

AMORE++

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

Generated by Doxygen 1.7.4

Mon Jun 13 2011 18:20:30

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	12

5.2.2.3	Con	12
5.2.2.4	Con	12
5.2.2.5	Con	12
5.2.2.6	Con	12
5.2.2.7	~Con	12
5.2.3	Member Function Documentation	12
5.2.3.1	getFrom	12
5.2.3.2	getFrom	13
5.2.3.3	getId	13
5.2.3.4	getId	14
5.2.3.5	getWeight	14
5.2.3.6	getWeight	14
5.2.3.7	setFrom	15
5.2.3.8	setFrom	16
5.2.3.9	setWeight	16
5.2.3.10	setWeight	17
5.2.3.11	show	17
5.2.3.12	show	17
5.2.3.13	validate	18
5.2.3.14	validate	19
5.2.4	Member Data Documentation	19
5.2.4.1	from	19
5.2.4.2	weight	19
5.3	ConContainer Class Reference	20
5.3.1	Detailed Description	23
5.3.2	Member Typedef Documentation	23
5.3.2.1	const_iterator	23
5.3.2.2	const_reference	24
5.3.2.3	iterator	24
5.3.2.4	value_type	24
5.3.3	Constructor & Destructor Documentation	24
5.3.3.1	ConContainer	24
5.3.3.2	ConContainer	24
5.3.4	Member Function Documentation	24

5.3.4.1	erase	24
5.3.4.2	getId	26
5.3.4.3	numOfCons	28
5.3.4.4	select	29
5.3.4.5	setFrom	31
5.3.4.6	setWeight	33
5.3.4.7	setWeight	35
5.3.4.8	validate	36
5.4	Container< T > Class Template Reference	37
5.4.1	Detailed Description	41
5.4.2	Member Typedef Documentation	41
5.4.2.1	const_iterator	41
5.4.2.2	const_iterator	41
5.4.2.3	const_reference	41
5.4.2.4	const_reference	42
5.4.2.5	iterator	42
5.4.2.6	iterator	42
5.4.2.7	value_type	42
5.4.2.8	value_type	42
5.4.3	Constructor & Destructor Documentation	42
5.4.3.1	Container	42
5.4.3.2	Container	43
5.4.3.3	Container	43
5.4.3.4	Container	43
5.4.4	Member Function Documentation	43
5.4.4.1	append	43
5.4.4.2	begin	45
5.4.4.3	begin	46
5.4.4.4	buildAndAppend	46
5.4.4.5	clear	47
5.4.4.6	clear	47
5.4.4.7	empty	47
5.4.4.8	empty	47
5.4.4.9	end	47

5.4.4.10	end	48
5.4.4.11	erase	48
5.4.4.12	insert	48
5.4.4.13	load	49
5.4.4.14	operator[]	50
5.4.4.15	push_back	50
5.4.4.16	push_back	51
5.4.4.17	reserve	51
5.4.4.18	reserve	51
5.4.4.19	resize	51
5.4.4.20	resize	51
5.4.4.21	show	51
5.4.4.22	show	53
5.4.4.23	size	53
5.4.4.24	size	53
5.4.4.25	store	53
5.4.4.26	validate	54
5.4.4.27	validate	54
5.4.5	Member Data Documentation	54
5.4.5.1	collection	54
5.5	MLPlayer Class Reference	55
5.5.1	Detailed Description	58
5.6	MLPlayerContainer Class Reference	58
5.6.1	Detailed Description	61
5.7	MLPneuralNet Class Reference	61
5.7.1	Detailed Description	63
5.7.2	Member Data Documentation	63
5.7.2.1	nLayers	63
5.8	MLPneuron Class Reference	63
5.8.1	Detailed Description	66
5.8.2	Member Data Documentation	66
5.8.2.1	bias	66
5.9	MLPneuronContainer Class Reference	66
5.9.1	Detailed Description	69

5.9.2	Member Function Documentation	69
5.9.2.1	buildAndAppend	69
5.9.2.2	getId	69
5.10	NeuralNet Class Reference	69
5.10.1	Detailed Description	70
5.10.2	Member Function Documentation	70
5.10.2.1	train	70
5.11	Neuron Class Reference	70
5.11.1	Detailed Description	73
5.11.2	Constructor & Destructor Documentation	74
5.11.2.1	Neuron	74
5.11.2.2	Neuron	74
5.11.2.3	Neuron	74
5.11.2.4	Neuron	75
5.11.2.5	Neuron	75
5.11.2.6	Neuron	75
5.11.2.7	~Neuron	75
5.11.3	Member Function Documentation	75
5.11.3.1	const_iterators	75
5.11.3.2	getConId	75
5.11.3.3	getId	75
5.11.3.4	getId	76
5.11.3.5	getWeight	76
5.11.3.6	iterators	76
5.11.3.7	numOfCons	77
5.11.3.8	setFrom	77
5.11.3.9	setId	77
5.11.3.10	setId	77
5.11.3.11	setWeight	77
5.11.3.12	show	77
5.11.3.13	show	78
5.11.3.14	validate	78
5.11.3.15	validate	78
5.11.4	Member Data Documentation	79

5.11.4.1	con	79
5.11.4.2	ld	79
5.11.4.3	nCons	79
5.11.4.4	outputValue	79
5.12	NeuronContainer Class Reference	79
5.12.1	Detailed Description	82
5.12.2	Member Typedef Documentation	82
5.12.2.1	const_iterator	82
5.12.2.2	const_reference	82
5.12.2.3	iterator	83
5.12.2.4	value_type	83
5.12.3	Constructor & Destructor Documentation	83
5.12.3.1	NeuronContainer	83
5.12.3.2	NeuronContainer	83
5.12.3.3	~NeuronContainer	83
5.12.4	Member Function Documentation	83
5.12.4.1	getConId	83
5.12.4.2	getFrom	84
5.12.4.3	getId	84
5.12.4.4	getWeight	84
5.12.4.5	numOfCons	84
5.12.4.6	numOfNeurons	85
5.12.4.7	setFrom	85
5.12.4.8	setId	85
5.12.4.9	setWeight	86
5.13	RBFneuralNet Class Reference	86
5.13.1	Detailed Description	87
6	File Documentation	89
6.1	pkg/AMORE/src/AMORE.h File Reference	89
6.1.1	Define Documentation	90
6.1.1.1	foreach	90
6.1.1.2	size_type	90
6.1.2	Typedef Documentation	90

6.1.2.1	ConContainer	90
6.1.2.2	ConContainer_const_iterator	90
6.1.2.3	ConContainer_iterator	91
6.1.2.4	ConContainerPtr	91
6.1.2.5	ConPtr	91
6.1.2.6	NeuronContainer	91
6.1.2.7	NeuronContainer_const_iterator	91
6.1.2.8	NeuronContainer_iterator	91
6.1.2.9	NeuronContainerPtr	91
6.1.2.10	NeuronPtr	91
6.1.2.11	NeuronWeakPtr	91
6.2	pkg/AMORE/src/Con.cpp File Reference	91
6.3	pkg/AMORE/src/old/Con.cpp File Reference	92
6.4	pkg/AMORE/src/Con.h File Reference	93
6.5	pkg/AMORE/src/old/Con.h File Reference	94
6.6	pkg/AMORE/src/Container.cpp File Reference	94
6.7	pkg/AMORE/src/old/Container.cpp File Reference	95
6.8	pkg/AMORE/src/Container.h File Reference	95
6.9	pkg/AMORE/src/old/Container.h File Reference	95
6.10	pkg/AMORE/src/Neuron.cpp File Reference	95
6.11	pkg/AMORE/src/old/Neuron.cpp File Reference	96
6.12	pkg/AMORE/src/Neuron.h File Reference	97
6.13	pkg/AMORE/src/old/Neuron.h File Reference	98
6.14	pkg/AMORE/src/old/ConContainer.cpp File Reference	98
6.15	pkg/AMORE/src/old/ConContainer.h File Reference	98
6.16	pkg/AMORE/src/old/MLPlayer.h File Reference	98
6.17	pkg/AMORE/src/old/MLPlayerContainer.h File Reference	99
6.18	pkg/AMORE/src/old/MLPneuralNet.h File Reference	99
6.19	pkg/AMORE/src/old/MLPneuralNetFactory.cpp File Reference	99
6.19.1	Function Documentation	99
6.19.1.1	CreateMLPneuralNet	99
6.20	pkg/AMORE/src/old/MLPneuron.h File Reference	100
6.21	pkg/AMORE/src/old/MLPneuronContainer.h File Reference	100
6.22	pkg/AMORE/src/old/NeuralNet.h File Reference	100

