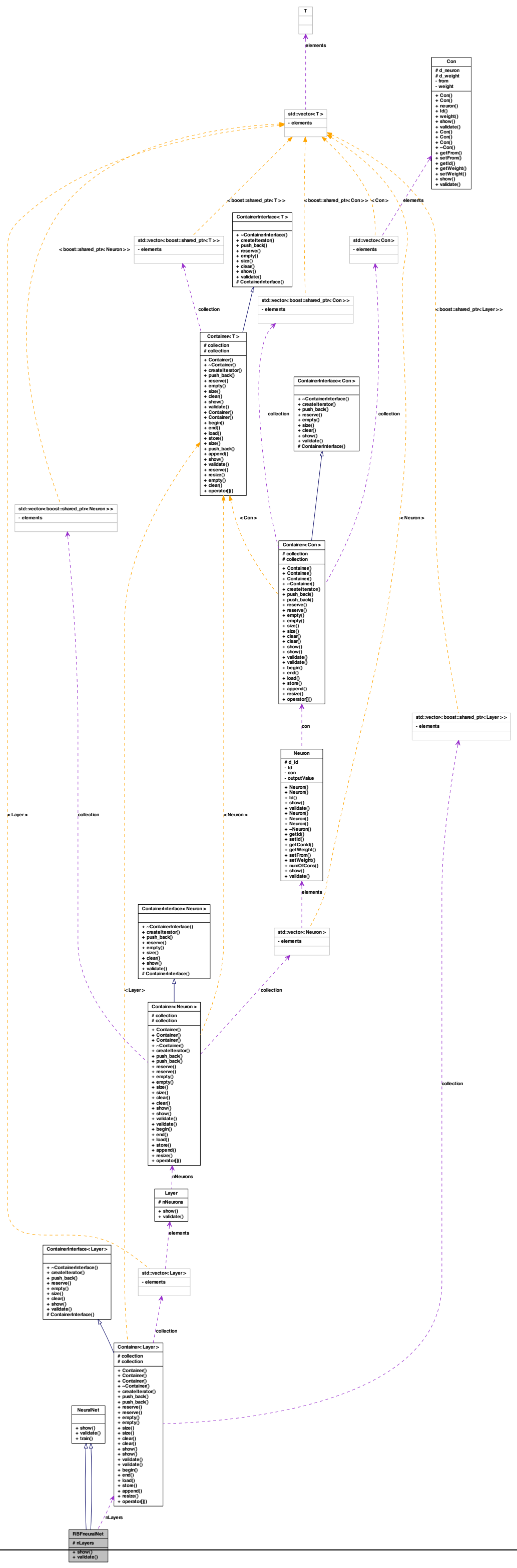
class RBFneuralNet -

`#include <RBFneuralNet.h>`

Inheritance diagram for RBFneuralNet:



Collaboration diagram for RBFneuralNet:



### Public Member Functions

- void show ()
- bool validate ()

### Protected Attributes

- Container< Layer > nLayers

### 5.21.1 Detailed Description

class RBFneuralNet -

Definition at line 5 of file RBFneuralNet.h.

### 5.21.2 Member Function Documentation

#### 5.21.2.1 void RBFneuralNet::show ( )

Reimplemented from NeuralNet.

#### 5.21.2.2 bool RBFneuralNet::validate ( )

Reimplemented from NeuralNet.

### 5.21.3 Member Data Documentation

#### 5.21.3.1 Container<Layer> RBFneuralNet::nLayers `[protected]`

Definition at line 8 of file RBFneuralNet.h.

