

AMORE++

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

Generated by Doxygen 1.7.4

Sat Jun 11 2011 22:28:02

Contents

1	The AMORE++ package	1
1.1	Introduction	1
1.2	Motivation	1
1.3	Road Map	1
2	Class Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Class Documentation	9
5.1	CompareId Struct Reference	9
5.1.1	Detailed Description	9
5.1.2	Member Function Documentation	9
5.1.2.1	operator()	9
5.1.2.2	operator()	9
5.1.2.3	operator()	10
5.1.2.4	operator()	10
5.2	Con Class Reference	10
5.2.1	Detailed Description	11
5.2.2	Constructor & Destructor Documentation	11
5.2.2.1	Con	11
5.2.2.2	Con	11

5.2.2.3	Con	12
5.2.2.4	~Con	12
5.2.3	Member Function Documentation	12
5.2.3.1	getFrom	12
5.2.3.2	getId	13
5.2.3.3	getWeight	14
5.2.3.4	setFrom	15
5.2.3.5	setWeight	16
5.2.3.6	show	16
5.2.3.7	validate	17
5.2.4	Member Data Documentation	18
5.2.4.1	from	18
5.2.4.2	weight	18
5.3	ConContainer Class Reference	19
5.3.1	Detailed Description	22
5.3.2	Member Typedef Documentation	22
5.3.2.1	const_iterator	22
5.3.2.2	const_reference	23
5.3.2.3	iterator	23
5.3.2.4	value_type	23
5.3.3	Constructor & Destructor Documentation	23
5.3.3.1	ConContainer	23
5.3.3.2	ConContainer	23
5.3.4	Member Function Documentation	23
5.3.4.1	erase	23
5.3.4.2	getId	25
5.3.4.3	numOfCons	27
5.3.4.4	select	28
5.3.4.5	setFrom	30
5.3.4.6	setWeight	32
5.3.4.7	setWeight	34
5.3.4.8	validate	36
5.4	Container< T > Class Template Reference	37
5.4.1	Detailed Description	40

5.4.2	Member Typedef Documentation	41
5.4.2.1	const_iterator	41
5.4.2.2	const_reference	41
5.4.2.3	iterator	41
5.4.2.4	value_type	41
5.4.3	Constructor & Destructor Documentation	41
5.4.3.1	Container	41
5.4.3.2	Container	41
5.4.4	Member Function Documentation	42
5.4.4.1	append	42
5.4.4.2	begin	43
5.4.4.3	clear	44
5.4.4.4	empty	44
5.4.4.5	end	45
5.4.4.6	load	45
5.4.4.7	operator[]	46
5.4.4.8	push_back	46
5.4.4.9	reserve	47
5.4.4.10	resize	47
5.4.4.11	show	48
5.4.4.12	size	49
5.4.4.13	store	50
5.4.4.14	validate	50
5.4.5	Member Data Documentation	50
5.4.5.1	collection	50
5.5	MLPlayer Class Reference	51
5.5.1	Detailed Description	54
5.6	MLPlayerContainer Class Reference	54
5.6.1	Detailed Description	57
5.7	MLPneuralNet Class Reference	57
5.7.1	Detailed Description	59
5.7.2	Member Data Documentation	59
5.7.2.1	nLayers	59
5.8	MLPneuron Class Reference	59

5.8.1	Detailed Description	62
5.8.2	Member Data Documentation	62
5.8.2.1	bias	62
5.9	MLPneuronContainer Class Reference	62
5.9.1	Detailed Description	65
5.9.2	Member Function Documentation	65
5.9.2.1	buildAndAppend	65
5.9.2.2	getId	65
5.10	NeuralNet Class Reference	65
5.10.1	Detailed Description	66
5.10.2	Member Function Documentation	66
5.10.2.1	train	66
5.11	Neuron Class Reference	66
5.11.1	Detailed Description	69
5.11.2	Constructor & Destructor Documentation	69
5.11.2.1	Neuron	69
5.11.2.2	Neuron	70
5.11.2.3	Neuron	70
5.11.2.4	~Neuron	70
5.11.3	Member Function Documentation	70
5.11.3.1	getConId	70
5.11.3.2	getId	71
5.11.3.3	getWeight	71
5.11.3.4	numOfCons	71
5.11.3.5	setFrom	72
5.11.3.6	setId	73
5.11.3.7	setWeight	73
5.11.3.8	show	73
5.11.3.9	validate	74
5.11.4	Member Data Documentation	74
5.11.4.1	con	75
5.11.4.2	id	75
5.11.4.3	outputValue	75
5.12	NeuronContainer Class Reference	75

5.12.1	Detailed Description	78
5.12.2	Member Typedef Documentation	78
5.12.2.1	const_iterator	78
5.12.2.2	const_reference	78
5.12.2.3	iterator	79
5.12.2.4	value_type	79
5.12.3	Constructor & Destructor Documentation	79
5.12.3.1	NeuronContainer	79
5.12.3.2	NeuronContainer	79
5.12.3.3	~NeuronContainer	79
5.12.4	Member Function Documentation	79
5.12.4.1	getConId	79
5.12.4.2	getFrom	80
5.12.4.3	getId	80
5.12.4.4	getWeight	80
5.12.4.5	numOfCons	80
5.12.4.6	numOfNeurons	81
5.12.4.7	setFrom	81
5.12.4.8	setId	81
5.12.4.9	setWeight	82
5.13	RBFneuralNet Class Reference	82
5.13.1	Detailed Description	83
6	File Documentation	85
6.1	pkg/AMORE/src/AMORE.h File Reference	85
6.1.1	Define Documentation	86
6.1.1.1	foreach	86
6.1.1.2	size_type	86
6.1.2	Typedef Documentation	86
6.1.2.1	ConContainerPtr	86
6.1.2.2	ConPtr	86
6.1.2.3	ContainerConPtr	87
6.1.2.4	ContainerNeuronPtr	87
6.1.2.5	NeuronContainer_const_iterator	87

6.1.2.6	NeuronContainer_iterator	87
6.1.2.7	NeuronContainerPtr	87
6.1.2.8	NeuronPtr	87
6.1.2.9	NeuronWeakPtr	87
6.2	pkg/AMORE/src/Con.cpp File Reference	87
6.3	pkg/AMORE/src/Con.h File Reference	89
6.4	pkg/AMORE/src/ConContainer.cpp File Reference	89
6.5	pkg/AMORE/src/ConContainer.h File Reference	90
6.6	pkg/AMORE/src/Container.cpp File Reference	90
6.7	pkg/AMORE/src/Container.h File Reference	91
6.8	pkg/AMORE/src/MLPlayer.h File Reference	91
6.9	pkg/AMORE/src/MLPlayerContainer.h File Reference	91
6.10	pkg/AMORE/src/MLPneuralNet.h File Reference	91
6.11	pkg/AMORE/src/MLPneuralNetFactory.cpp File Reference	92
6.12	pkg/AMORE/src/MLPneuron.h File Reference	92
6.13	pkg/AMORE/src/MLPneuronContainer.h File Reference	92
6.14	pkg/AMORE/src/NeuralNet.h File Reference	92
6.15	pkg/AMORE/src/Neuron.cpp File Reference	92
6.16	pkg/AMORE/src/Neuron.h File Reference	93
6.17	pkg/AMORE/src/NeuronContainer.cpp File Reference	94
6.18	pkg/AMORE/src/NeuronContainer.h File Reference	94
6.19	pkg/AMORE/src/RBFneuralNet.h File Reference	94

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:

CompareId	9
Con	10
Container< T >	37
Container< Con >	37
ConContainer	19
Container< MLPlayer >	37
MLPlayerContainer	54
Container< Neuron >	37
NeuronContainer	75
MLPneuronContainer	62
MLPlayer	51
NeuralNet	65
MLPneuralNet	57
RBFneuralNet	82
Neuron	66
MLPneuron	59

Chapter 3

Class Index

3.1 Class List

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

CompareId	9
Con (A class to handle the information needed to describe an input connection)	10
ConContainer (A vector of connections)	19
Container< T >	37
MLPlayer	51
MLPlayerContainer	54
MLPneuralNet	57
MLPneuron	59
MLPneuronContainer (A vector of connections)	62
NeuralNet	65
Neuron (A class to handle the information contained in a general Neuron)	66
NeuronContainer (A vector of neurons)	75
RBFneuralNet	82

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

pkg/AMORE/src/AMORE.h	85
pkg/AMORE/src/Con.cpp	87
pkg/AMORE/src/Con.h	89
pkg/AMORE/src/ConContainer.cpp	89
pkg/AMORE/src/ConContainer.h	90
pkg/AMORE/src/Container.cpp	90
pkg/AMORE/src/Container.h	91
pkg/AMORE/src/MLPlayer.h	91
pkg/AMORE/src/MLPlayerContainer.h	91
pkg/AMORE/src/MLPneuralNet.h	91
pkg/AMORE/src/MLPneuralNetFactory.cpp	92
pkg/AMORE/src/MLPneuron.h	92
pkg/AMORE/src/MLPneuronContainer.h	92
pkg/AMORE/src/NeuralNet.h	92
pkg/AMORE/src/Neuron.cpp	92
pkg/AMORE/src/Neuron.h	93
pkg/AMORE/src/NeuronContainer.cpp	94
pkg/AMORE/src/NeuronContainer.h	94
pkg/AMORE/src/RBFneuralNet.h	94

Chapter 5

Class Documentation

5.1 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`)

5.1.1 Detailed Description

Definition at line 352 of file `ConContainer.cpp`.

5.1.2 Member Function Documentation

5.1.2.1 `bool CompareId::operator() (const ConPtr a, const ConPtr b)` `[inline]`

Definition at line 356 of file `ConContainer.cpp`.

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

5.1.2.2 `bool CompareId::operator() (const int a, const int b)` `[inline]`

Definition at line 377 of file `ConContainer.cpp`.

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

5.1.2.3 `bool CompareId::operator() (const int a, const ConPtr b)` `[inline]`

Definition at line 370 of file ConContainer.cpp.

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

5.1.2.4 `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/ConContainer.cpp](#)

5.2 Con Class Reference

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

```
#include <Con.h>
```

Public Member Functions

- [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](#) ()
Pretty print of the [Con](#) information.
- bool [validate](#) ()
Object validator.

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.

5.2.1 Detailed Description

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 16 of file [Con.h](#).

5.2.2 Constructor & Destructor Documentation

5.2.2.1 [Con::Con \(\)](#)

Default Constructor.

Definition at line 17 of file [Con.cpp](#).

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

5.2.2.2 [Con::Con \(\[NeuronPtr\]\(#\) *neuronPtr* \)](#)

Constructor.

Definition at line 40 of file [Con.cpp](#).

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

5.2.2.3 Con::Con (NeuronPtr neuronPtr, double value)

Constructor.

Definition at line 29 of file Con.cpp.

```

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

```

5.2.2.4 Con::~Con ()

Default Destructor.

Definition at line 46 of file Con.cpp.

```

{
}

```

5.2.3 Member Function Documentation

5.2.3.1 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());
}
```

5.2.3.2 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.

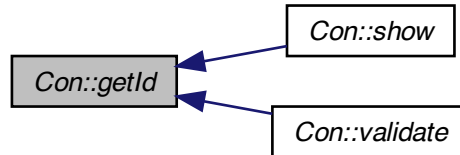
Definition at line 123 of file `Con.cpp`.

References from.

Referenced by `show()`, and `validate()`.

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

Here is the caller graph for this function:



5.2.3.3 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](#).

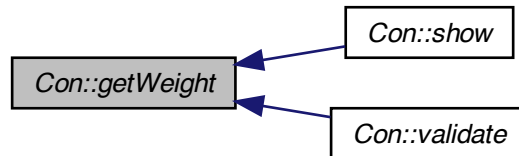
Referenced by `show()`, and `validate()`.