6.23	pkg/AMORE/src/old/NeuronContainer.cpp File Reference	100
6.24	pkg/AMORE/src/old/NeuronContainer.h File Reference	100
6.25	pkg/AMORE/src/old/RBFneuralNet.h File Reference	100

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	20
Container< MLPlayer >	37
MLPlayerContainer	58
Container< Neuron >	37
NeuronContainer	79
NeuralNet	69
MLPneuralNet	61
RBFneuralNet	86
Neuron	70
MLPneuron	63
NeuronContainer< MLP >	79
MLPneuronContainer	66
MLPlayer	55

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)	20
Container< T > (Class Container -)	37
MLPlayer	55
MLPlayerContainer	58
MLPneuralNet	61
MLPneuron	63
MLPneuronContainer (A vector of connections)	66
NeuralNet	69
Neuron (Class Neuron -)	70
NeuronContainer (A vector of neurons)	79
RBFneuralNet	86

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

pkg/AMORE/src/AMORE.h	89
pkg/AMORE/src/Con.cpp	91
pkg/AMORE/src/Con.h	93
pkg/AMORE/src/Container.cpp	94
pkg/AMORE/src/Container.h	95
pkg/AMORE/src/Neuron.cpp	95
pkg/AMORE/src/Neuron.h	97
pkg/AMORE/src/old/Con.cpp	92
pkg/AMORE/src/old/Con.h	94
pkg/AMORE/src/old/ConContainer.cpp	98
pkg/AMORE/src/old/ConContainer.h	98
pkg/AMORE/src/old/Container.cpp	95
pkg/AMORE/src/old/Container.h	95
pkg/AMORE/src/old/MLPlayer.h	98
pkg/AMORE/src/old/MLPlayerContainer.h	99
pkg/AMORE/src/old/MLPneuralNet.h	99
pkg/AMORE/src/old/MLPneuralNetFactory.cpp	99
pkg/AMORE/src/old/MLPneuron.h	100
pkg/AMORE/src/old/MLPneuronContainer.h	100
pkg/AMORE/src/old/NeuralNet.h	100
pkg/AMORE/src/old/Neuron.cpp	96
pkg/AMORE/src/old/Neuron.h	98
pkg/AMORE/src/old/NeuronContainer.cpp	100
pkg/AMORE/src/old/NeuronContainer.h	100
pkg/AMORE/src/old/RBFneuralNet.h	100

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/old/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 \(NeuronPtr neuronPtr, double weight\)](#)
Constructor.
- [NeuronPtr getFrom \(\)](#)
from field accessor.
- [int getId \(\)](#)
A getter of the Id of the [Neuron](#) pointed by the from field.
- [double getWeight \(\)](#)
weight field accessor.
- [void setFrom \(NeuronPtr neuronPtr\)](#)
from field accessor.
- [void setWeight \(double weight\)](#)
weight field accessor.

- void [show](#) ()
Pretty print of the [Con](#) information.
- bool [validate](#) ()
Object validator.
- [Con](#) ()
- [Con](#) ([NeuronPtr](#) neuronPtr)
- [Con](#) ([NeuronPtr](#) neuronPtr, double value)
- [~Con](#) ()
Default Destructor.
- [NeuronPtr](#) [getFrom](#) ()
- void [setFrom](#) ([NeuronPtr](#) neuronPtr)
- int [getId](#) ()
- double [getWeight](#) ()
- void [setWeight](#) (double value)
- bool [show](#) ()
- bool [validate](#) ()

Protected 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 3 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 28 of file Con.cpp.

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

5.2.2.3 Con::Con (NeuronPtr *neuronPtr*, double *weight*)

Constructor.

Definition at line 40 of file Con.cpp.

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

5.2.2.4 Con::Con ()

5.2.2.5 Con::Con (NeuronPtr *neuronPtr*)

5.2.2.6 Con::Con (NeuronPtr *neuronPtr*, double *value*)

5.2.2.7 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 67 of file `Con.cpp`.

References from.

```

{
    return (from.lock());
}

```

5.2.3.2 NeuronPtr Con::getFrom ()**5.2.3.3 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 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.

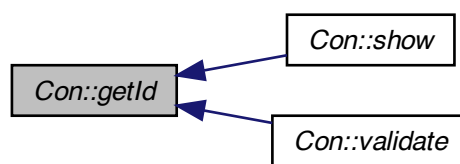
Definition at line 94 of file Con.cpp.

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:



5.2.3.4 int Con::getId ()

5.2.3.5 double Con::getWeight ()

5.2.3.6 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 160 of file `Con.cpp`.

References `weight`.

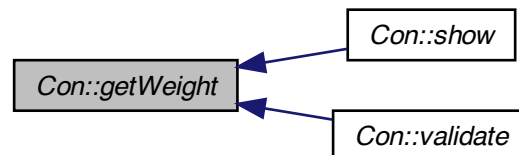
Referenced by `show()`, and `validate()`.

```

{
    return (weight);
}

```

Here is the caller graph for this function:

**5.2.3.7 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 131 of file `Con.cpp`.

References from.

```

{
    from = neuronPtr;
}

```

5.2.3.8 void Con::setFrom (NeuronPtr neuronPtr)

5.2.3.9 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 188 of file `Con.cpp`.

References `weight`.

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

5.2.3.10 `void Con::setWeight (double value)`

5.2.3.11 `bool Con::show ()`

5.2.3.12 `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

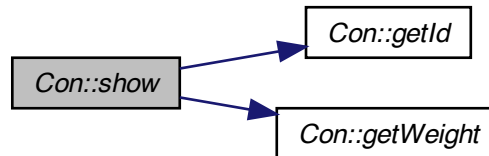
[setWeight](#) and the unit test files, e.g., `runit.Cpp.Con.R`, for usage examples.

Definition at line 199 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());  
    }  
}
```

Here is the call graph for this function:



5.2.3.13 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

<i>An</i>	std::range error if weight or from are not finite.
-----------	--

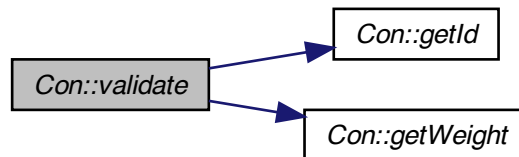
Definition at line 219 of file Con.cpp.

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:



5.2.3.14 `bool Con::validate ()`

5.2.4 Member Data Documentation

5.2.4.1 `NeuronWeakPtr Con::from` `[protected]`

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 7 of file `Con.h`.

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

5.2.4.2 `double Con::weight` `[protected]`

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 8 of file `Con.h`.

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

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

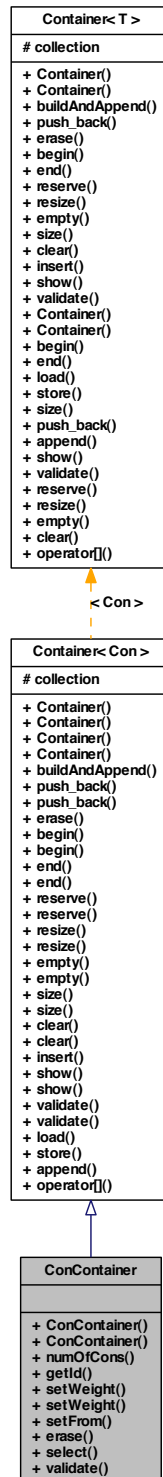
- `pkg/AMORE/src/Con.h`
- `pkg/AMORE/src/old/Con.h`
- `pkg/AMORE/src/Con.cpp`
- `pkg/AMORE/src/old/Con.cpp`

5.3 ConContainer Class Reference

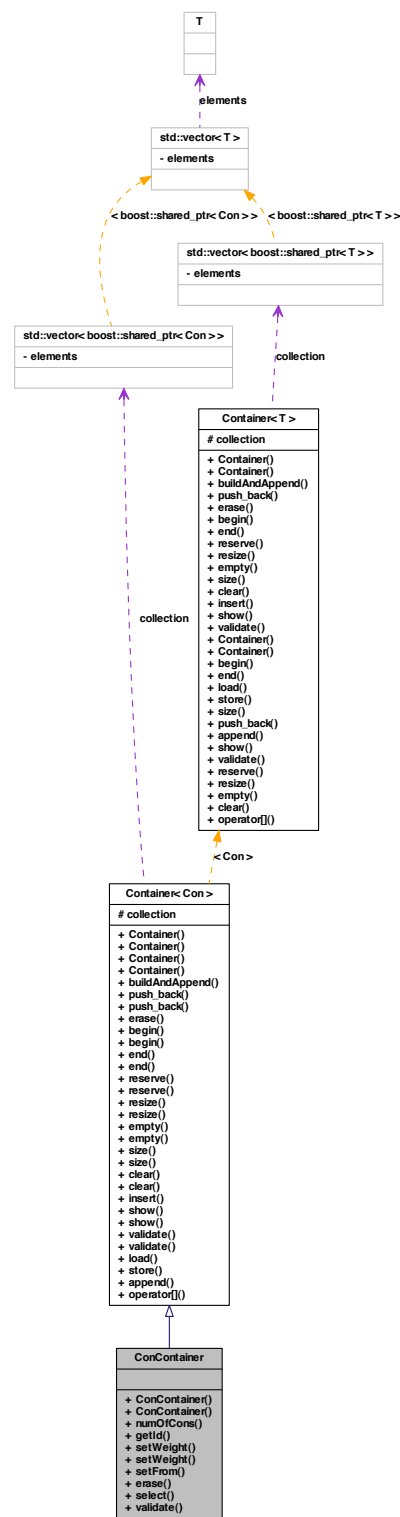
A vector of connections.