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

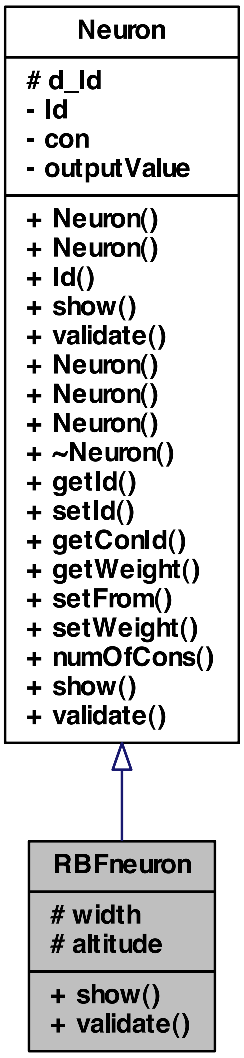
- pkg/AMORE/src/dia/RBFneuralNet.h

## 5.22 RBFneuron Class Reference

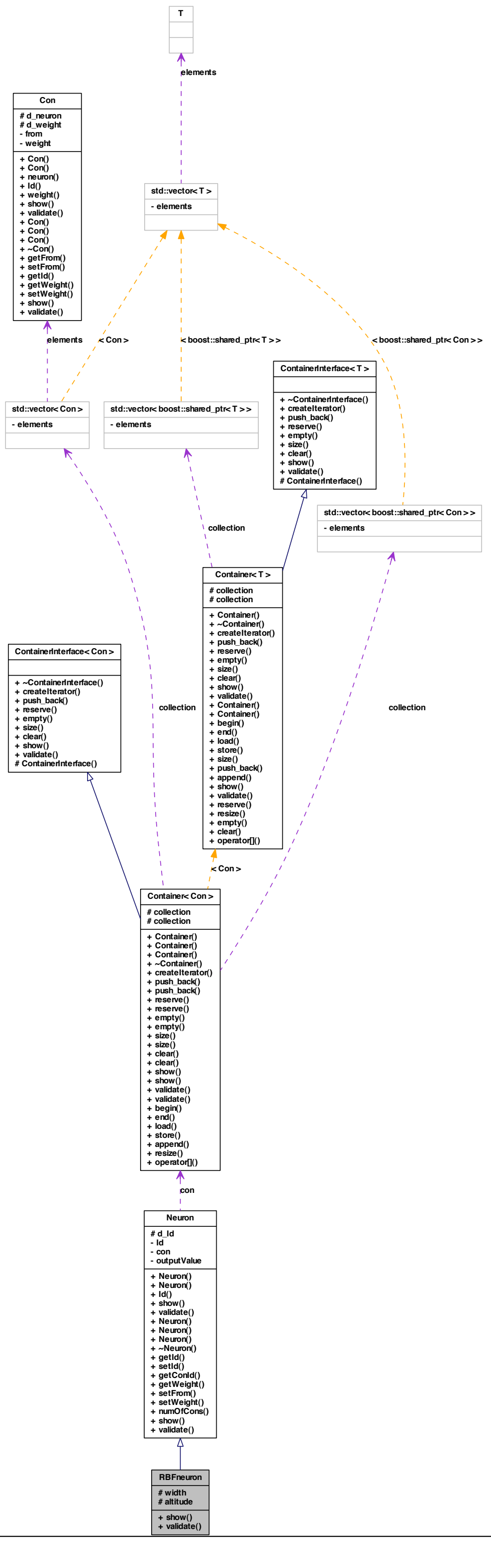
class RBFneuron -

`#include <RBFneuron.h>`

Inheritance diagram for RBFneuron:



Collaboration diagram for RBFneuron:



### Public Member Functions

- void show ()
- bool validate ()

### Protected Attributes

- double width
- double altitude

### 5.22.1 Detailed Description

class RBFneuron -

Definition at line 5 of file RBFneuron.h.

### 5.22.2 Member Function Documentation

#### 5.22.2.1 void RBFneuron::show ( )

Reimplemented from Neuron.

#### 5.22.2.2 bool RBFneuron::validate ( )

Reimplemented from Neuron.

### 5.22.3 Member Data Documentation

#### 5.22.3.1 double **RBFneuron::altitude** `[protected]`

Definition at line 9 of file RBFneuron.h.

#### 5.22.3.2 double **RBFneuron::width** `[protected]`

Definition at line 8 of file RBFneuron.h.

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

- pkg/AMORE/src/dia/RBFneuron.h

## 5.23 SimulationVariables Class Reference

class SimulationVariables -

`#include <SimulationVariables.h>`

### Protected Attributes

- double outputValue

### 5.23.1 Detailed Description

class SimulationVariables -

Definition at line 3 of file SimulationVariables.h.

### 5.23.2 Member Data Documentation

#### 5.23.2.1 double SimulationVariables::outputValue `[protected]`

Definition at line 6 of file SimulationVariables.h.

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

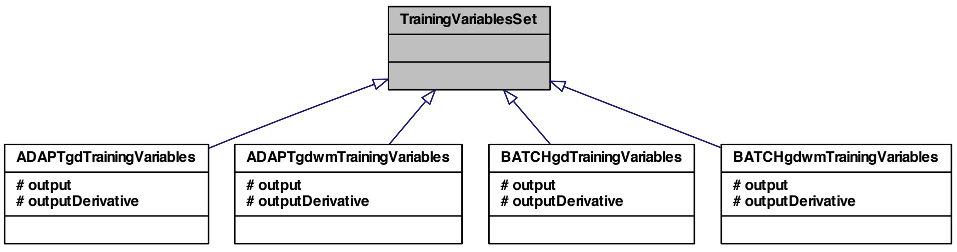
- pkg/AMORE/src/dia/SimulationVariables.h

## 5.24 TrainingVariablesSet Class Reference

class TrainingVariablesSet -

`#include <TrainingVariablesSet.h>`

Inheritance diagram for TrainingVariablesSet:



### 5.24.1 Detailed Description

class TrainingVariablesSet -

Definition at line 3 of file TrainingVariablesSet.h.