```

{
    return (weight);
}

```

Here is the caller graph for this function:



5.2.3.4 void Con::setFrom (NeuronPtr neuronPtr)

from field accessor.

This method sets the value of the [from](#) field with the address used as parameter.

Parameters

<i>f</i>	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;
}
  
```

5.2.3.5 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;
}
```

5.2.3.6 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

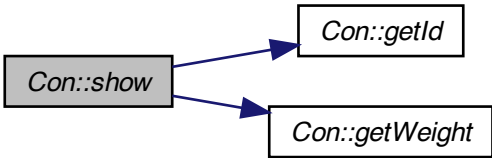
[setWeight](#) and the unit test files, e.g., runit.Cpp.Con.R, for usage examples.

Definition at line 197 of file Con.cpp.

References [getId\(\)](#), and [getWeight\(\)](#).


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

Here is the call graph for this function:



5.2.3.7 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.
----	--

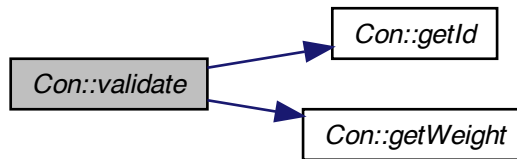
Definition at line 218 of file Con.cpp.

References getId(), and getWeight().

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

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

Here is the call graph for this function:



5.2.4 Member Data Documentation

5.2.4.1 `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()`.

5.2.4.2 `double Con::weight` [private]

A double variable that contains the weight of the connection.

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.

Definition at line 27 of file `Con.h`.

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

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

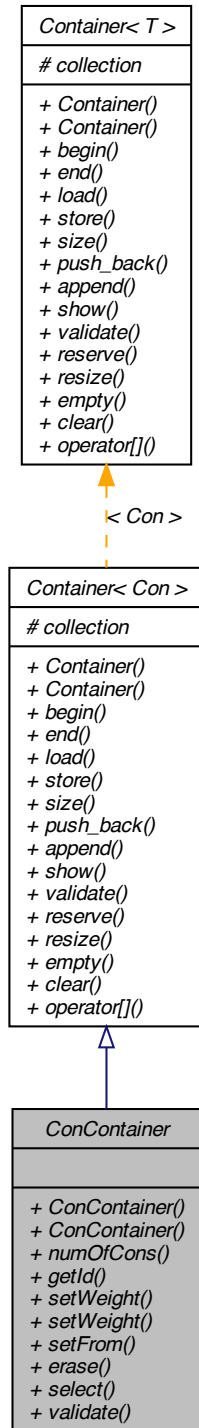
- `pkg/AMORE/src/Con.h`
- `pkg/AMORE/src/Con.cpp`

5.3 ConContainer Class Reference

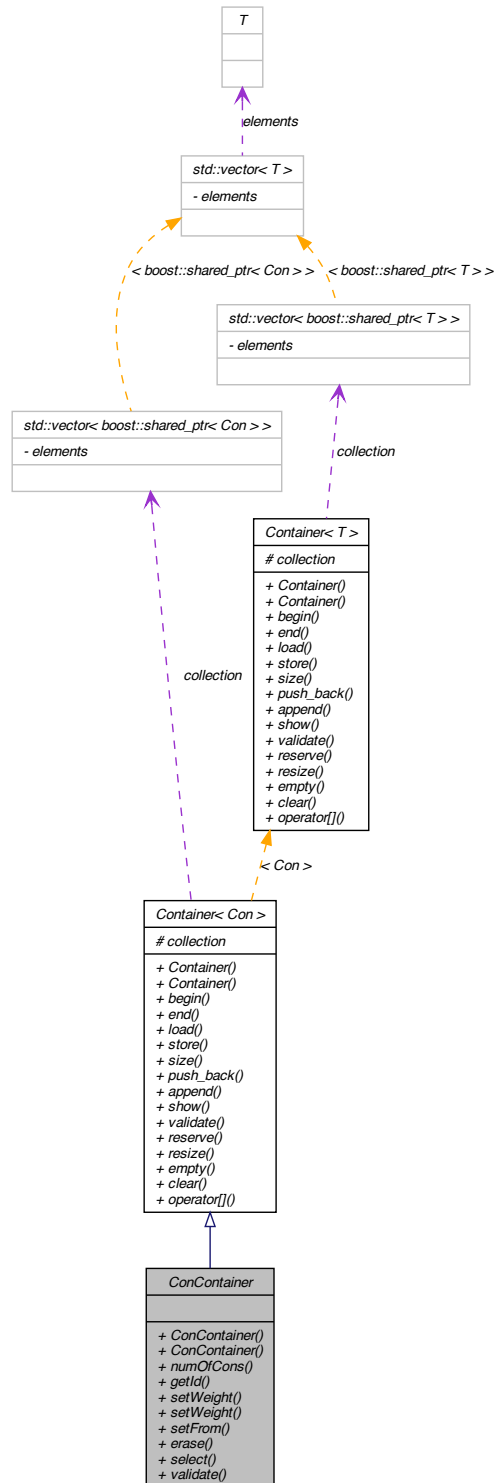
A vector of connections.

```
#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 > [nlds](#))
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 > [nlds](#))
Erase the specified elements from the [vecCom](#) object.
- [ConContainerPtr](#) [select](#) (std::vector< int > [nlds](#))
Selects the specified elements from the [vecCom](#) object.
- bool [validate](#) ()
Object validator.

5.3.1 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](#).

5.3.2 Member Typedef Documentation

5.3.2.1 typedef std::vector<boost::shared_ptr<Con> >::const_iterator [ConContainer::const_iterator](#)

Reimplemented from [Container< Con >](#).

Definition at line 23 of file [ConContainer.h](#).

5.3.2.2 typedef value_type const& ConContainer::const_reference

Reimplemented from [Container< Con >](#).

Definition at line 27 of file ConContainer.h.

5.3.2.3 typedef std::vector<boost::shared_ptr<Con> >::iterator ConContainer::iterator

Reimplemented from [Container< Con >](#).

Definition at line 21 of file ConContainer.h.

5.3.2.4 typedef boost::shared_ptr<Con> ConContainer::value_type

Reimplemented from [Container< Con >](#).

Definition at line 25 of file ConContainer.h.

5.3.3 Constructor & Destructor Documentation

5.3.3.1 ConContainer::ConContainer ()

Definition at line 8 of file ConContainer.cpp.

```
{
}
```

5.3.3.2 ConContainer::ConContainer (std::vector< ConPtr > collection)

Definition at line 12 of file ConContainer.cpp.

```

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

5.3.4 Member Function Documentation

5.3.4.1 void ConContainer::erase (std::vector< int > nlds)

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

<i>vFrom</i>	An std::vector<int> with the lds 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.

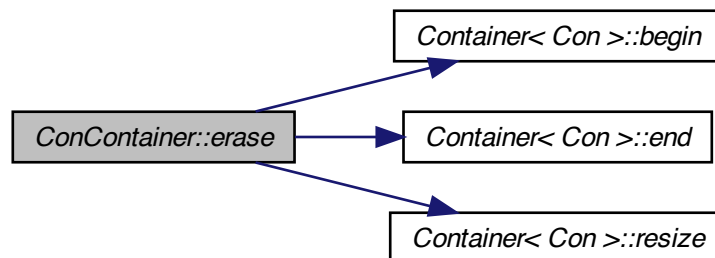
Definition at line 450 of file `ConContainer.cpp`.

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:



5.3.4.2 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) );
    new Con and assign to ptCon
    conContainer.push_back(ptCon);
/ push_back
    ptCon.reset( new Con(&N3, 3.33) );
    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().

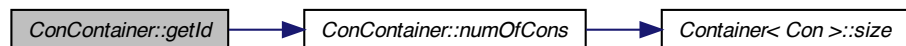
Referenced by Neuron::getConId(), and validate().

```

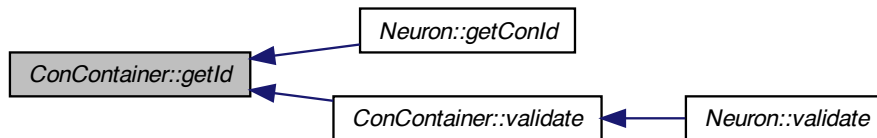
{
    std::vector<int> result;
    result.reserve(numOfCons());
    foreach (ConPtr itr, *this)
    {
        result.push_back(itr->getId());
    }
    return result;
}

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.4.3 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

Container<Neuron>( ) );
ConContainer( ) );

std::vector<int> result;
std::vector<ConPtr> vcA, vcB;
ContainerNeuronPtr neuronContainerPtr( new
ConContainerPtr conContainerPtr( new
ConPtr ptC;
NeuronPtr ptN;
int ids[] = {10, 20, 30};
double weights[] = {1.13, 2.22, 3.33 };
for (int i=0; i<=2 ; i++) {
    ptN.reset( new Neuron( ids[i] ) );
    neuronContainerPtr->push_back(ptN);
}

// Test
for (int i=0; i<=2 ; i++) {
    result.push_back(conContainerPtr->numOfCons());
    // Append numOfCons to result, create new Con and push_back into
    conContainer
    ptC.reset( new Con( neuronContainerPtr->load().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()`.

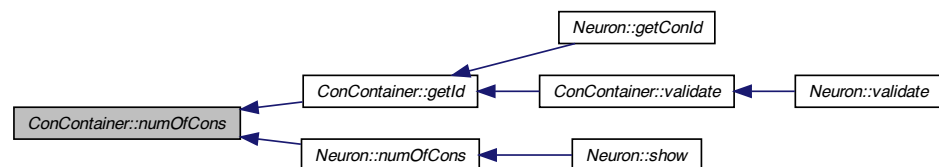
Referenced by `getId()`, and `Neuron::numOfCons()`.

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

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.4.4 ConContainerPtr ConContainer::select (std::vector< int > nlds)

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

<i>vFrom</i>	An <code>std::vector<int></code> 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()`.