```
#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.

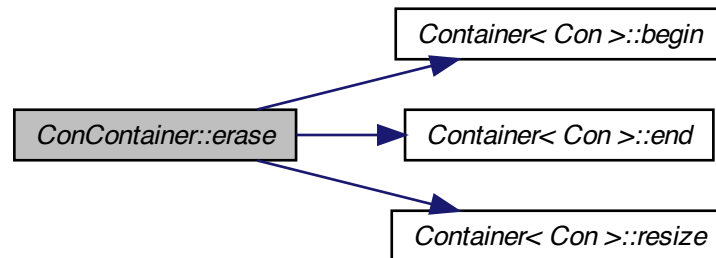
Definition at line 450 of file `ConContainer.cpp`.

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()`.

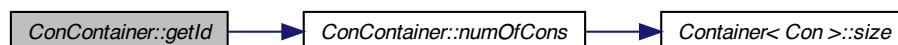
Referenced by `validate()`.

```

{
    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->numOfCon
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()`.

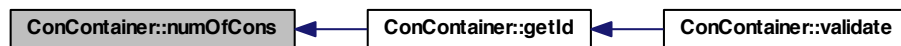
Referenced by `getId()`.

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

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.4.4 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()`.

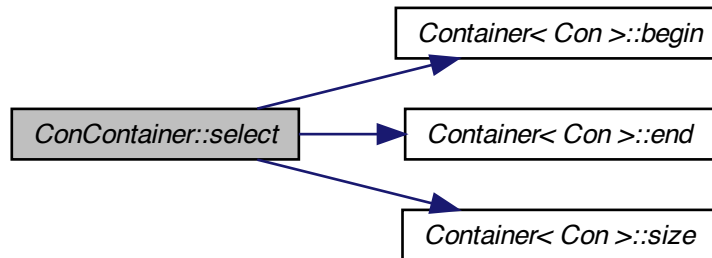
```

{
    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()`.

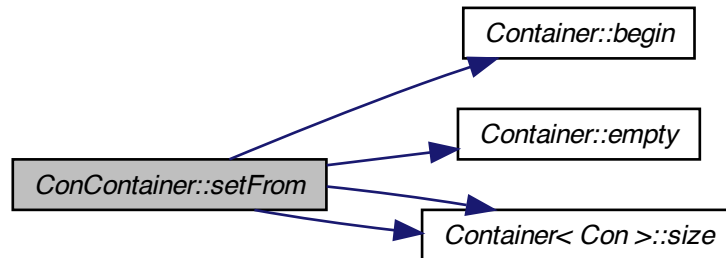
```

{
    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:



5.3.4.6 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 std::vector<int> 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.

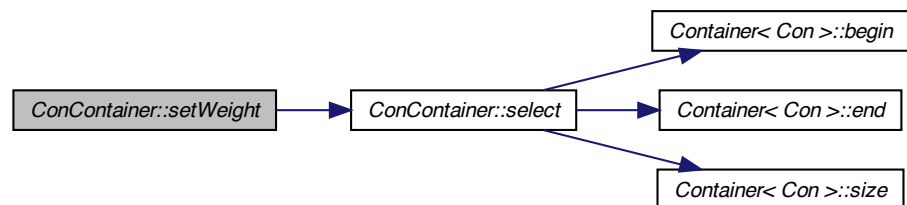
References `select()`.

```

{
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()`.

```
{
    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:



5.3.4.8 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().

```
{
    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:

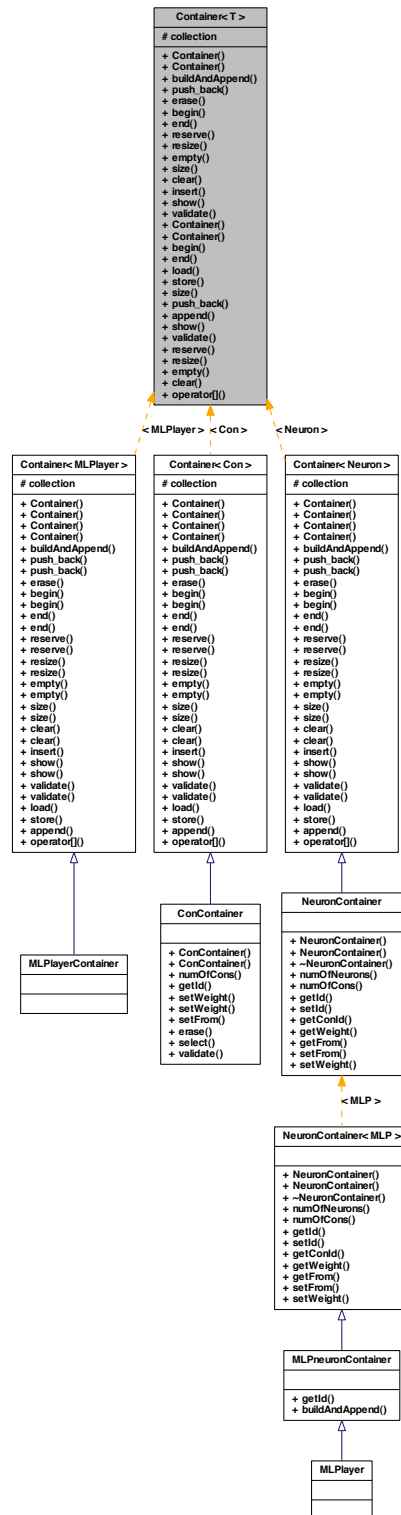
- pkg/AMORE/src/old/[ConContainer.h](#)
- pkg/AMORE/src/old/[ConContainer.cpp](#)