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

- pkg/AMORE/src/dia/TrainingVariablesSet.h

# Chapter 6

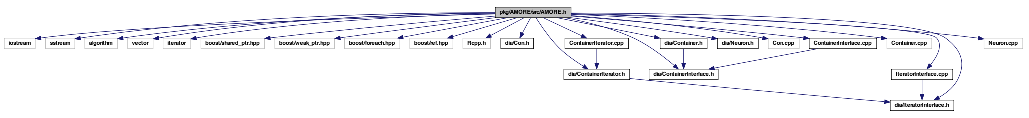
# File Documentation

## 6.1 pkg/AMORE/src/AMORE.h File Reference

```
#include <iostream>
#include <sstream>
#include <algorithm>
#include <vector>
#include <iterator>
#include <boost/shared_ptr.hpp>
#include <boost/weak_ptr.hpp>
#include <boost/foreach.hpp>
#include <boost/ref.hpp>
#include <Rcpp.h>
#include "dia/Con.h"
#include "dia/IteratorInterface.h"
#include "dia/ContainerInterface.h"
#include "dia/Container.h"
#include "dia/ContainerIterator.h"
#include "dia/Neuron.h"
#include "Con.cpp"
#include "ContainerInterface.cpp"
#include "Container.cpp"
#include "IteratorInterface.cpp"
#include "ContainerIterator.cpp"
```

```
#include "Neuron.cpp"
```

Include dependency graph for AMORE.h:



## Defines

- #define foreach BOOST_FOREACH
- #define size_type unsigned int

## Typedefs

- typedef boost::reference_wrapper< Neuron > NeuronRef
- typedef boost::shared_ptr< Neuron > NeuronPtr
- typedef boost::shared_ptr< Con > ConPtr
- typedef Container< Con > ConContainer
- typedef Container< Neuron > NeuronContainer

## 6.1.1 Define Documentation

### 6.1.1.1 #define foreach BOOST_FOREACH

Definition at line 53 of file AMORE.h.

### 6.1.1.2 #define size_type unsigned int

Definition at line 58 of file AMORE.h.

## 6.1.2 Typedef Documentation

### 6.1.2.1 typedef Container<Con> ConContainer

Definition at line 74 of file AMORE.h.

### 6.1.2.2 typedef boost::shared_ptr<Con> ConPtr

Definition at line 63 of file AMORE.h.

#### 6.1.2.3 typedef Container<Neuron> NeuronContainer

Definition at line 75 of file AMORE.h.

#### 6.1.2.4 typedef boost::shared_ptr<Neuron> NeuronPtr

Definition at line 62 of file AMORE.h.

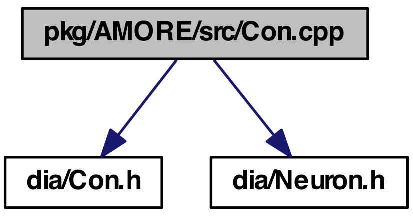
#### 6.1.2.5 typedef boost::reference_wrapper<Neuron> NeuronRef

Definition at line 61 of file AMORE.h.

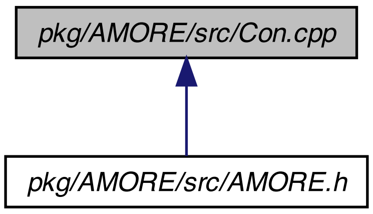
## 6.2 pkg/AMORE/src/Con.cpp File Reference

```
#include "dia/Con.h"
#include "dia/Neuron.h"
```

Include dependency graph for Con.cpp:



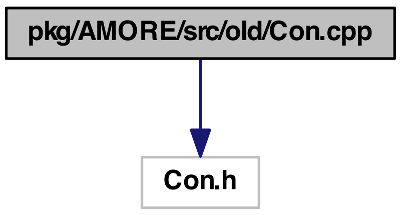
This graph shows which files directly or indirectly include this file:



## 6.3 pkg/AMORE/src/old/Con.cpp File Reference

```
#include "Con.h"
```

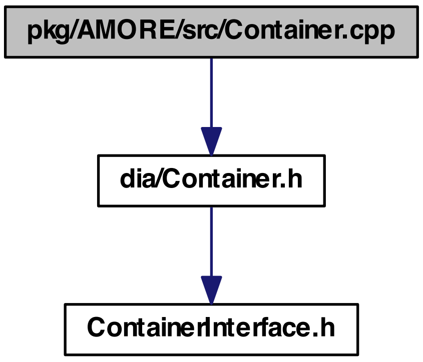
Include dependency graph for Con.cpp:



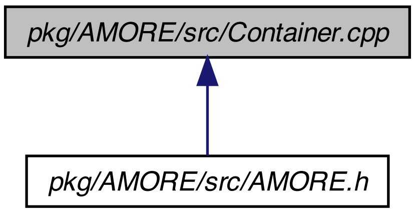
## 6.4 pkg/AMORE/src/Container.cpp File Reference

```
#include "dia/Container.h"
```

Include dependency graph for Container.cpp:



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

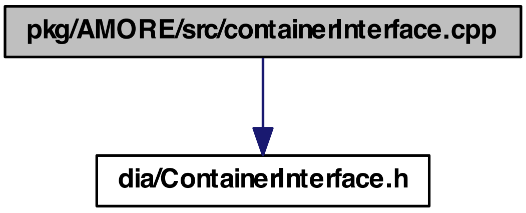


## 6.5 pkg/AMORE/src/old/Container.cpp File Reference

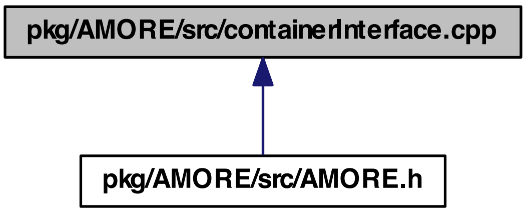
## 6.6 pkg/AMORE/src/containerInterface.cpp File Reference

```
#include "dia/ContainerInterface.h"
```

Include dependency graph for containerInterface.cpp:



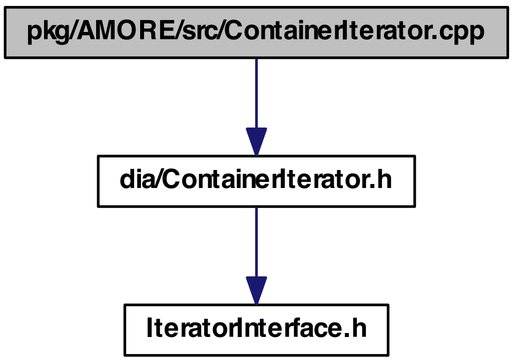
This graph shows which files directly or indirectly include this file:



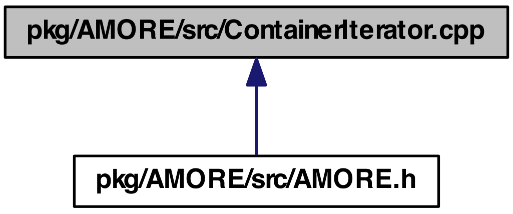
## 6.7 pkg/AMORE/src/ContainerIterator.cpp File Reference

```
#include "dia/ContainerIterator.h"
```

Include dependency graph for ContainerIterator.cpp:



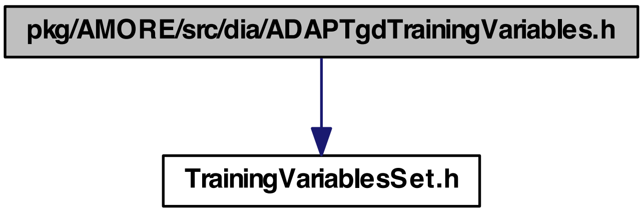
This graph shows which files directly or indirectly include this file:



## 6.8 pkg/AMORE/src/dia/ADAPTgdTrainingVariables.h File Reference

```
#include "TrainingVariablesSet.h"
```

Include dependency graph for ADAPTgdTrainingVariables.h:



### Classes

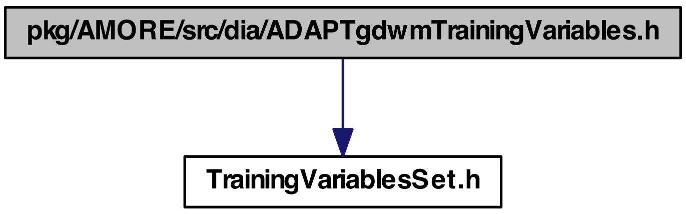
- class ADAPTgdTrainingVariables

  *class ADAPTgdTrainingVariables -*

## 6.9 pkg/AMORE/src/dia/ADAPTgdwmTrainingVariables.h File Reference

```
#include "TrainingVariablesSet.h"
```

Include dependency graph for ADAPTgdwmTrainingVariables.h:



### Classes

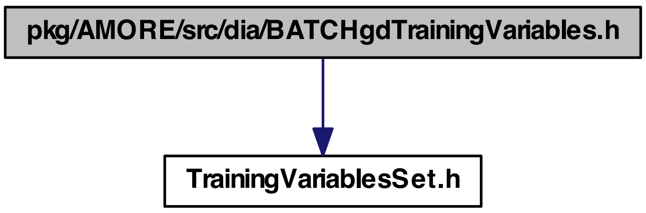
- class ADAPTgdwmTrainingVariables

*class ADAPTgdwmTrainingVariables -*

## 6.10 pkg/AMORE/src/dia/BATCHgdTrainingVariables.h File Reference

```
#include "TrainingVariablesSet.h"
```

Include dependency graph for BATCHgdTrainingVariables.h:



### Classes

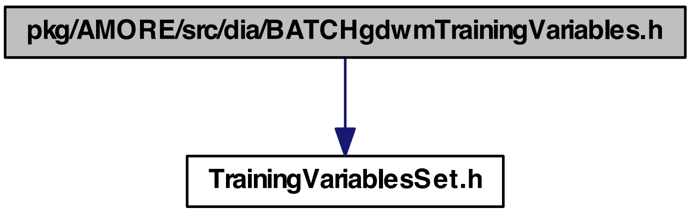
- class BATCHgdTrainingVariables

  *class BATCHgdTrainingVariables -*

## 6.11 pkg/AMORE/src/dia/BATCHgdwmTrainingVariables.h File Reference

```
#include "TrainingVariablesSet.h"
```

Include dependency graph for BATCHgdwmTrainingVariables.h:



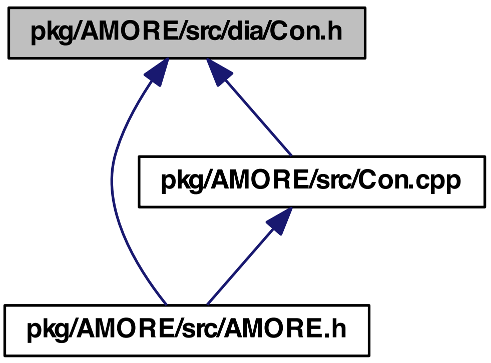
### Classes

- class BATCHgdwmTrainingVariables

  *class BATCHgdwmTrainingVariables -*

## 6.12 pkg/AMORE/src/dia/Con.h File Reference

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



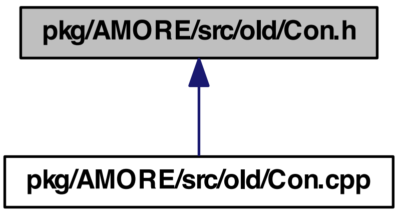
### Classes

- class Con

  *class Con -*

## 6.13 pkg/AMORE/src/old/Con.h File Reference

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



### Classes

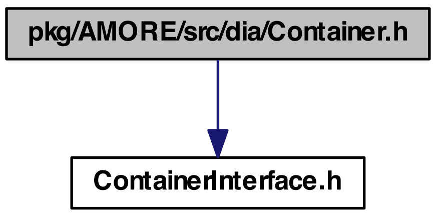
- class Con

  *class Con -*

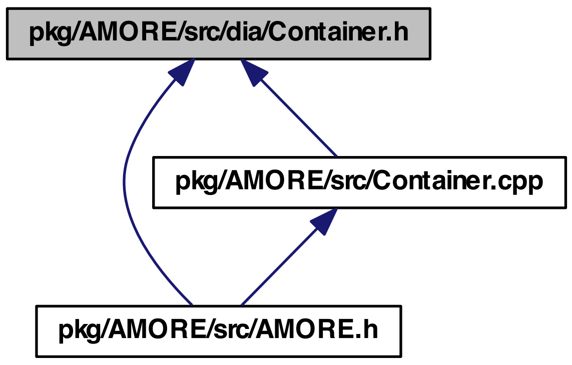
## 6.14 pkg/AMORE/src/dia/Container.h File Reference

```
#include "ContainerInterface.h"
```

Include dependency graph for Container.h:



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



## Classes

- class Container< T >

    *class Container -*

## 6.15 pkg/AMORE/src/old/Container.h File Reference

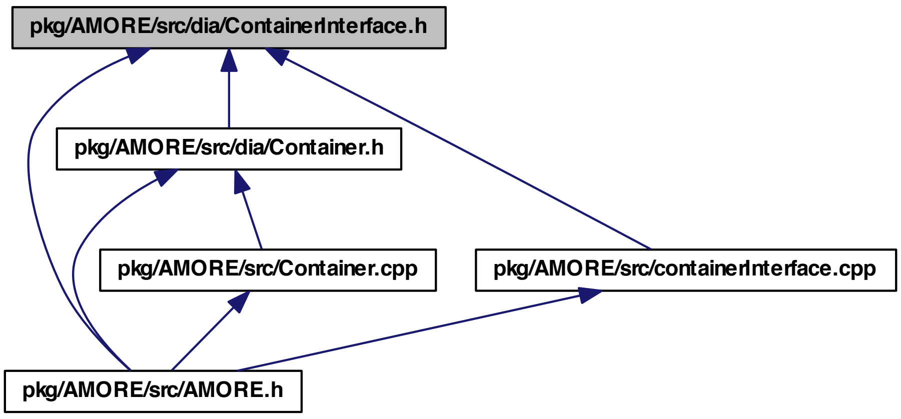
### Classes

- class Container< T >

  *class Container -*

## 6.16 pkg/AMORE/src/dia/ContainerInterface.h File Reference

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


pkg/AMORE/src/dia/ContainerInterface.h
pkg/AMORE/src/dia/Container.h
pkg/AMORE/src/Container.cpp
pkg/AMORE/src/containerInterface.cpp
pkg/AMORE/src/AMORE.h