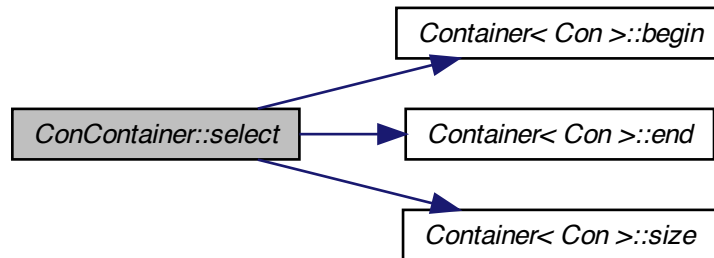
```

{
    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:



5.3.4.5 `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

<i>vFrom</i>	An <code>std::vector<NeuronPtr></code> 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()`.

Referenced by `Neuron::setFrom()`.

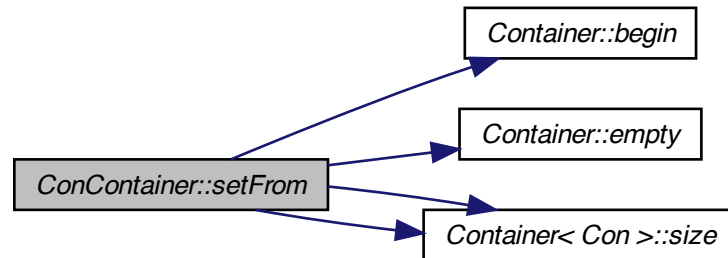
```

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

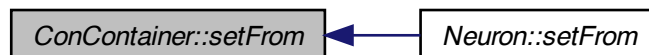
    if (neuronContainer.size() != size())
    {
        throw std::range_error(
            "[C++ ConContainer::setFrom]: Error, neuronContainer.size() != collecti
on.size()");
    }
    std::vector<NeuronPtr>::iterator itrNeuron = neuronContainer.begin();
    foreach(ConPtr itr , *this)
    {
        itr->setFrom(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

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.4.6 `bool ConContainer::setWeight (std::vector< double > nWeights, std::vector< int > nlds)`

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

<i>nWeights</i>	A numeric (double) vector with the weights to be set in the Con objects contained in the ConContainer object.
<i>vFrom</i>	An <code>std::vector<int></code> with the lds 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.

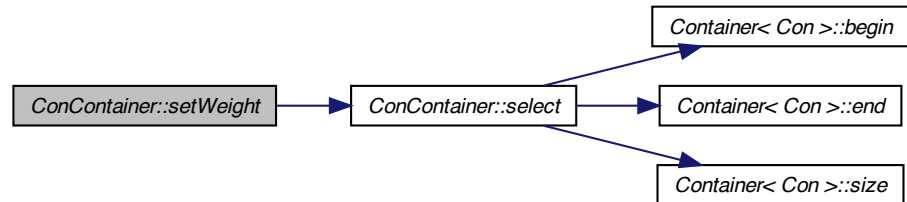
References select().

```

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

```

Here is the call graph for this function:



5.3.4.7 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

<i>nWeights</i>	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[i]) );
    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()`.

Referenced by `Neuron::setWeight()`.

```

{
    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.size()");
    }
    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:



Here is the caller graph for this function:



5.3.4.8 bool ConContainer::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 [ConContainer](#) object are identified as corrupted.

Returns

true in case the checks are Ok.

Exceptions

<i>An</i>	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()`.

Referenced by `Neuron::validate()`.

```

{
    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:



Here is the caller graph for this function:



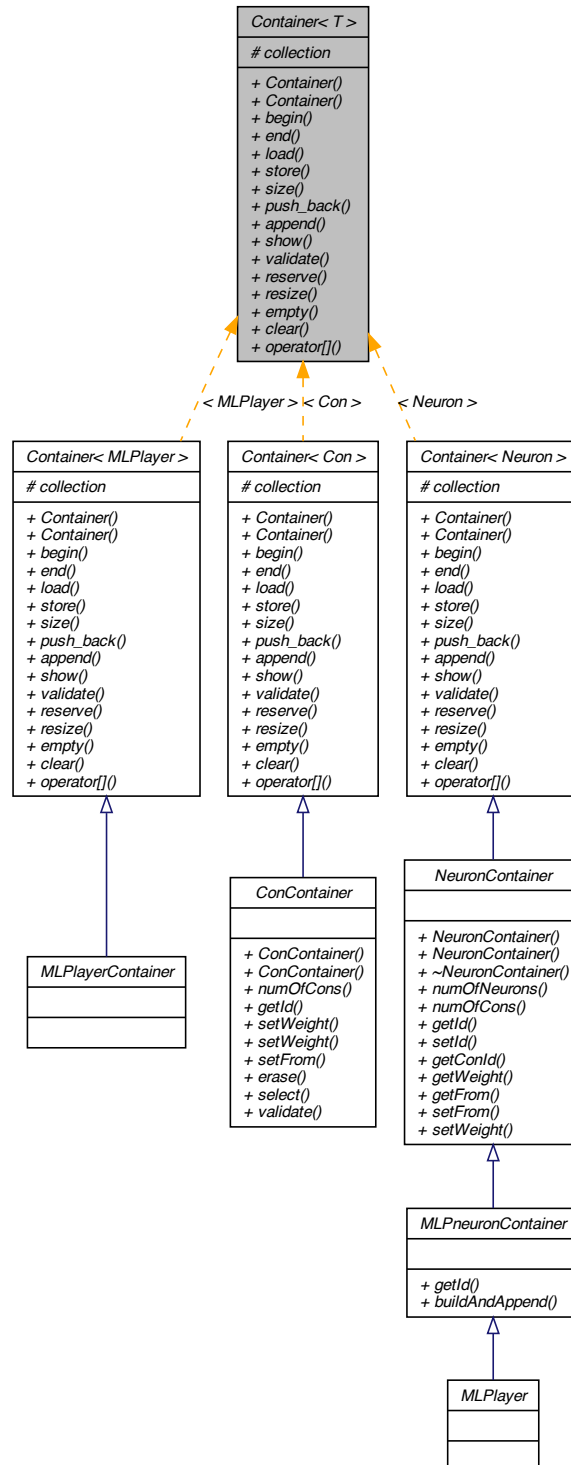
The documentation for this class was generated from the following files:

- [pkg/AMORE/src/ConContainer.h](#)
- [pkg/AMORE/src/ConContainer.cpp](#)

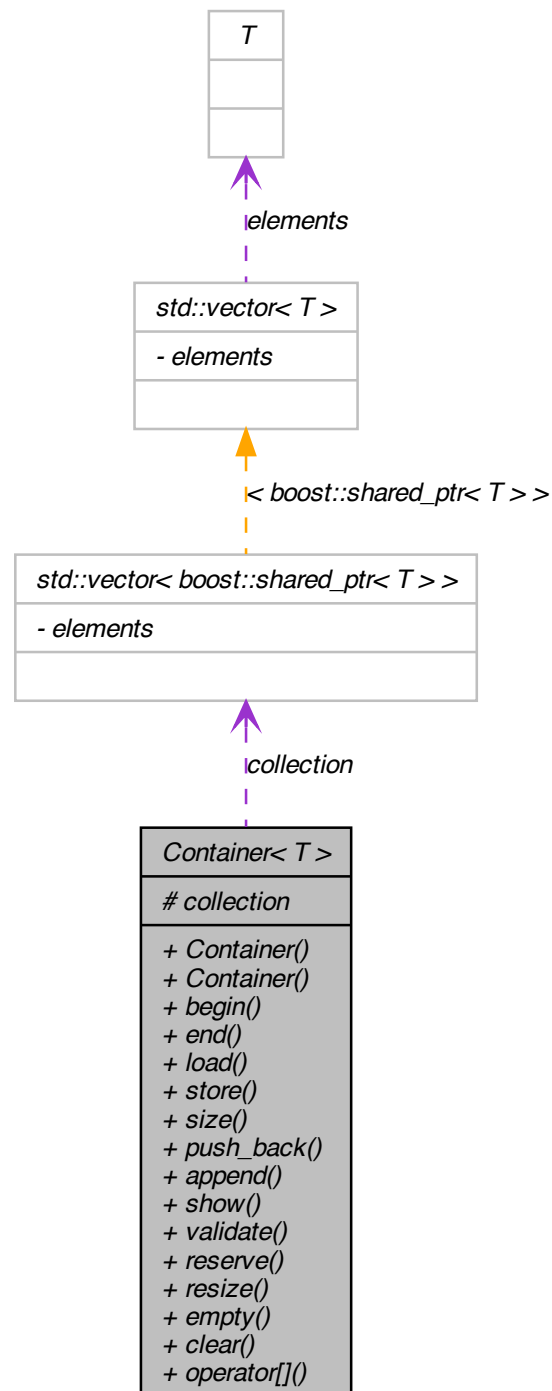
5.4 Container< T > Class Template Reference

```
#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 (typename std::vector< boost::shared_ptr< T > > collection)
- 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 ()
Returns the size or length of the vector.
- 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 ()
Pretty print of the Container<T>
- bool validate ()
Object validator.
- void reserve (int n)
- void resize (int n)
- bool empty ()
- void clear ()
- boost::shared_ptr< T > & operator[] (size_type offset)

Protected Attributes

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

5.4.1 Detailed Description

template<typename T>class Container< T >

Definition at line 12 of file Container.h.

5.4.2 Member Typedef Documentation

5.4.2.1 `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.

5.4.2.2 `template<typename T> typedef value_type const& Container< T
>::const_reference`

Reimplemented in [ConContainer](#), and [NeuronContainer](#).

Definition at line 26 of file Container.h.

5.4.2.3 `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.

5.4.2.4 `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.

5.4.3 Constructor & Destructor Documentation

5.4.3.1 `template<typename T> Container< T >::Container ()`

Definition at line 9 of file Container.cpp.

```
{
}
```

5.4.3.2 `template<typename T> Container< T >::Container (typename std::vector<
boost::shared_ptr< T > > collection)`

Definition at line 14 of file Container.cpp.

```

                                     :
    collection(collection)
    {
    }
```

5.4.4 Member Function Documentation

5.4.4.1 `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 <code>Container<T></code> 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

Container<Neuron>() );
Container<Con>() );
Container<Con>() );

std::vector<int> result;
std::vector<ConPtr> vcA, vcB;
ContainerNeuronPtr neuronContainerPtr( new
ContainerConPtr conContainerPtrA( new
ContainerConPtr conContainerPtrB( new
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->load().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->load().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

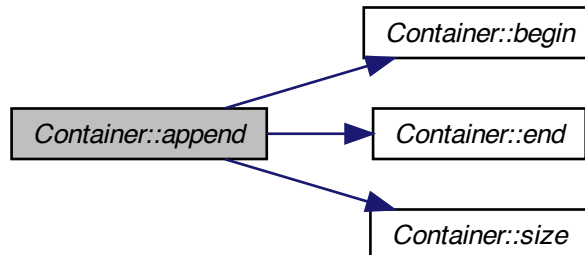
[Container::store](#), [Container::push_back](#) and the unit test files, e.g., `runit.Cpp.Container.R`, for usage examples.

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:



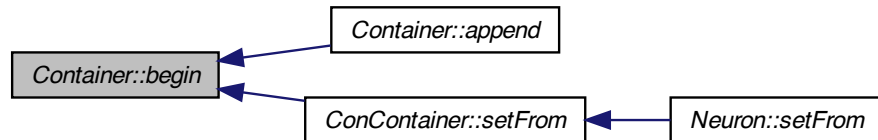
5.4.4.2 `template<typename T> std::vector< boost::shared_ptr< T >>::iterator Container< T >::begin ()`

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:



5.4.4.3 `template<typename T> void Container< T>::clear ()`

Definition at line 310 of file Container.cpp.

```

{
    collection.clear();
}

```

5.4.4.4 `template<typename T> bool Container< T>::empty ()`

Definition at line 303 of file Container.cpp.

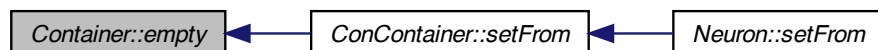
Referenced by `ConContainer::setFrom()`.