5.4 Container< T > Class Template Reference

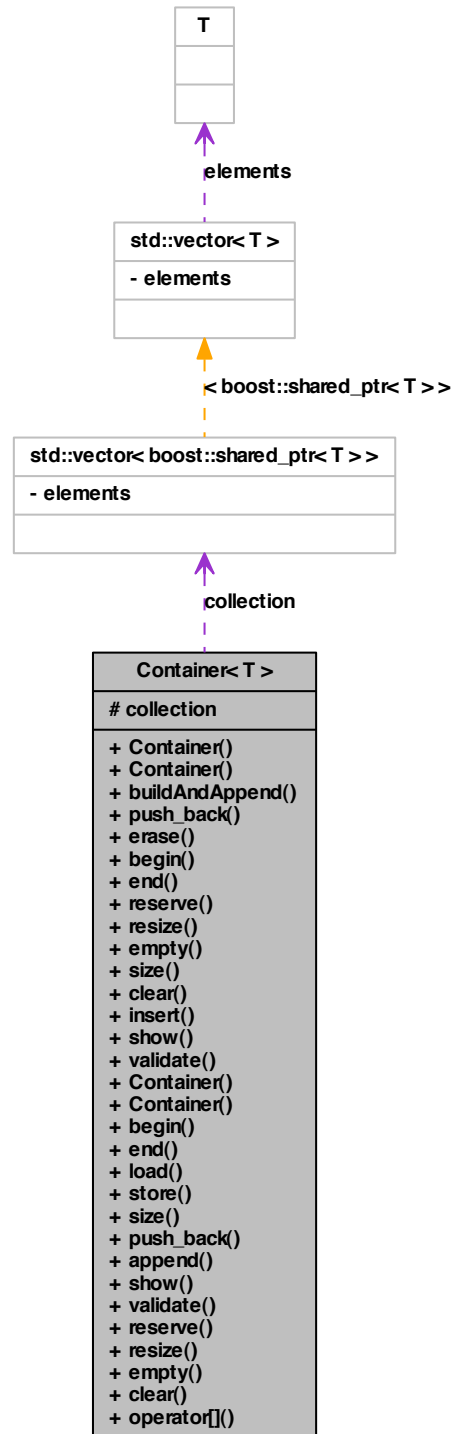
class [Container](#) -

```
#include <Container.h>
```

Inheritance diagram for Container< T >:



Collaboration diagram for Container< T >:



Public Types

- typedef boost::shared_ptr< T > [value_type](#)
- typedef boost::shared_ptr< T > const & [const_reference](#)
- typedef std::vector< boost::shared_ptr< T > >::iterator [iterator](#)
- typedef std::vector< boost::shared_ptr< T > >::const_iterator [const_iterator](#)
- 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 > >::iterator first, typename std::vector< boost::shared_ptr< T > >::iterator last)
- bool [buildAndAppend](#) ([NeuronContainer_iterator](#) firstNeuron, [NeuronContainer_iterator](#) lastNeuron, std::vector< double >::iterator firstWeight, std::vector< double >::iterator lastWeight)
- void [push_back](#) ([value_type](#) const &[const_reference](#))
Append a shared_ptr at the end of collection.
- void [erase](#) ([iterator](#) first, [iterator](#) last)
- [iterator](#) [begin](#) ()
- [iterator](#) [end](#) ()
- void [reserve](#) (int n)
- void [resize](#) (int n)
- void [empty](#) ()
- size_type [size](#) ()
Returns the size or length of the vector.
- void [clear](#) ()
- void [insert](#) ([iterator](#) position, [const_iterator](#) first, [const_iterator](#) last)
- 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](#))

- 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< boost::shared_ptr< T > > [collection](#)

5.4.1 Detailed Description

template<typename T>class Container< T >

class [Container](#) -

Definition at line 13 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 21 of file Container.h.

5.4.2.2 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.

5.4.2.3 template<typename T> typedef boost::shared_ptr<T> const& Container< T
>::const_reference

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

Definition at line 18 of file Container.h.

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

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

Definition at line 26 of file Container.h.

5.4.2.5 `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.

5.4.2.6 `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.

5.4.2.7 `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.

5.4.2.8 `template<typename T> typedef boost::shared_ptr<T> Container< T
>::value_type`

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

Definition at line 17 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 > >::iterator first, typename std::vector< boost::shared_ptr< T > >::iterator last)`

Definition at line 14 of file Container.cpp.

References Container< T >::collection.

```

                                :
    collection()
    {
        collection.insert(collection.begin(), first, last);
    }

```

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

5.4.3.4 `template<typename T> Container< T >::Container (std::vector< value > first, std::vector< value > last)`

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 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

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`.

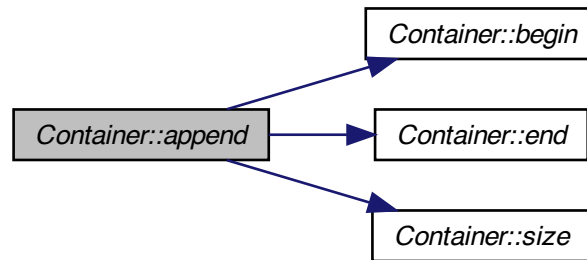
References `Container< T >::begin()`, `Container< T >::end()`, and `Container< T >::size()`.

```

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

```

Here is the call graph for this function:



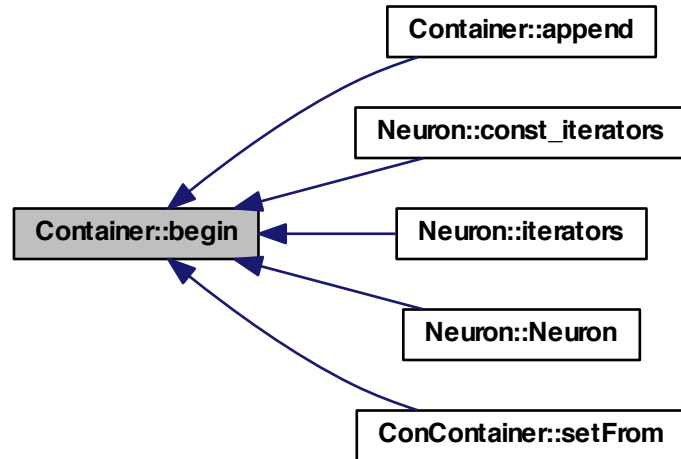
5.4.4.2 `template<typename T> std::vector< boost::shared_ptr< T >>::iterator Container< T >::begin ()`

Definition at line 24 of file `Container.cpp`.

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

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

Here is the caller graph for this function:



5.4.4.3 `template<typename T> iterator Container< T >::begin ()`

5.4.4.4 `template<typename T > bool Container< T >::buildAndAppend (
NeuronContainer_iterator firstNeuron, NeuronContainer_iterator lastNeuron,
std::vector< double >::iterator firstWeight, std::vector< double >::iterator
lastWeight)`

Definition at line 207 of file Container.cpp.

```

{
    BEGIN_RCPP

    bool emptyNeuronContainer = (firstNeuron==lastNeuron);
    if (emptyNeuronContainer)
    {
        throw std::range_error(
            "[ConContainer::BuildAndAppend]: Error, neuronContainer is empty");
    }

    bool differentSize = (lastNeuron - firstNeuron) != (lastWeight - firstWeight);
    if (differentSize)
    {
        throw std::range_error(
            "[ConContainer::buildAndAppend]: Error, neuronContainer.size() != nWeig
            hts.size()");
    }
}

```



```

reserve(size() + (lastNeuron - firstNeuron));
ConPtr ptCon;
while (firstNeuron != lastNeuron)
{
    ptCon.reset(new Con(*firstNeuron++, *firstWeight++));
    push_back(ptCon);
}
return true;
END_RCPP}

```

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

Definition at line 191 of file Container.cpp.

```

{
    collection.clear();
}

```

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

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

Definition at line 184 of file Container.cpp.

Referenced by `ConContainer::setFrom()`.

```

{
    return (collection.empty());
}

```

Here is the caller graph for this function:



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

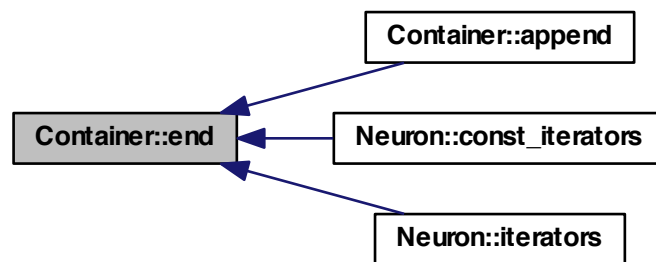
5.4.4.9 `template<typename T> std::vector< boost::shared_ptr< T > >::iterator Container< T >::end ()`

Definition at line 31 of file Container.cpp.

Referenced by `Container< T >::append()`, `Neuron::const_iterators()`, and `Neuron::iterators()`.

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

Here is the caller graph for this function:



5.4.4.10 `template<typename T> iterator Container< T >::end ()`

5.4.4.11 `template<typename T> void Container< T >::erase (iterator first, iterator last)`

5.4.4.12 `template<typename T> void Container< T >::insert (iterator position,
const_iterator first, const_iterator last)`

Referenced by `Neuron::Neuron()`.