### Classes

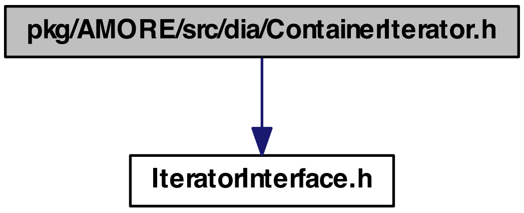
- class ContainerInterface< T >

  *class ContainerInterface -*

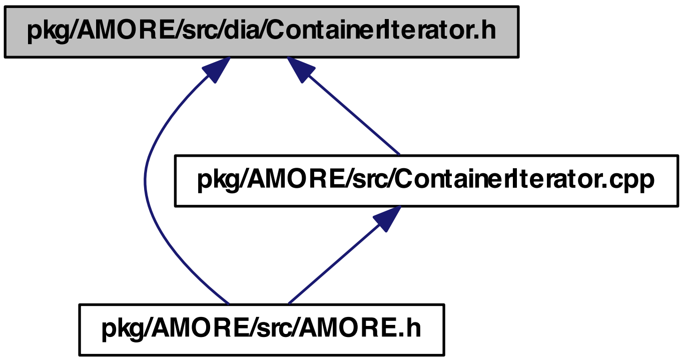
## 6.17 pkg/AMORE/src/dia/ContainerIterator.h File Reference

```
#include "IteratorInterface.h"
```

Include dependency graph for ContainerIterator.h:



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


pkg/AMORE/src/dia/ContainerIterator.h
pkg/AMORE/src/ContainerIterator.cpp
pkg/AMORE/src/AMORE.h

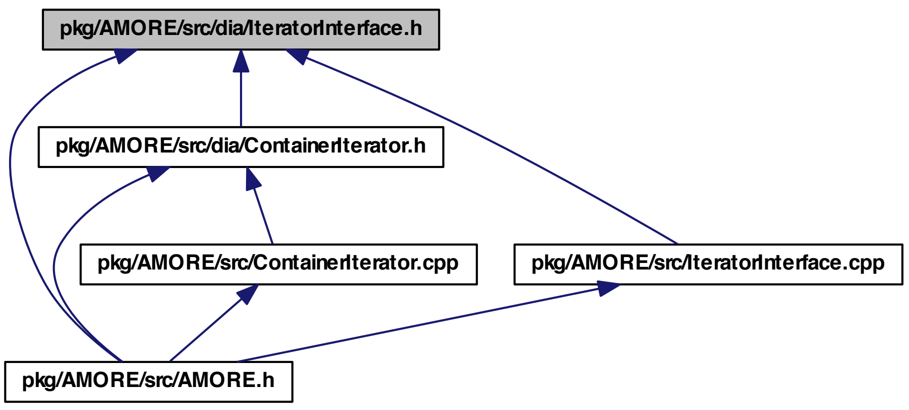
## Classes

- class ContainerIterator< T >

  *class ContainerIterator -*

## 6.18 pkg/AMORE/src/dia/IteratorInterface.h File Reference

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


pkg/AMORE/src/dia/IteratorInterface.h
pkg/AMORE/src/dia/ContainerIterator.h
pkg/AMORE/src/ContainerIterator.cpp
pkg/AMORE/src/IteratorInterface.cpp
pkg/AMORE/src/AMORE.h

### Classes

- class IteratorInterface< T >

  *class IteratorInterface -*

## 6.19 pkg/AMORE/src/dia/Layer.h File Reference

### Classes

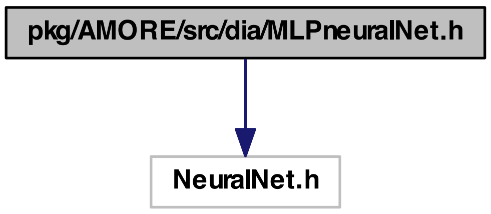
- class Layer

  *class Layer -*

## 6.20 pkg/AMORE/src/dia/MLPneuralNet.h File Reference

```
#include "NeuralNet.h"
```

Include dependency graph for MLPneuralNet.h:



### Classes

- class MLPneuralNet

  *class MLPneuralNet -*

## 6.21 pkg/AMORE/src/old/MLPneuralNet.h File Reference

### Classes

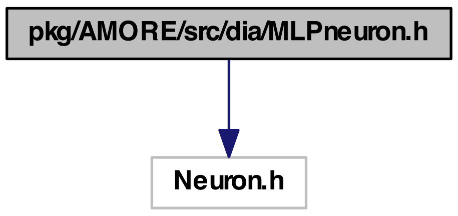
- class MLPneuralNet

  *class MLPneuralNet -*

## 6.22 pkg/AMORE/src/dia/MLPneuron.h File Reference

```
#include "Neuron.h"
```

Include dependency graph for MLPneuron.h:



### Classes

- class MLPneuron

    *class MLPneuron -*

## 6.23 pkg/AMORE/src/old/MLPneuron.h File Reference

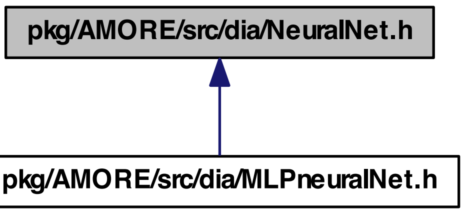
### Classes

- class MLPneuron

    *class MLPneuron -*

## 6.24 pkg/AMORE/src/dia/NeuralNet.h File Reference

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


pkg/AMORE/src/dia/NeuralNet.h
pkg/AMORE/src/dia/MLPneuralNet.h

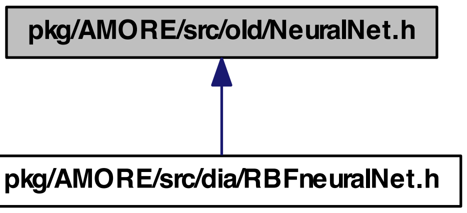
### Classes

- class NeuralNet

  *class NeuralNet -*

## 6.25 pkg/AMORE/src/old/NeuralNet.h File Reference

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


pkg/AMORE/src/old/NeuralNet.h
pkg/AMORE/src/dia/RBFneuralNet.h

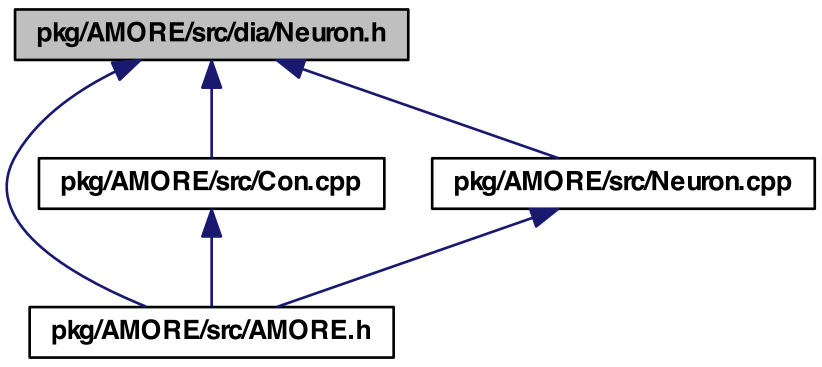
### Classes

- class NeuralNet

*class NeuralNet -*

## 6.26 pkg/AMORE/src/dia/Neuron.h File Reference

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



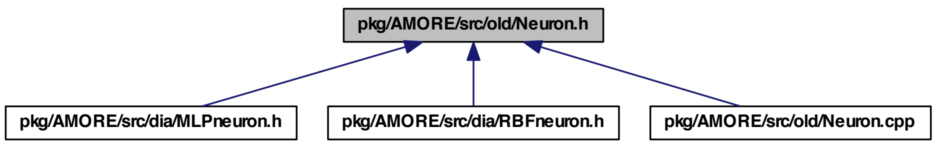
### Classes

- class Neuron

  *class Neuron -*

## 6.27 pkg/AMORE/src/old/Neuron.h File Reference

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


pkg/AMORE/src/old/Neuron.h
pkg/AMORE/src/dia/MLPneuron.h
pkg/AMORE/src/dia/RBFneuron.h
pkg/AMORE/src/old/Neuron.cpp

### Classes

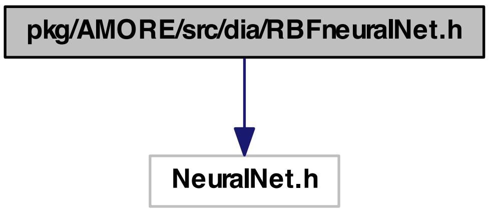
- class Neuron

  *class Neuron -*

## 6.28 pkg/AMORE/src/dia/RBFneuralNet.h File Reference

```
#include "NeuralNet.h"
```

Include dependency graph for RBFneuralNet.h:



### Classes

- class RBFneuralNet

  *class RBFneuralNet -*

## 6.29 pkg/AMORE/src/old/RBFneuralNet.h File Reference

### Classes

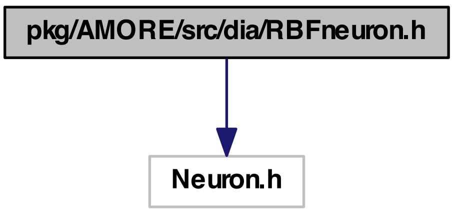
- class RBFneuralNet

  *class RBFneuralNet -*

## 6.30 pkg/AMORE/src/dia/RBFneuron.h File Reference

```
#include "Neuron.h"
```

Include dependency graph for RBFneuron.h:



### Classes

- class RBFneuron

  *class RBFneuron -*

## 6.31 pkg/AMORE/src/dia/SimulationVariables.h File Reference

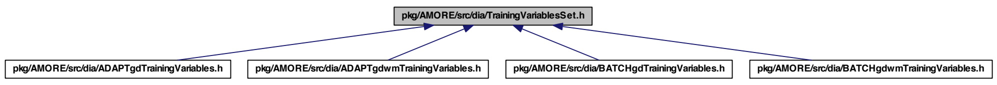
### Classes

- class SimulationVariables

  *class SimulationVariables -*

## 6.32 pkg/AMORE/src/dia/TrainingVariablesSet.h File Reference

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


pkg/AMORE/src/dia/TrainingVariablesSet.h
pkg/AMORE/src/dia/ADAPTgdTrainingVariables.h
pkg/AMORE/src/dia/ADAPTgdwmTrainingVariables.h
pkg/AMORE/src/dia/BATCHgdTrainingVariables.h
pkg/AMORE/src/dia/BATCHgdwmTrainingVariables.h

### Classes

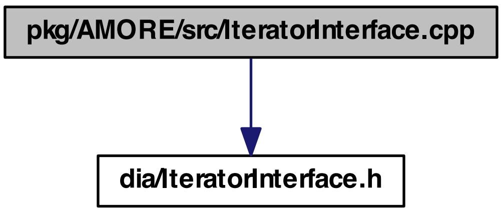
- class TrainingVariablesSet

  *class TrainingVariablesSet -*

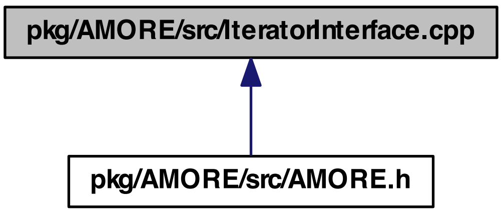
## 6.33 pkg/AMORE/src/IteratorInterface.cpp File Reference

```
#include "dia/IteratorInterface.h"
```

Include dependency graph for IteratorInterface.cpp:



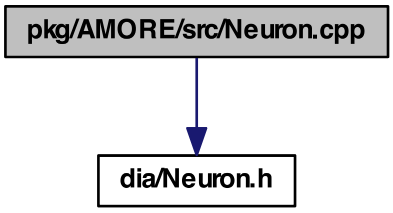
This graph shows which files directly or indirectly include this file:



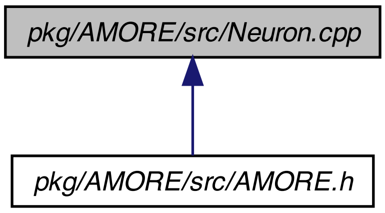
## 6.34 pkg/AMORE/src/Neuron.cpp File Reference

```
#include "dia/Neuron.h"
```

Include dependency graph for Neuron.cpp:



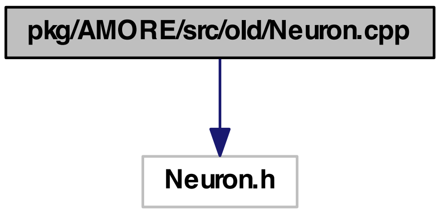
This graph shows which files directly or indirectly include this file:



## 6.35 pkg/AMORE/src/old/Neuron.cpp File Reference

```
#include "Neuron.h"
```

Include dependency graph for Neuron.cpp:



## 6.36 pkg/AMORE/src/old/ConContainer.cpp File Reference

### Classes

- struct CompareId

## 6.37 pkg/AMORE/src/old/ConContainer.h File Reference

### Classes

- class ConContainer

    *A vector of connections.*

## 6.38 pkg/AMORE/src/old/MLPlayer.h File Reference

### Classes

- class MLPlayer

## 6.39 pkg/AMORE/src/old/MLPlayerContainer.h File Reference

### Classes

- class MLPlayerContainer

## 6.40 pkg/AMORE/src/old/MLPneuralNetFactory.cpp File Reference

### Functions

- MLPneuralNet CreateMLPneuralNet (std::vector< int > numberOfNeuronsPerLayer)

### 6.40.1 Function Documentation

#### 6.40.1.1 **MLPneuralNet** CreateMLPneuralNet ( std::vector< int > *numberOfNeuronsPerLayer* )

Definition at line 2 of file MLPneuralNetFactory.cpp.

```
{

  net = new MLPNeuralNet();

  MLPlayerPtr mlpLayerPtr;

  std::vector<int> idx;

foreach (int n, numberOfNeuronsPerLayer)
  {
    for (int i=1; i<=n; ++i)
      {
        idx.push_back(i);
      }
    mlpLayerPtr.reset(new MLPlayer( idx ) );
    net.nLayers.push_back(mlpLayerPtr);
  }



for (int i=1; i<=; ++i)
  {

    mlpPtr->buildAndAppend();

  }

}
```

## 6.41 pkg/AMORE/src/old/MLPneuronContainer.h File Reference

### Classes

- class MLPneuronContainer

  *A vector of connections.*

## 6.42 pkg/AMORE/src/old/NeuronContainer.cpp File Reference

## 6.43 pkg/AMORE/src/old/NeuronContainer.h File Reference

### Classes

- class NeuronContainer

  *A vector of neurons.*

# Index