```

{
    return (collection.empty());
}

```

Here is the caller graph for this function:



5.4.4.5 `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:



5.4.4.6 `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++) {
    ptN.reset( new Neuron( ids[i] ) );
    neuronContainerPtr->push_back(ptN);
}
/ Let's create a vector with three neurons
```

```

                                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;
}

```

5.4.4.7 `template<typename T> boost::shared_ptr< T> & Container< T>::operator[] (size_type offset)`

Definition at line 317 of file `Container.cpp`.

```

{
    return collection[offset];
}

```

5.4.4.8 `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

<i>TsharedPtr</i>	A <code>shared_ptr</code> 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);
}

```

5.4.4.9 template<typename T> void Container< T >::reserve (int n)

Definition at line 296 of file `Container.cpp`.

```

{
    collection.reserve(n);
}

```

5.4.4.10 template<typename T> void Container< T >::resize (int n)

Definition at line 289 of file `Container.cpp`.

```

{
    collection.resize(n);
}

```

5.4.4.11 `template<typename T> bool Container<T>::show ()`

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.

Definition at line 122 of file `Container.cpp`.

Referenced by `Neuron::show()`.

```
{
// This is equivalent to:
// for( auto x : collection)      { x.show(); }
// Waiting for C++0x
```

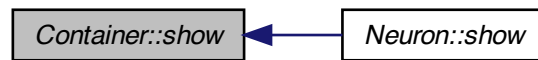


```

foreach (typename boost::shared_ptr<T> itr, *this)
{
    itr->show();
}
return true;
}

```

Here is the caller graph for this function:



5.4.4.12 `template<typename T> size_type Container< T >::size ()`

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.

Definition at line 281 of file `Container.cpp`.

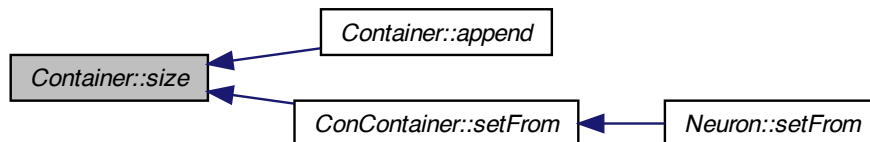
Referenced by `Container< T >::append()`, and `ConContainer::setFrom()`.

```

{
    return collection.size();
}

```

Here is the caller graph for this function:



5.4.4.13 `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;
}
```

5.4.4.14 `template<typename T> bool Container< T >::validate ()`

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.

Reimplemented in [ConContainer](#).

Definition at line 144 of file `Container.cpp`.

```
{
    foreach (typename boost::shared_ptr<T> itr, *this)
    {
        itr->validate();
    }
    return true;
}
```

5.4.5 Member Data Documentation

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

Definition at line 15 of file `Container.h`.

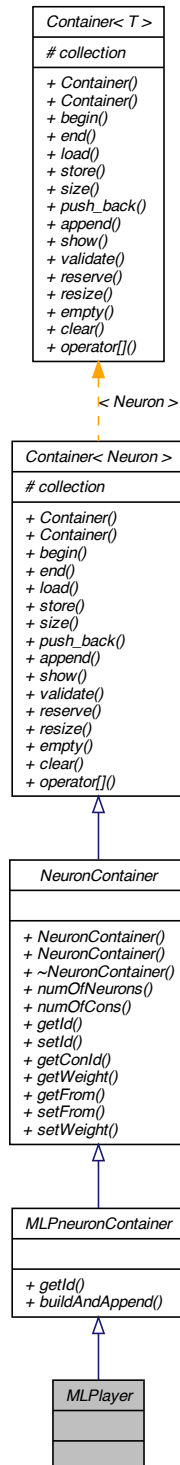
The documentation for this class was generated from the following files:

- `pkg/AMORE/src/Container.h`
- `pkg/AMORE/src/Container.cpp`

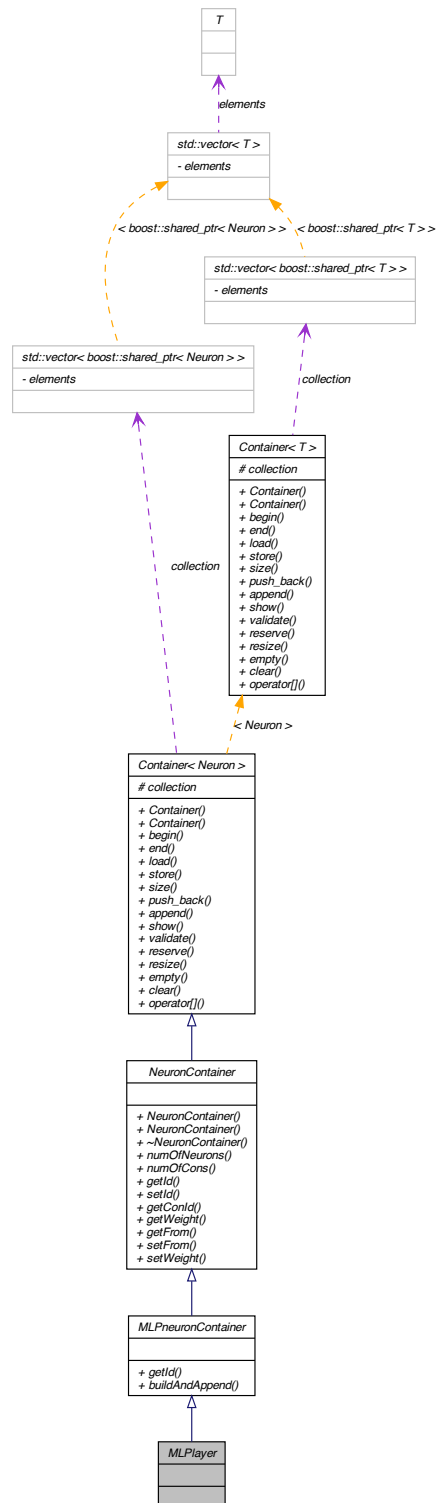
5.5 MLPlayer Class Reference

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

Inheritance diagram for MLPlayer:



Collaboration diagram for MLPlayer:



5.5.1 Detailed Description

Definition at line 1 of file MLPlayer.h.

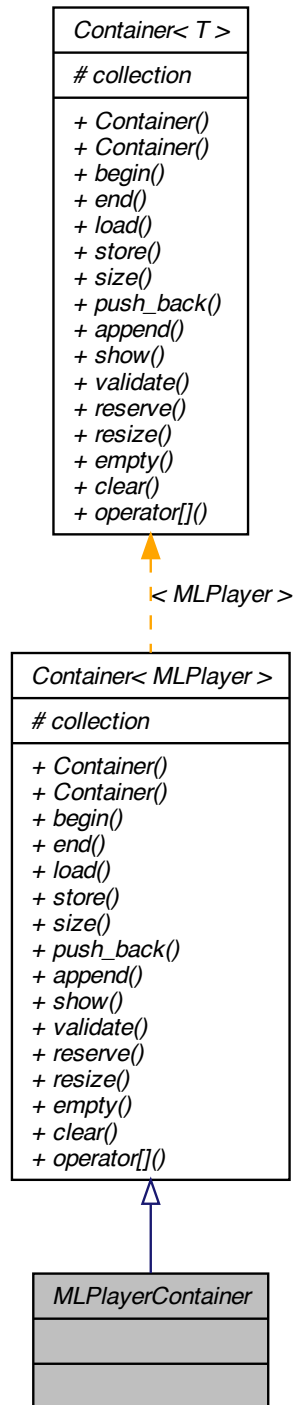
The documentation for this class was generated from the following file:

- pkg/AMORE/src/[MLPlayer.h](#)

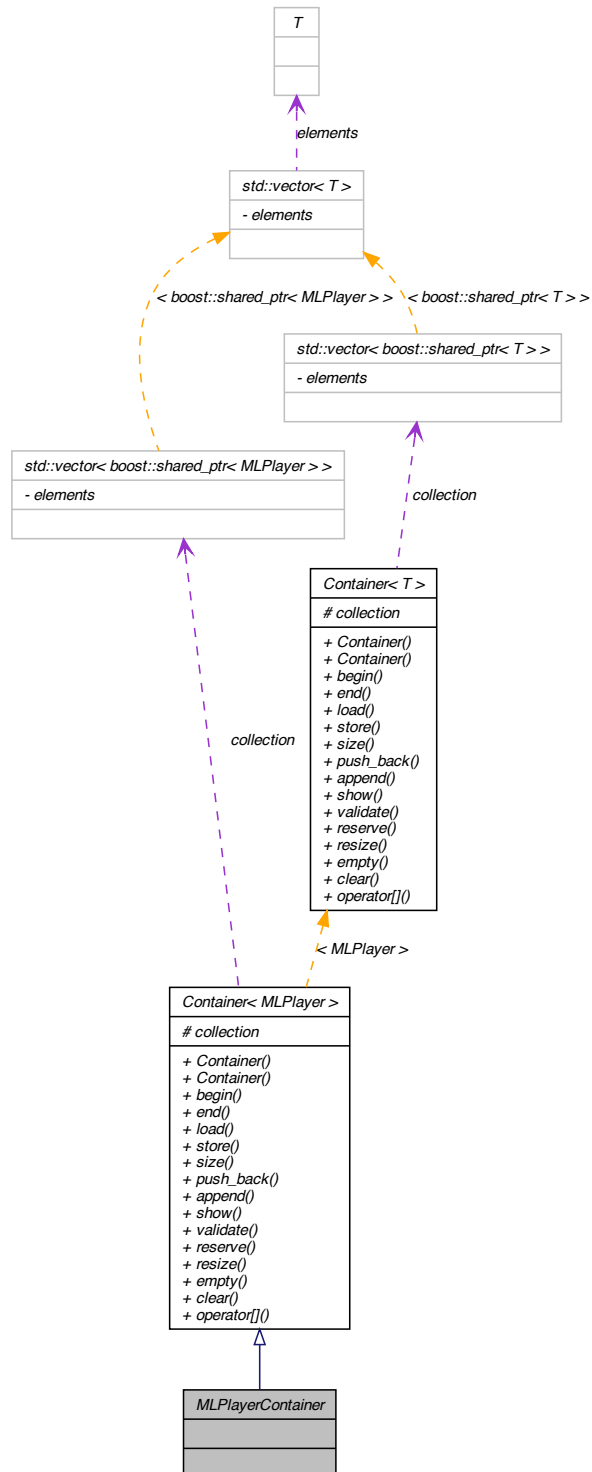
5.6 MLPlayerContainer Class Reference

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

Inheritance diagram for MLPlayerContainer:



Collaboration diagram for MLPlayerContainer:



5.6.1 Detailed Description

Definition at line 1 of file MLPPlayerContainer.h.