Here is the caller graph for this function:



5.4.4.13 `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;
}
```

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

Definition at line 317 of file Container.cpp.

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

5.4.4.15 `template<typename T> void Container< T>::push_back (value_type 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 73 of file Container.cpp.

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

5.4.4.16 `template<typename T> void Container< T >::push_back (boost::shared_ptr< T > const & const_reference)`

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

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

Definition at line 177 of file Container.cpp.

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

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

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

Definition at line 170 of file Container.cpp.

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

5.4.4.21 `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 124 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();
    }
}

```

Here is the caller graph for this function:



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

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

Returns the size or length of the vector.

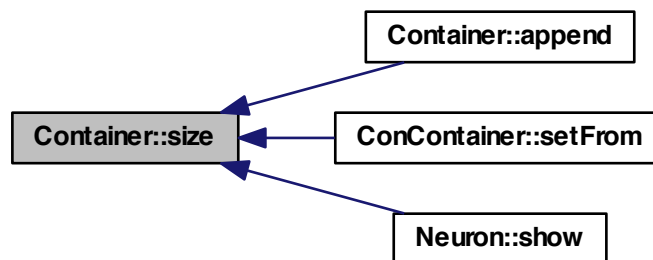
This method returns the size of the vector. In the classes derived from Container<T> this is aliased as numOfCons, numOfNeurons and numOfLayers. The unit test files, e.g., runit.Cpp.Container.R, for usage examples.

Definition at line 162 of file Container.cpp.

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

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

Here is the caller graph for this function:



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

5.4.4.25 `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;
}
```

5.4.4.26 `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;
}
```

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

Reimplemented in [ConContainer](#).

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 23 of file `Container.h`.

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

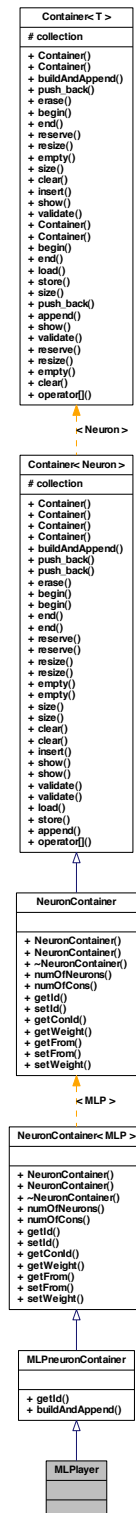
The documentation for this class was generated from the following files:

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

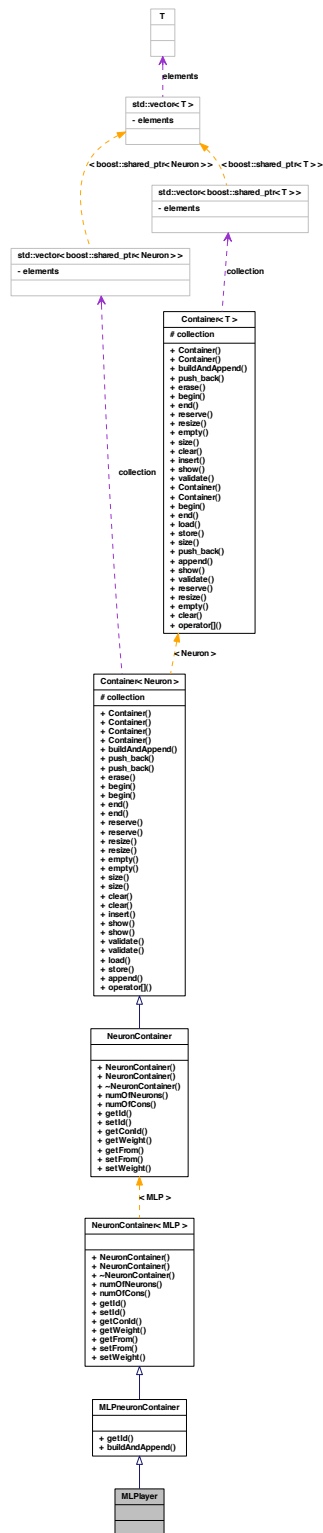
5.5 MLPlayer Class Reference

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

Inheritance diagram for MLPlayer:



Collaboration diagram for MLPlayer:



5.5.1 Detailed Description

Definition at line 1 of file MLPlayer.h.

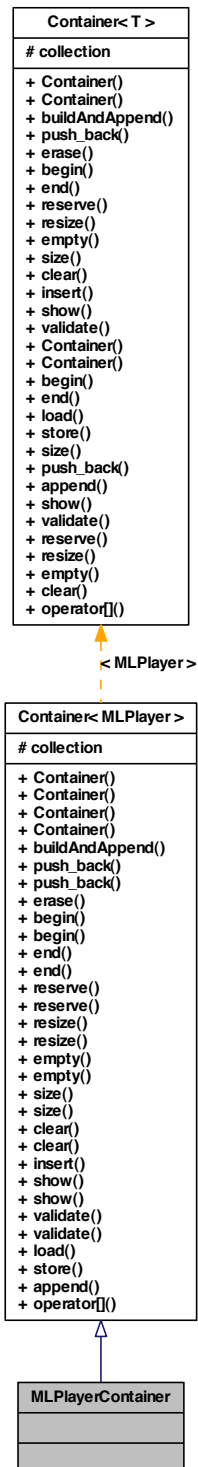
The documentation for this class was generated from the following file:

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

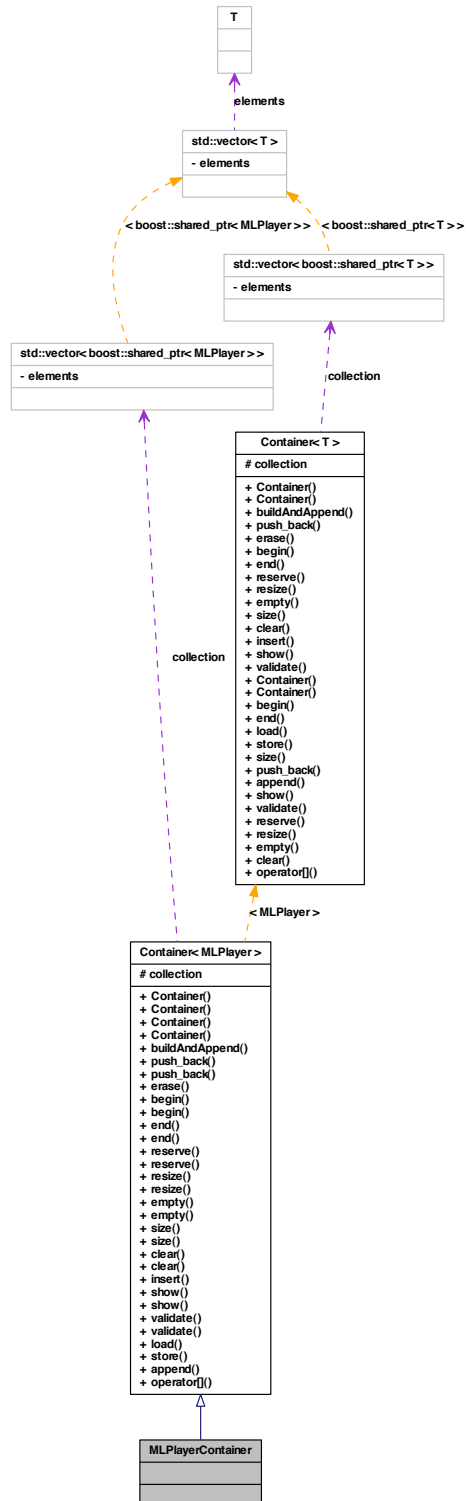
5.6 MLPlayerContainer Class Reference

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

Inheritance diagram for MLPlayerContainer:



Collaboration diagram for MLPlayerContainer:



5.6.1 Detailed Description

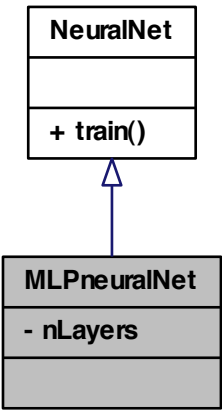
Definition at line 1 of file MLPlayerContainer.h.
The documentation for this class was generated from the following file:

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

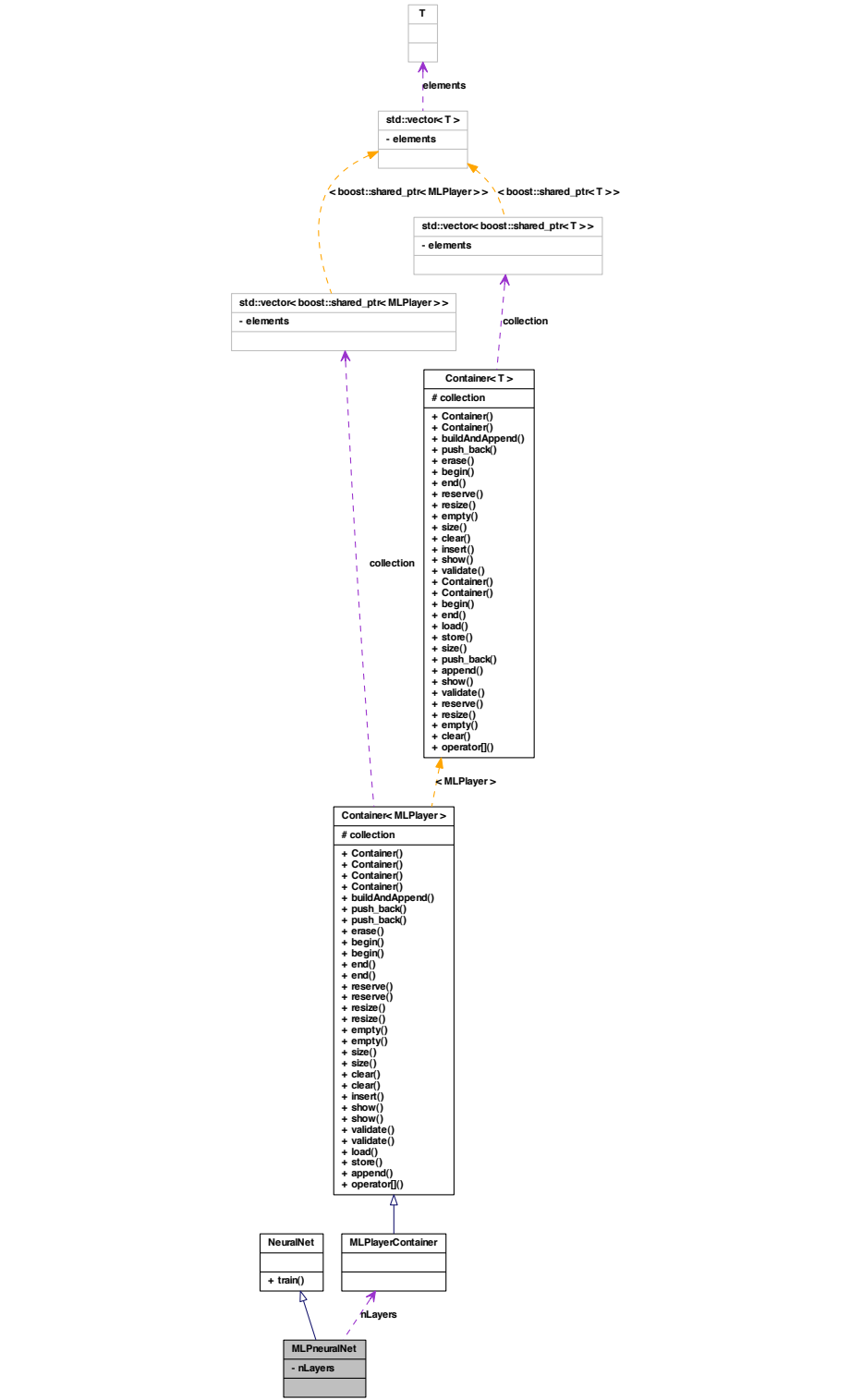
5.7 MLPneuralNet Class Reference

```
#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.

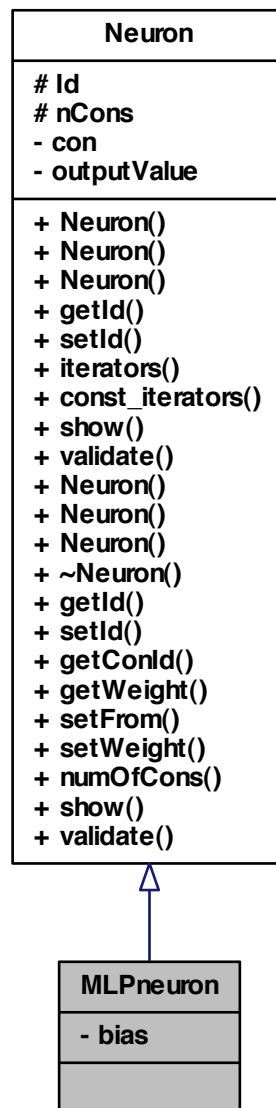
The documentation for this class was generated from the following file:

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

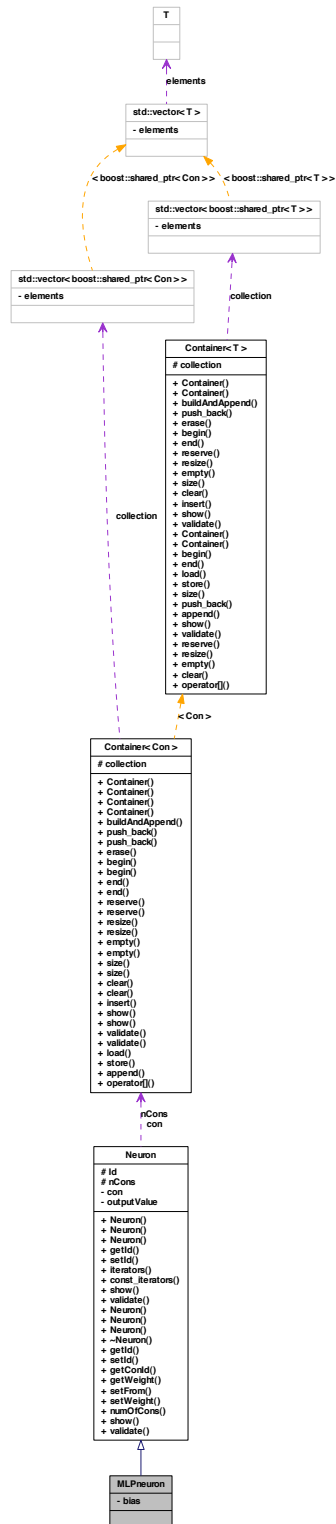
5.8 MLPneuron Class Reference

```
#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:

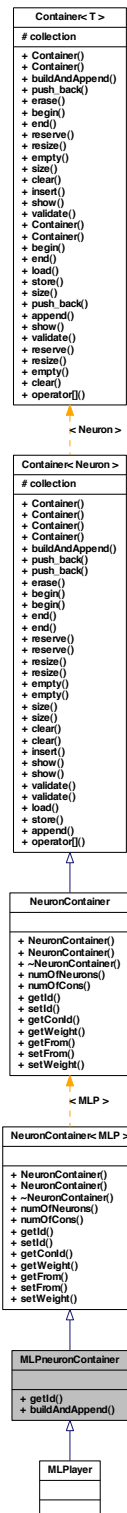
- pkg/AMORE/src/old/[MLPneuron.h](#)