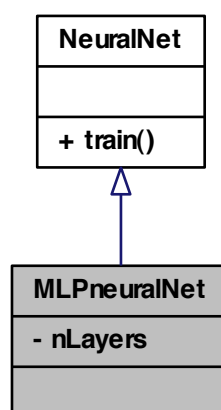
The documentation for this class was generated from the following file:

- [pkg/AMORE/src/MLPlayerContainer.h](#)

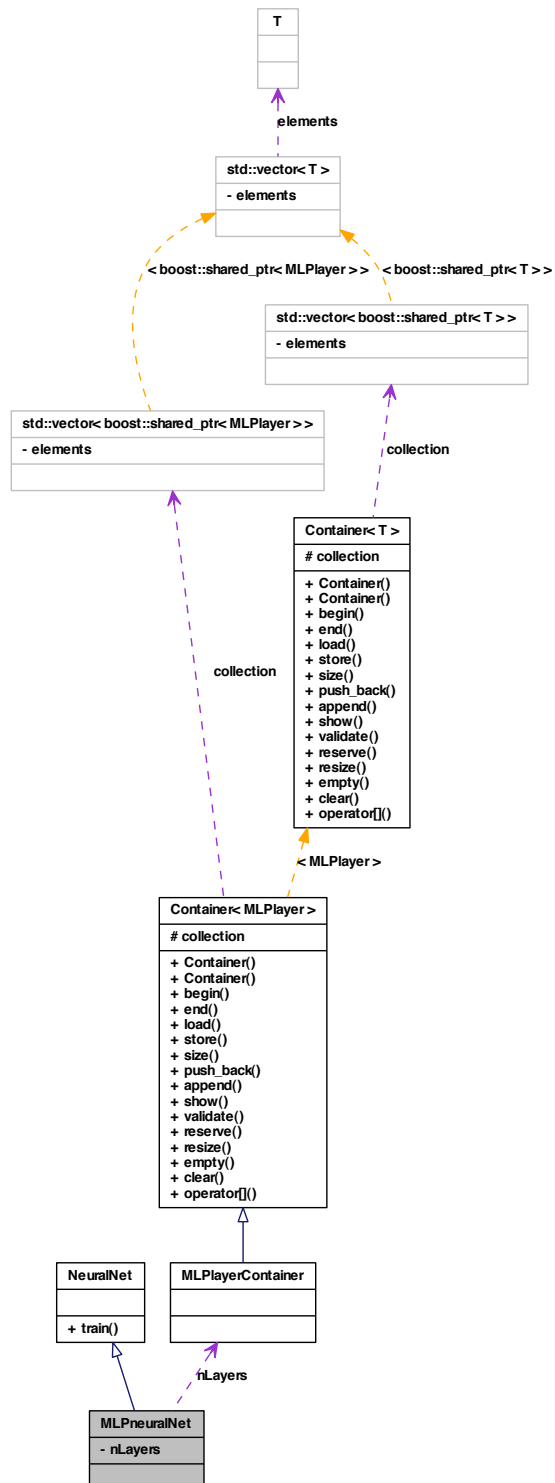
5.7 MLPneuralNet Class Reference

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

Inheritance diagram for MLPneuralNet:



Collaboration diagram for MLPneuralNet:



Private Attributes

- [MLPlayerContainer nLayers](#)

5.7.1 Detailed Description

Definition at line 1 of file MLPneuralNet.h.

5.7.2 Member Data Documentation

5.7.2.1 MLPlayerContainer MLPneuralNet::nLayers [private]

Definition at line 2 of file MLPneuralNet.h.

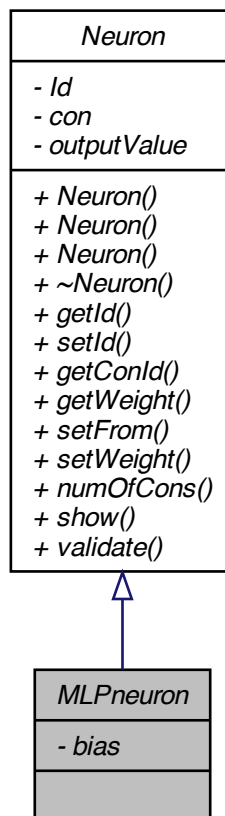
The documentation for this class was generated from the following file:

- [pkg/AMORE/src/MLPneuralNet.h](#)

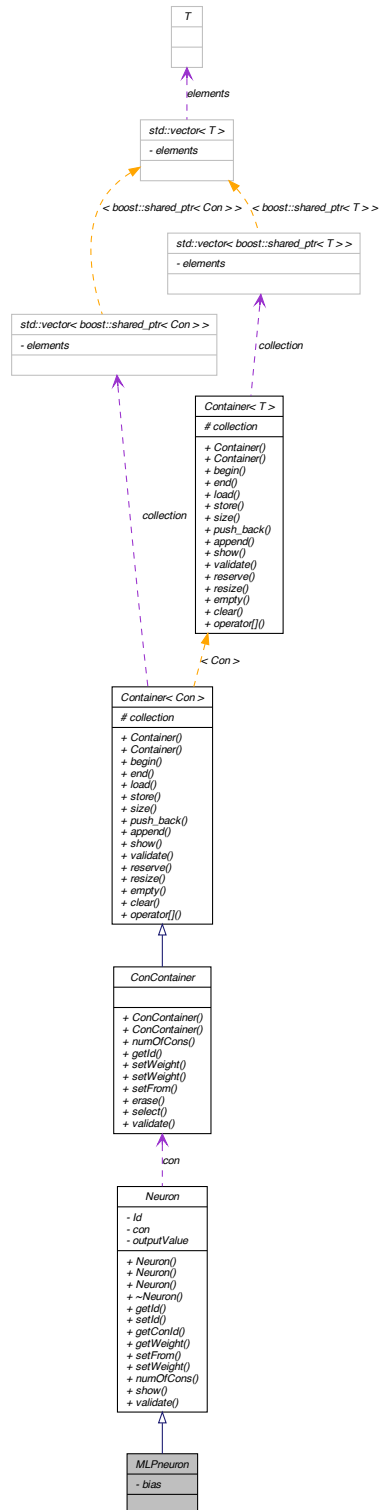
5.8 MLPneuron Class Reference

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

Inheritance diagram for MLPneuron:



Collaboration diagram for MLPneuron:



Private Attributes

- int [bias](#)

5.8.1 Detailed Description

Definition at line 1 of file MLPneuron.h.

5.8.2 Member Data Documentation

5.8.2.1 int MLPneuron::bias [private]

Definition at line 2 of file MLPneuron.h.

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

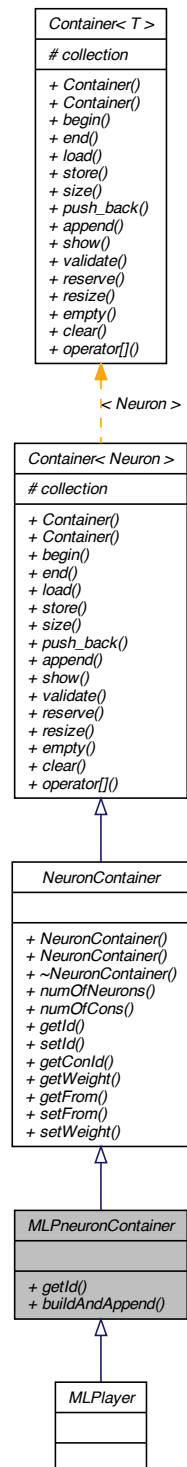
- pkg/AMORE/src/[MLPneuron.h](#)

5.9 MLPneuronContainer Class Reference

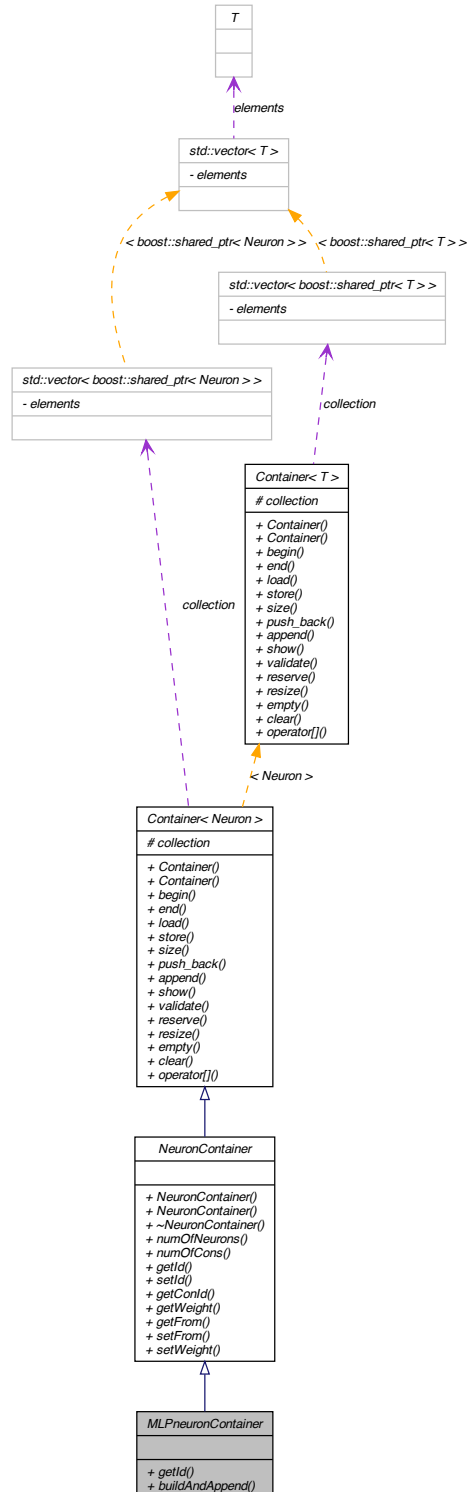
A vector of connections.

```
#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)`

5.9.1 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`.

5.9.2 Member Function Documentation

5.9.2.1 `bool MLPneuronContainer::buildAndAppend (std::vector< int > IDS, std::vector< int > BIAS, ConContainer VC)`

5.9.2.2 `std::vector<int> MLPneuronContainer::getId ()`

Reimplemented from [NeuronContainer](#).

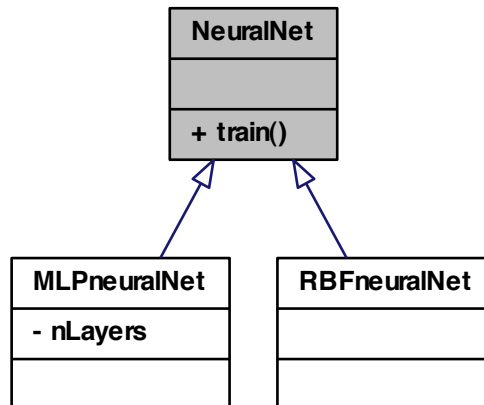
The documentation for this class was generated from the following file:

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

5.10 NeuralNet Class Reference

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

Inheritance diagram for NeuralNet:



Public Member Functions

- virtual void [train](#) ()=0

5.10.1 Detailed Description

Definition at line 1 of file NeuralNet.h.

5.10.2 Member Function Documentation

5.10.2.1 virtual void NeuralNet::train () [pure virtual]

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

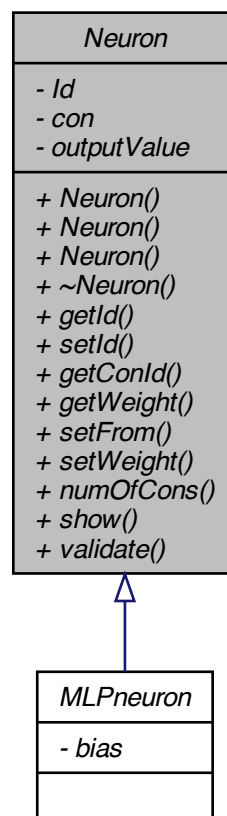
- pkg/AMORE/src/[NeuralNet.h](#)

5.11 Neuron Class Reference

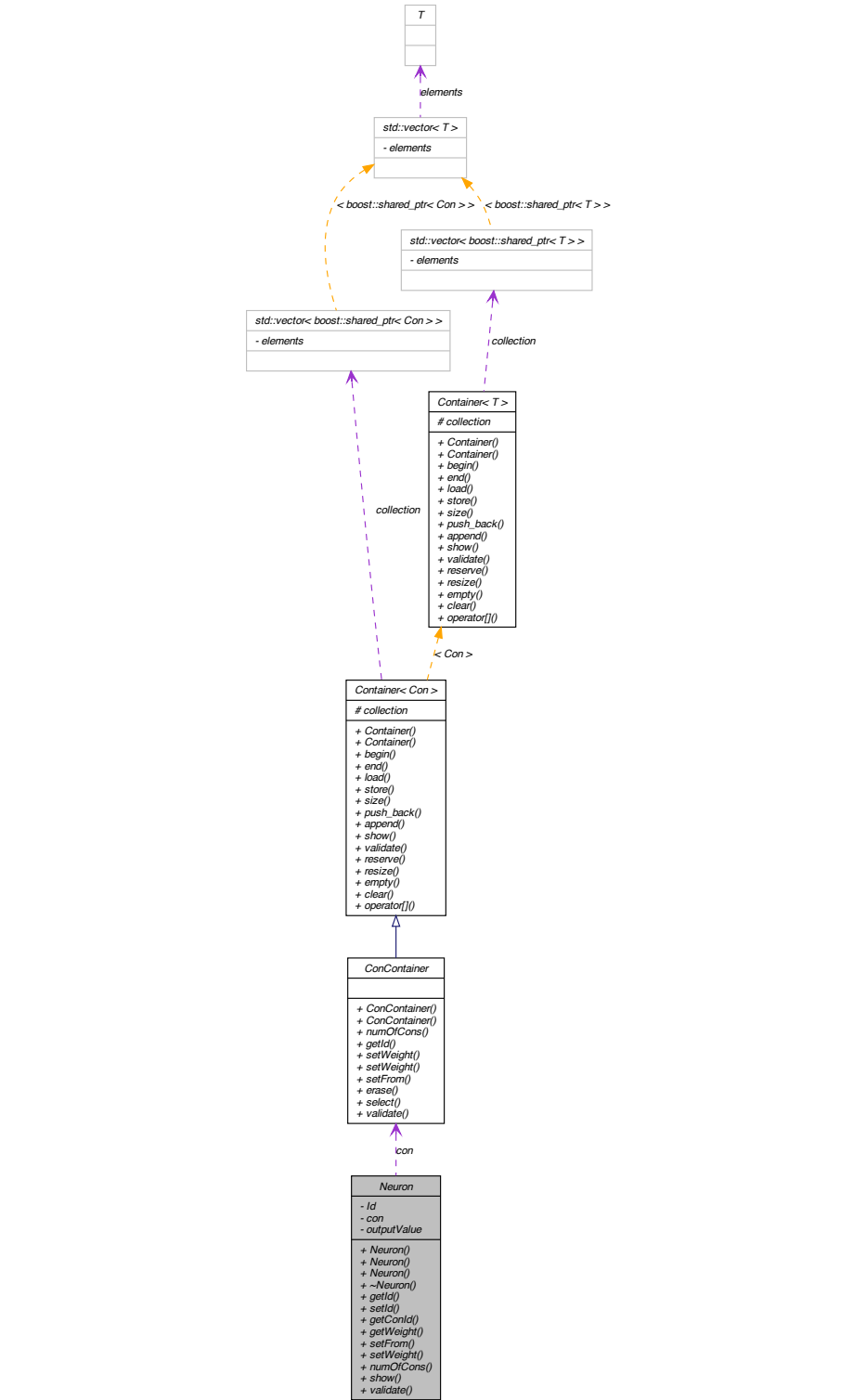
A class to handle the information contained in a general [Neuron](#).

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

Inheritance diagram for Neuron:



Collaboration diagram for Neuron:



Public Member Functions

- [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](#) ()

Private Attributes

- int [Id](#)
An integer variable with the [Neuron](#) Id.
- [ConContainer](#) [con](#)
A vector of input connections.
- double [outputValue](#)

5.11.1 Detailed Description

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 16 of file [Neuron.h](#).

5.11.2 Constructor & Destructor Documentation

5.11.2.1 [Neuron::Neuron](#) ()

Definition at line 10 of file [Neuron.cpp](#).

```

:
  Id (NA_INTEGER), con ()
{
}

```

5.11.2.2 Neuron::Neuron (int *Id*)

Definition at line 15 of file Neuron.cpp.

```

        :
        Id(Id), outputValue(0.0)
    {
    }

```

5.11.2.3 Neuron::Neuron (int *Id*, ConContainer *con*)

Definition at line 20 of file Neuron.cpp.

```

        :
        Id(Id), con(con), outputValue(0.0)
    {
    }

```

5.11.2.4 Neuron::~~Neuron ()

Definition at line 25 of file Neuron.cpp.

```

{
}

```

5.11.3 Member Function Documentation

5.11.3.1 std::vector< int > Neuron::getConId ()

Definition at line 43 of file Neuron.cpp.

References `con`, and `ConContainer::getId()`.

```

{
    return con.getId();
}

```

Here is the call graph for this function:



5.11.3.2 int Neuron::getId ()

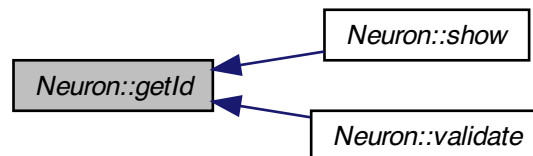
Definition at line 30 of file Neuron.cpp.

References Id.

Referenced by show(), and validate().

```
{  
    return Id;  
}
```

Here is the caller graph for this function:



5.11.3.3 std::vector< double > Neuron::getWeight ()

Definition at line 49 of file Neuron.cpp.

References con.

```
{  
    return con.getWeight();  
}
```

5.11.3.4 int Neuron::numOfCons ()

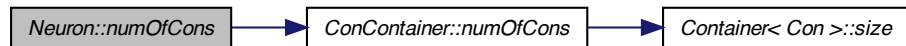
Definition at line 67 of file Neuron.cpp.

References con, and ConContainer::numOfCons().

Referenced by show().

```
{  
    return con.numOfCons();  
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.3.5 bool Neuron::setFrom (NeuronContainer neuronContainer)

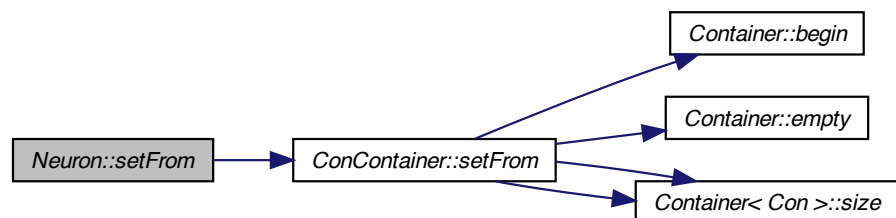
Definition at line 55 of file Neuron.cpp.

References `con`, and `ConContainer::setFrom()`.

```

{
    con.setFrom(neuronContainer);
}
  
```

Here is the call graph for this function:



5.11.3.6 void Neuron::setId (int *value*)

Definition at line 36 of file Neuron.cpp.

References Id.

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

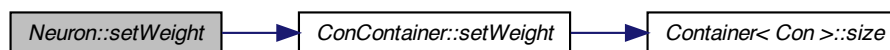
5.11.3.7 bool Neuron::setWeight (std::vector< double > *nWeights*)

Definition at line 61 of file Neuron.cpp.

References con, and ConContainer::setWeight().

```
{
    con.setWeight(nWeights);
}
```

Here is the call graph for this function:

**5.11.3.8 bool Neuron::show ()**

Definition at line 73 of file Neuron.cpp.

References con, getId(), numOfCons(), and Container< T >::show().

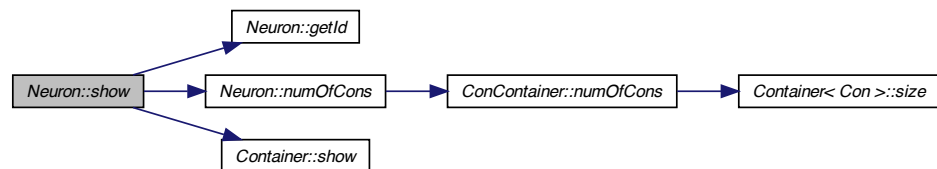
```
{
    int id = getId();
    Rprintf("\n-----\n");
    if (id == NA_INTEGER)
    {
        Rprintf("\n Id: NA, Invalid neuron Id");
    }
    else
    {
        Rprintf("\n Id: %d", id);
    }
    Rprintf("\n-----\n");
    if (numOfCons() == 0)
    {
```

```

        Rprintf("\n No connections defined");
    }
    else
    {
        con.show();
    }
    Rprintf("\n-----\n");
    return true;
}

```

Here is the call graph for this function:



5.11.3.9 bool Neuron::validate ()

Definition at line 100 of file Neuron.cpp.

References `con`, `getId()`, and `ConContainer::validate()`.

```

{
    BEGIN_RCPP
    if (getId() == NA_INTEGER ) throw std::range_error("[C++ Neuron::validate]: Err
        or, Id is NA.");
    con.validate();
    return (TRUE);
    END_RCPP
}

```

Here is the call graph for this function:



5.11.4 Member Data Documentation

5.11.4.1 ConContainer Neuron::con [private]

A vector of input connections.

Definition at line 29 of file Neuron.h.

Referenced by getConId(), getWeight(), numOfCons(), setFrom(), setWeight(), show(), and validate().

5.11.4.2 int Neuron::ld [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.

Referenced by getId(), and setId().

5.11.4.3 double Neuron::outputValue [private]

Definition at line 30 of file Neuron.h.

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

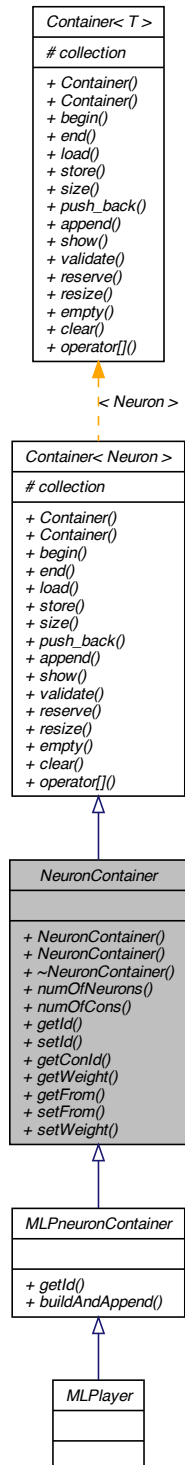
- pkg/AMORE/src/[Neuron.h](#)
- pkg/AMORE/src/[Neuron.cpp](#)