5.9 MLPneuronContainer Class Reference

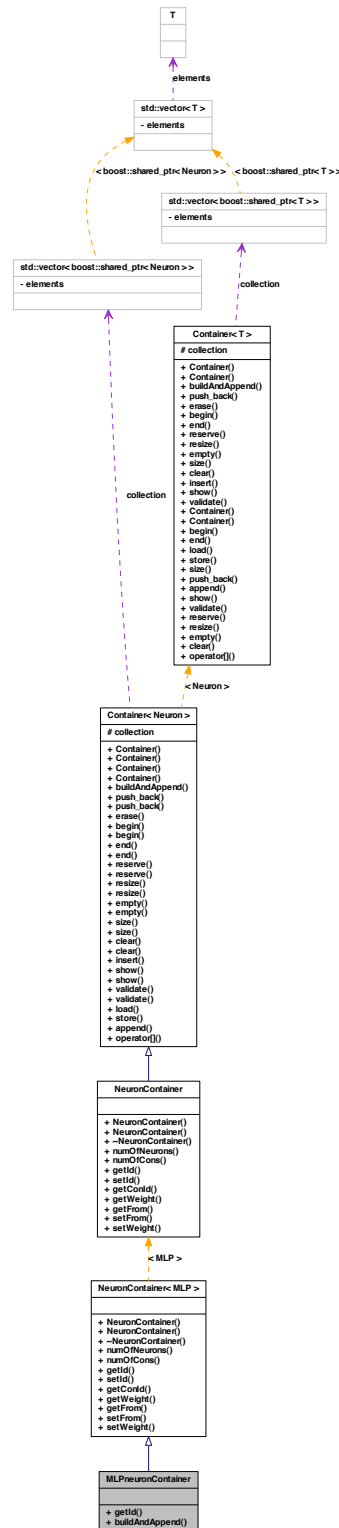
A vector of connections.

```
#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< MLP >](#).

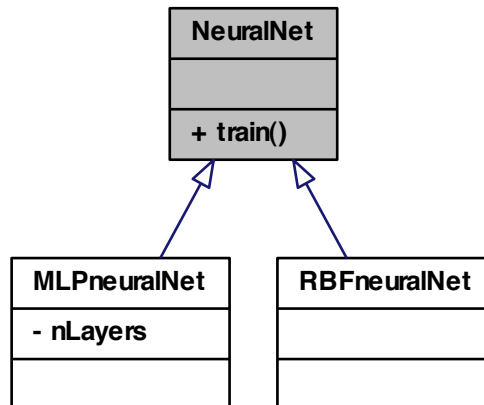
The documentation for this class was generated from the following file:

- `pkg/AMORE/src/old/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:

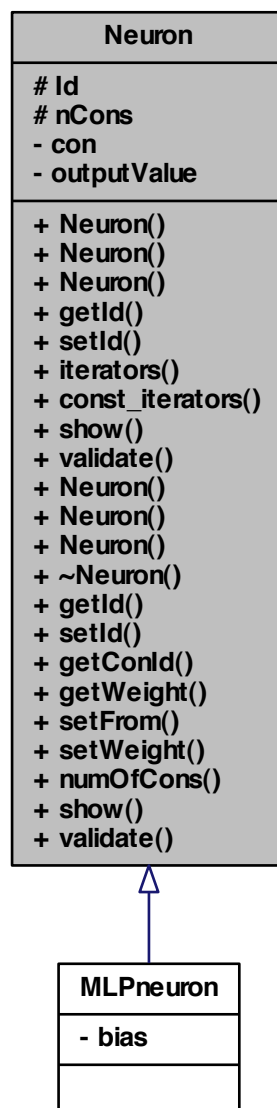
- pkg/AMORE/src/old/[NeuralNet.h](#)

5.11 Neuron Class Reference

class [Neuron](#) -

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


Inheritance diagram for Neuron:



Collaboration diagram for Neuron:



Public Member Functions

- [Neuron](#) ()
- [Neuron](#) (int [Id](#))
- [Neuron](#) (int [Id](#), [Container](#)< [Con](#) >::const_iterator firstCon, [Container](#)< [Con](#) >::const_iterator lastCon)
- int [getId](#) ()
- void [setId](#) (int [Id](#))
- void [iterators](#) ([Container](#)< [Con](#) >::iterator &first, [Container](#)< [Con](#) >::iterator &last)
- void [const_iterators](#) ([Container](#)< [Con](#) >::const_iterator &first, [Container](#)< [Con](#) >::const_iterator &last)
- 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 [Id](#)
An integer variable with the [Neuron](#) Id.
- [Container](#)< [Con](#) > [nCons](#)

Private Attributes

- [ConContainer](#) con
A vector of input connections.
- double [outputValue](#)

5.11.1 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 5 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), nCons()
    {
    }

```

5.11.2.2 Neuron::Neuron (int Id)

Definition at line 15 of file Neuron.cpp.

```

        :
        Id(Id), nCons()
    {
    }

```

5.11.2.3 Neuron::Neuron (int Id, Container< Con >::const_iterator firstCon, Container< Con >::const_iterator lastCon)

Definition at line 20 of file Neuron.cpp.

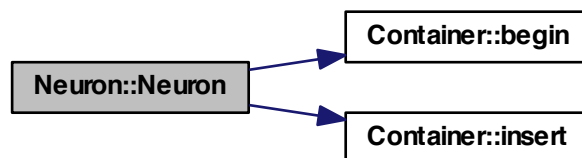
References Container< T >::begin(), Container< T >::insert(), and nCons.

```

        :
        Id(Id), nCons()
    {
        nCons.insert(nCons.begin(), firstCon, lastCon);
    }

```

Here is the call graph for this function:



5.11.2.4 `Neuron::Neuron ()`

5.11.2.5 `Neuron::Neuron (int Id)`

5.11.2.6 `Neuron::Neuron (int Id, ConContainer con)`

5.11.2.7 `Neuron::~~Neuron ()`

5.11.3 Member Function Documentation

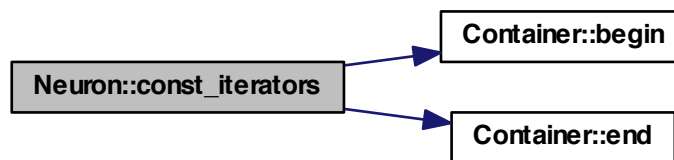
5.11.3.1 `void Neuron::const_iterators (Container< Con >::const_iterator & first, Container< Con >::const_iterator & last)`

Definition at line 50 of file Neuron.cpp.

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

```
{  
    first=nCons.begin();  
    last=nCons.end();  
}
```

Here is the call graph for this function:



5.11.3.2 `std::vector<int> Neuron::getConId ()`

5.11.3.3 `int Neuron::getId ()`

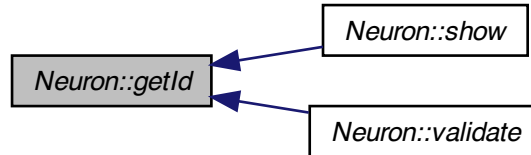
Definition at line 29 of file Neuron.cpp.

References `Id`.

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

```
{  
    return Id;  
}
```

Here is the caller graph for this function:



5.11.3.4 `int Neuron::getId ()`

5.11.3.5 `std::vector<double> Neuron::getWeight ()`

5.11.3.6 `void Neuron::iterators (Container< Con >::iterator & first, Container< Con >::iterator & last)`

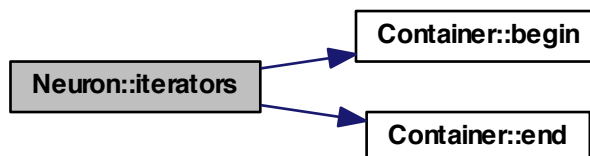
Definition at line 42 of file `Neuron.cpp`.

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

```

{
    first=nCons.begin();
    last=nCons.end();
}
  
```

Here is the call graph for this function:



5.11.3.7 `int Neuron::numOfCons ()`

5.11.3.8 `bool Neuron::setFrom (NeuronContainer neuronContainer)`

5.11.3.9 `void Neuron::setId (int Id)`

Definition at line 35 of file Neuron.cpp.

References `Id`.

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

5.11.3.10 `void Neuron::setId (int value)`

5.11.3.11 `bool Neuron::setWeight (std::vector< double > nWeights)`

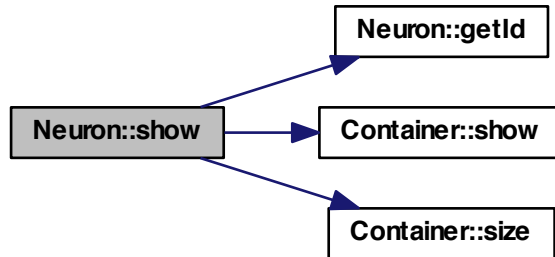
5.11.3.12 `bool Neuron::show ()`

Definition at line 58 of file Neuron.cpp.

References `getId()`, `nCons`, `Container< T >::show()`, and `Container< T >::size()`.