5.12 NeuronContainer Class Reference

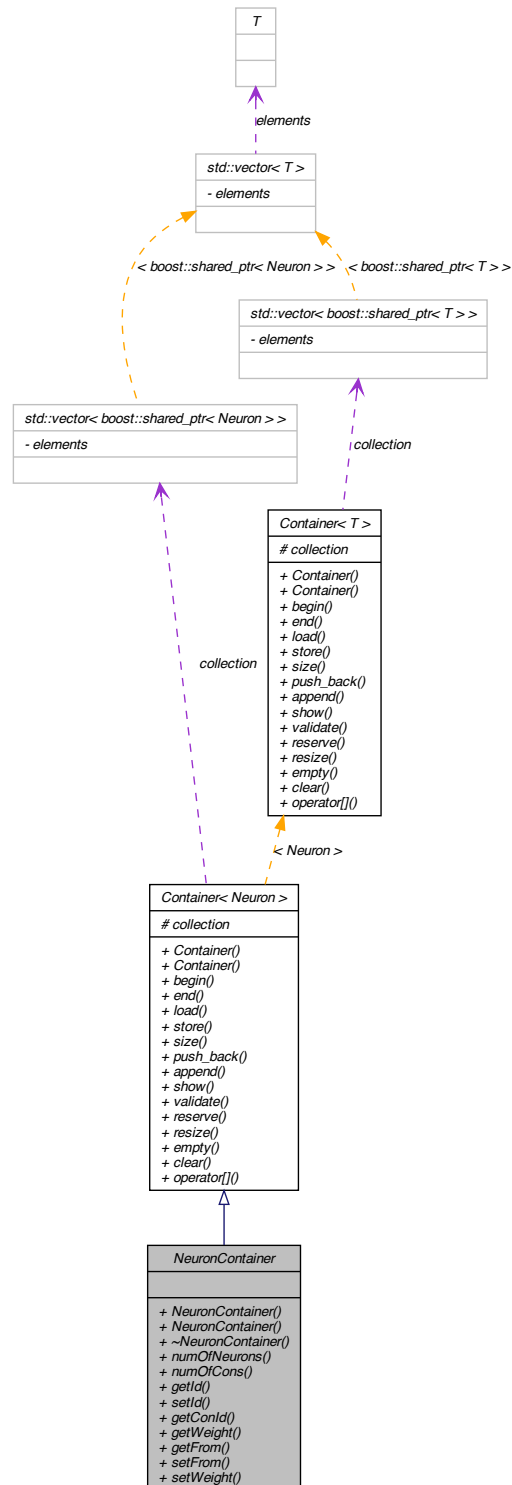
A vector of neurons.

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

Inheritance diagram for NeuronContainer:



Collaboration diagram for 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 > nlds)
- 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)

5.12.1 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.

5.12.2 Member Typedef Documentation

5.12.2.1 typedef [NeuronContainer_const_iterator](#) [NeuronContainer::const_iterator](#)

Reimplemented from [Container< Neuron >](#).

Definition at line 23 of file NeuronContainer.h.

5.12.2.2 typedef [value_type](#) const& [NeuronContainer::const_reference](#)

Reimplemented from [Container< Neuron >](#).

Definition at line 27 of file NeuronContainer.h.

5.12.2.3 `typedef NeuronContainer_iterator NeuronContainer::iterator`

Reimplemented from [Container< Neuron >](#).

Definition at line 21 of file NeuronContainer.h.

5.12.2.4 `typedef boost::shared_ptr<Neuron> NeuronContainer::value_type`

Reimplemented from [Container< Neuron >](#).

Definition at line 25 of file NeuronContainer.h.

5.12.3 Constructor & Destructor Documentation

5.12.3.1 `NeuronContainer::NeuronContainer ()`

Definition at line 8 of file NeuronContainer.cpp.

```
{
}
```

5.12.3.2 `NeuronContainer::NeuronContainer (std::vector< NeuronPtr > neuronContainer)`

Definition at line 12 of file NeuronContainer.cpp.

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

5.12.3.3 `NeuronContainer::~~NeuronContainer ()`

Definition at line 17 of file NeuronContainer.cpp.

```
{
}
```

5.12.4 Member Function Documentation

5.12.4.1 `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;
}
```

5.12.4.2 `std::vector<NeuronContainer> NeuronContainer::getFrom ()`

5.12.4.3 `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;
}
```

5.12.4.4 `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;
}
```

5.12.4.5 `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;
}
```


5.12.4.6 `int NeuronContainer::numOfNeurons ()`

Definition at line 22 of file NeuronContainer.cpp.

References `Container< Neuron >::size()`.

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

Here is the call graph for this function:

5.12.4.7 `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++;
    }
}
```

5.12.4.8 `void NeuronContainer::setId (std::vector< int > nlds)`

Definition at line 50 of file NeuronContainer.cpp.

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

5.12.4.9 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++;
    }
}
```

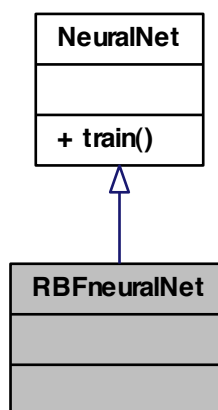
The documentation for this class was generated from the following files:

- pkg/AMORE/src/[NeuronContainer.h](#)
- pkg/AMORE/src/[NeuronContainer.cpp](#)

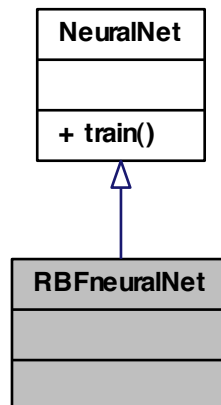
5.13 RBFneuralNet Class Reference

```
#include <RBFneuralNet.h>
```

Inheritance diagram for RBFneuralNet:



Collaboration diagram for RBFneuralNet:



5.13.1 Detailed Description

Definition at line 1 of file RBFneuralNet.h.

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

- pkg/AMORE/src/[RBFneuralNet.h](#)

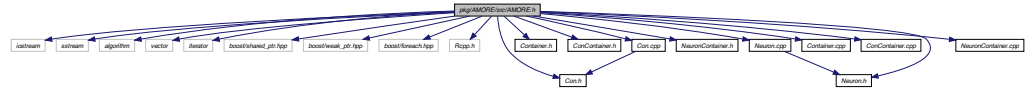
Chapter 6

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 <Rcpp.h>
#include "Con.h"
#include "Container.h"
#include "ConContainer.h"
#include "Neuron.h"
#include "NeuronContainer.h"
#include "Con.cpp"
#include "Container.cpp"
#include "ConContainer.cpp"
#include "Neuron.cpp"
#include "NeuronContainer.cpp"
```

Include dependency graph for AMORE.h:



Defines

- `#define` [foreach](#) BOOST_FOREACH
- `#define` [size_type](#) unsigned int

Typedefs

- `typedef boost::shared_ptr< Con > ConPtr`
- `typedef boost::shared_ptr< Neuron > NeuronPtr`
- `typedef boost::weak_ptr< Neuron > NeuronWeakPtr`
- `typedef boost::shared_ptr< Container< Con > > ContainerConPtr`
- `typedef boost::shared_ptr< Container< Neuron > > ContainerNeuronPtr`
- `typedef boost::shared_ptr< ConContainer > ConContainerPtr`
- `typedef boost::shared_ptr< NeuronContainer > NeuronContainerPtr`
- `typedef std::vector< NeuronPtr >::iterator NeuronContainer_iterator`
- `typedef std::vector< NeuronPtr >::const_iterator NeuronContainer_const_iterator`

6.1.1 Define Documentation

6.1.1.1 `#define` `foreach` BOOST_FOREACH

Definition at line 37 of file AMORE.h.

6.1.1.2 `#define` `size_type` unsigned int

Definition at line 40 of file AMORE.h.

6.1.2 Typedef Documentation

6.1.2.1 `typedef boost::shared_ptr<ConContainer> ConContainerPtr`

Definition at line 48 of file AMORE.h.

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

Definition at line 43 of file AMORE.h.

6.1.2.3 `typedef boost::shared_ptr<Container<Con> > ContainerConPtr`

Definition at line 46 of file AMORE.h.

6.1.2.4 `typedef boost::shared_ptr<Container<Neuron> > ContainerNeuronPtr`

Definition at line 47 of file AMORE.h.

6.1.2.5 `typedef std::vector<NeuronPtr>::const_iterator NeuronContainer_const_iterator`

Definition at line 52 of file AMORE.h.

6.1.2.6 `typedef std::vector<NeuronPtr>::iterator NeuronContainer_iterator`

Definition at line 51 of file AMORE.h.

6.1.2.7 `typedef boost::shared_ptr<NeuronContainer> NeuronContainerPtr`

Definition at line 49 of file AMORE.h.

6.1.2.8 `typedef boost::shared_ptr<Neuron> NeuronPtr`

Definition at line 44 of file AMORE.h.

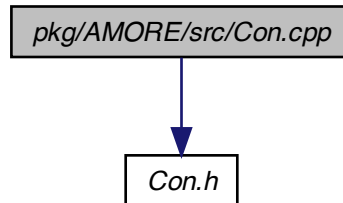
6.1.2.9 `typedef boost::weak_ptr<Neuron> NeuronWeakPtr`

Definition at line 45 of file AMORE.h.

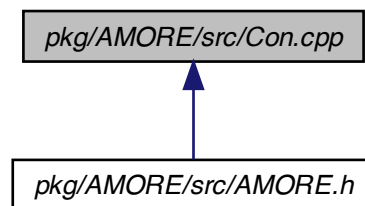
6.2 pkg/AMORE/src/Con.cpp File Reference

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

Include dependency graph for Con.cpp:

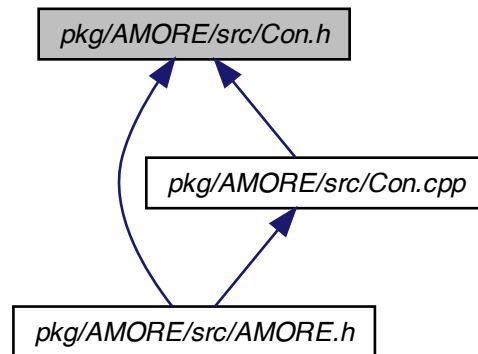


This graph shows which files directly or indirectly include this file:



6.3 pkg/AMORE/src/Con.h File Reference

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



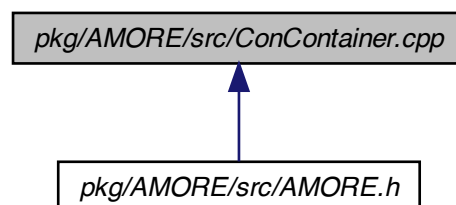
Classes

- class `Con`

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

6.4 pkg/AMORE/src/ConContainer.cpp File Reference

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

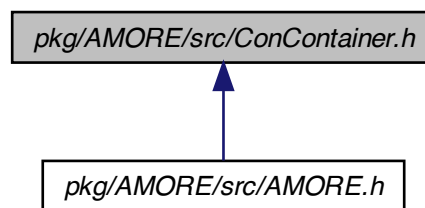


Classes

- struct [CompareId](#)

6.5 pkg/AMORE/src/ConContainer.h File Reference

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

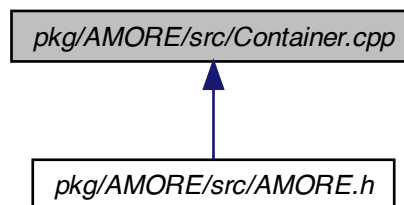


Classes

- class [ConContainer](#)
A vector of connections.

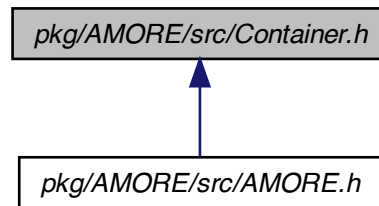
6.6 pkg/AMORE/src/Container.cpp File Reference

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



6.7 pkg/AMORE/src/Container.h File Reference

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



Classes

- class [Container< T >](#)

6.8 pkg/AMORE/src/MLPlayer.h File Reference

Classes

- class [MLPlayer](#)

6.9 pkg/AMORE/src/MLPlayerContainer.h File Reference

Classes

- class [MLPlayerContainer](#)

6.10 pkg/AMORE/src/MLPneuralNet.h File Reference

Classes

- class [MLPneuralNet](#)

6.11 pkg/AMORE/src/MLPneuralNetFactory.cpp File Reference

6.12 pkg/AMORE/src/MLPneuron.h File Reference

Classes

- class [MLPneuron](#)

6.13 pkg/AMORE/src/MLPneuronContainer.h File Reference

Classes

- class [MLPneuronContainer](#)
A vector of connections.

6.14 pkg/AMORE/src/NeuralNet.h File Reference

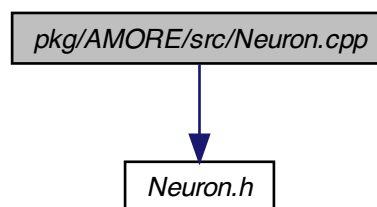
Classes

- class [NeuralNet](#)

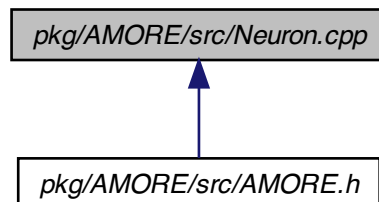
6.15 pkg/AMORE/src/Neuron.cpp File Reference

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

Include dependency graph for Neuron.cpp:

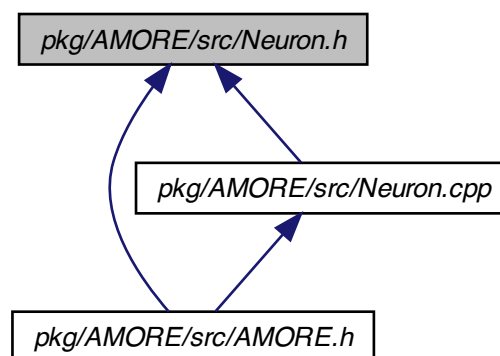


This graph shows which files directly or indirectly include this file:



6.16 pkg/AMORE/src/Neuron.h File Reference

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



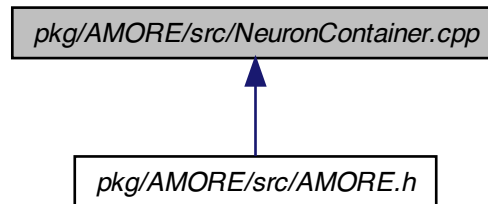
Classes

- class [Neuron](#)

A class to handle the information contained in a general [Neuron](#).

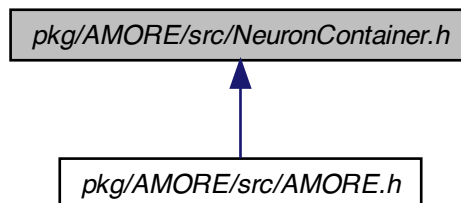
6.17 pkg/AMORE/src/NeuronContainer.cpp File Reference

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



6.18 pkg/AMORE/src/NeuronContainer.h File Reference

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



Classes

- class [NeuronContainer](#)
A vector of neurons.

6.19 pkg/AMORE/src/RBFneuralNet.h File Reference

Classes

- class [RBFneuralNet](#)

Index

- ~Con
 - Con, [12](#)
- ~Neuron
 - Neuron, [70](#)
- ~NeuronContainer
 - NeuronContainer, [79](#)
- AMORE.h
 - ConContainerPtr, [86](#)
 - ConPtr, [86](#)
 - ContainerConPtr, [86](#)
 - ContainerNeuronPtr, [87](#)
 - foreach, [86](#)
 - NeuronContainer_const_iterator, [87](#)
 - NeuronContainer_iterator, [87](#)
 - NeuronContainerPtr, [87](#)
 - NeuronPtr, [87](#)
 - NeuronWeakPtr, [87](#)
 - size_type, [86](#)
- append
 - Container, [42](#)
- begin
 - Container, [43](#)
- bias
 - MLPneuron, [62](#)
- buildAndAppend
 - MLPneuronContainer, [65](#)
- clear
 - Container, [44](#)
- collection
 - Container, [50](#)
- CompareId, [9](#)
 - operator(), [9](#), [10](#)
- Con, [10](#)
 - ~Con, [12](#)
 - Con, [11](#)
 - from, [18](#)
 - getFrom, [12](#)
 - getId, [13](#)
 - getWeight, [14](#)
 - setFrom, [15](#)
 - setWeight, [15](#)
 - show, [16](#)
 - validate, [17](#)
 - weight, [18](#)
- con
 - Neuron, [74](#)
- ConContainer, [19](#)
 - ConContainer, [23](#)
 - const_iterator, [22](#)
 - const_reference, [22](#)
 - erase, [23](#)
 - getId, [25](#)
 - iterator, [23](#)
 - numOfCons, [27](#)
 - select, [28](#)
 - setFrom, [30](#)
 - setWeight, [32](#), [34](#)
 - validate, [36](#)
 - value_type, [23](#)
- ConContainerPtr
 - AMORE.h, [86](#)
- ConPtr
 - AMORE.h, [86](#)
- const_iterator
 - ConContainer, [22](#)
 - Container, [41](#)
 - NeuronContainer, [78](#)
- const_reference
 - ConContainer, [22](#)
 - Container, [41](#)
 - NeuronContainer, [78](#)
- Container, [37](#)
 - append, [42](#)
 - begin, [43](#)
 - clear, [44](#)
 - collection, [50](#)
 - const_iterator, [41](#)
 - const_reference, [41](#)
 - Container, [41](#)

- empty, [44](#)
- end, [44](#)
- iterator, [41](#)
- load, [45](#)
- push_back, [46](#)
- reserve, [47](#)
- resize, [47](#)
- show, [47](#)
- size, [49](#)
- store, [49](#)
- validate, [50](#)
- value_type, [41](#)
- ContainerConPtr
 - AMORE.h, [86](#)
- ContainerNeuronPtr
 - AMORE.h, [87](#)
- empty
 - Container, [44](#)
- end
 - Container, [44](#)
- erase
 - ConContainer, [23](#)
- foreach
 - AMORE.h, [86](#)
- from
 - Con, [18](#)
- getConId
 - Neuron, [70](#)
 - NeuronContainer, [79](#)
- getFrom
 - Con, [12](#)
 - NeuronContainer, [80](#)
- getId
 - Con, [13](#)
 - ConContainer, [25](#)
 - MLPneuronContainer, [65](#)
 - Neuron, [70](#)
 - NeuronContainer, [80](#)
- getWeight
 - Con, [14](#)
 - Neuron, [71](#)
 - NeuronContainer, [80](#)
- Id
 - Neuron, [75](#)
- iterator
 - ConContainer, [23](#)
- Container, [41](#)
- NeuronContainer, [78](#)
- load
 - Container, [45](#)
- MLPlayer, [51](#)
- MLPlayerContainer, [54](#)
- MLPneuralNet, [57](#)
 - nLayers, [59](#)
- MLPneuron, [59](#)
 - bias, [62](#)
- MLPneuronContainer, [62](#)
 - buildAndAppend, [65](#)
 - getId, [65](#)
- NeuralNet, [65](#)
 - train, [66](#)
- Neuron, [66](#)
 - ~Neuron, [70](#)
 - con, [74](#)
 - getConId, [70](#)
 - getId, [70](#)
 - getWeight, [71](#)
 - Id, [75](#)
 - Neuron, [69](#), [70](#)
 - numOfCons, [71](#)
 - outputValue, [75](#)
 - setFrom, [72](#)
 - setId, [72](#)
 - setWeight, [73](#)
 - show, [73](#)
 - validate, [74](#)
- NeuronContainer, [75](#)
 - ~NeuronContainer, [79](#)
 - const_iterator, [78](#)
 - const_reference, [78](#)
 - getConId, [79](#)
 - getFrom, [80](#)
 - getId, [80](#)
 - getWeight, [80](#)
 - iterator, [78](#)
 - NeuronContainer, [79](#)
 - numOfCons, [80](#)
 - numOfNeurons, [80](#)
 - setFrom, [81](#)
 - setId, [81](#)
 - setWeight, [81](#)
 - value_type, [79](#)
- NeuronContainer_const_iterator

- AMORE.h, [87](#)
- NeuronContainer_iterator
 - AMORE.h, [87](#)
- NeuronContainerPtr
 - AMORE.h, [87](#)
- NeuronPtr
 - AMORE.h, [87](#)
- NeuronWeakPtr
 - AMORE.h, [87](#)
- nLayers
 - MLPneuralNet, [59](#)
- numOfCons
 - ConContainer, [27](#)
 - Neuron, [71](#)
 - NeuronContainer, [80](#)
- numOfNeurons
 - NeuronContainer, [80](#)
- operator()
 - CompareId, [9, 10](#)
- outputValue
 - Neuron, [75](#)
- pkg/AMORE/src/AMORE.h, [85](#)
- pkg/AMORE/src/Con.cpp, [87](#)
- pkg/AMORE/src/Con.h, [89](#)
- pkg/AMORE/src/ConContainer.cpp, [89](#)
- pkg/AMORE/src/ConContainer.h, [90](#)
- pkg/AMORE/src/Container.cpp, [90](#)
- pkg/AMORE/src/Container.h, [91](#)
- pkg/AMORE/src/MLPlayer.h, [91](#)
- pkg/AMORE/src/MLPlayerContainer.h, [91](#)
- pkg/AMORE/src/MLPneuralNet.h, [91](#)
- pkg/AMORE/src/MLPneuralNetFactory.cpp, [92](#)
- pkg/AMORE/src/MLPneuron.h, [92](#)
- pkg/AMORE/src/MLPneuronContainer.h, [92](#)
- pkg/AMORE/src/NeuralNet.h, [92](#)
- pkg/AMORE/src/Neuron.cpp, [92](#)
- pkg/AMORE/src/Neuron.h, [93](#)
- pkg/AMORE/src/NeuronContainer.cpp, [94](#)
- pkg/AMORE/src/NeuronContainer.h, [94](#)
- pkg/AMORE/src/RBFneuralNet.h, [94](#)
- push_back
 - Container, [46](#)
- RBFneuralNet, [82](#)
- reserve
 - Container, [47](#)
- resize
 - Container, [47](#)
- select
 - ConContainer, [28](#)
- setFrom
 - Con, [15](#)
 - ConContainer, [30](#)
 - Neuron, [72](#)
 - NeuronContainer, [81](#)
- setId
 - Neuron, [72](#)
 - NeuronContainer, [81](#)
- setWeight
 - Con, [15](#)
 - ConContainer, [32, 34](#)
 - Neuron, [73](#)
 - NeuronContainer, [81](#)
- show
 - Con, [16](#)
 - Container, [47](#)
 - Neuron, [73](#)
- size
 - Container, [49](#)
- size_type
 - AMORE.h, [86](#)
- store
 - Container, [49](#)
- train
 - NeuralNet, [66](#)
- validate
 - Con, [17](#)
 - ConContainer, [36](#)
 - Container, [50](#)
 - Neuron, [74](#)
- value_type
 - ConContainer, [23](#)
 - Container, [41](#)
 - NeuronContainer, [79](#)
- weight
 - Con, [18](#)