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

Here is the call graph for this function:



5.11.3.13 `bool Neuron::show ()`

5.11.3.14 `bool Neuron::validate ()`

5.11.3.15 `bool Neuron::validate ()`

Definition at line 84 of file `Neuron.cpp`.

References `getId()`.

```

{
    BEGIN_RCPP
    if (getId() == NA_INTEGER ) throw std::range_error("[C++ Neuron::validate]: Err
        or, Id is NA.");
    // nCons.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.

5.11.4.2 int Neuron::Id [protected]

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 8 of file Neuron.h.

Referenced by `getId()`, and `setId()`.

5.11.4.3 Container<Con> Neuron::nCons [protected]

Definition at line 9 of file Neuron.h.

Referenced by `const_iterators()`, `iterators()`, `Neuron()`, and `show()`.

5.11.4.4 double Neuron::outputValue [private]

Definition at line 30 of file Neuron.h.

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

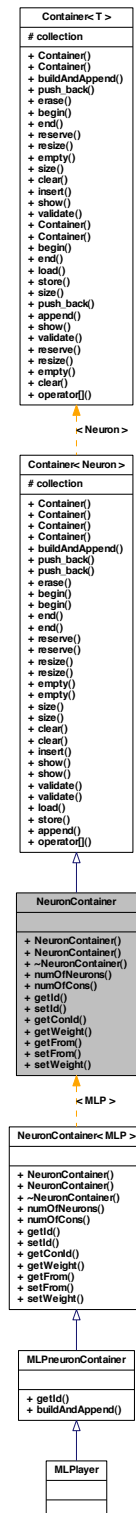
- `pkg/AMORE/src/Neuron.h`
- `pkg/AMORE/src/old/Neuron.h`
- `pkg/AMORE/src/Neuron.cpp`
- `pkg/AMORE/src/old/Neuron.cpp`

5.12 NeuronContainer Class Reference

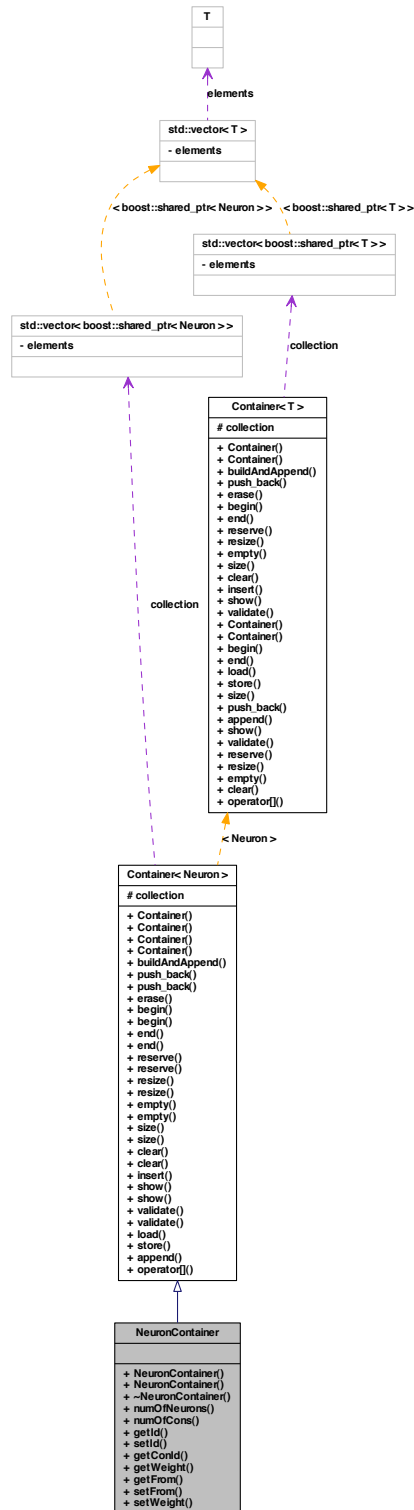
A vector of neurons.

```
#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 > 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)

5.12.1 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 > nIds)

Definition at line 50 of file NeuronContainer.cpp.

```
{
    std::vector<int>::iterator itrId(nIds.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++;
    }
}
```

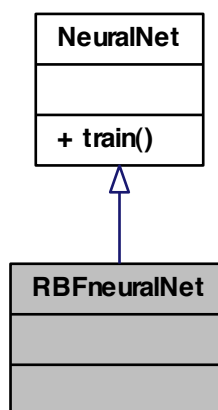
The documentation for this class was generated from the following files:

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

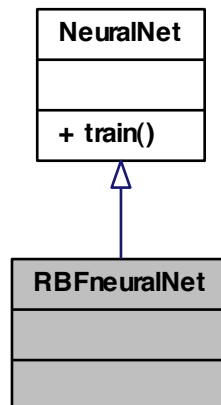
5.13 RBFneuralNet Class Reference

```
#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/old/[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 "Neuron.h"
#include "Con.cpp"
#include "Container.cpp"
#include "Neuron.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 Container< Con > ConContainer`
- `typedef Container< Neuron > NeuronContainer`
- `typedef boost::shared_ptr< ConContainer > ConContainerPtr`
- `typedef boost::shared_ptr< NeuronContainer > NeuronContainerPtr`
- `typedef std::vector< boost::shared_ptr< Con > >::iterator ConContainer_iterator`
- `typedef std::vector< boost::shared_ptr< Con > >::const_iterator ConContainer_const_iterator`
- `typedef std::vector< boost::shared_ptr< Neuron > >::iterator NeuronContainer_iterator`
- `typedef std::vector< boost::shared_ptr< Neuron > >::const_iterator NeuronContainer_const_iterator`

6.1.1 Define Documentation

6.1.1.1 `#define foreach BOOST_FOREACH`

Definition at line 49 of file AMORE.h.

6.1.1.2 `#define size_type unsigned int`

Definition at line 52 of file AMORE.h.

6.1.2 Typedef Documentation

6.1.2.1 `typedef Container<Con> ConContainer`

Definition at line 59 of file AMORE.h.

6.1.2.2 `typedef std::vector<boost::shared_ptr<Con> >::const_iterator ConContainer_const_iterator`

Definition at line 66 of file AMORE.h.

6.1.2.3 `typedef std::vector<boost::shared_ptr<Con> >::iterator ConContainer_iterator`

Definition at line 65 of file AMORE.h.

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

Definition at line 62 of file AMORE.h.

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

Definition at line 55 of file AMORE.h.

6.1.2.6 `typedef Container<Neuron> NeuronContainer`

Definition at line 60 of file AMORE.h.

6.1.2.7 `typedef std::vector<boost::shared_ptr<Neuron> >::const_iterator
NeuronContainer_const_iterator`

Definition at line 68 of file AMORE.h.

6.1.2.8 `typedef std::vector<boost::shared_ptr<Neuron> >::iterator
NeuronContainer_iterator`

Definition at line 67 of file AMORE.h.

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

Definition at line 63 of file AMORE.h.

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

Definition at line 56 of file AMORE.h.

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

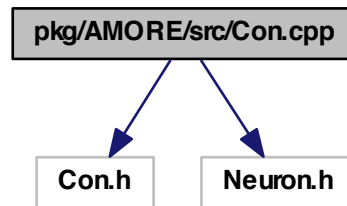
Definition at line 57 of file AMORE.h.

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

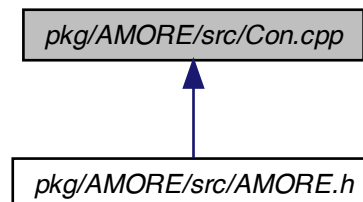
```
#include "Con.h"
```

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

Include dependency graph for Con.cpp:



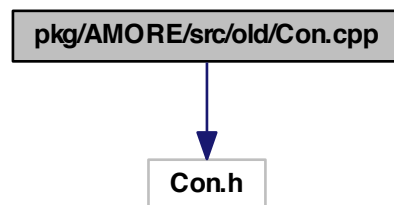
This graph shows which files directly or indirectly include this file:



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

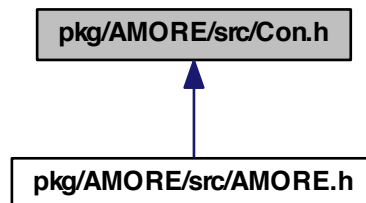
```
#include "Con.h"
```

Include dependency graph for Con.cpp:



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

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



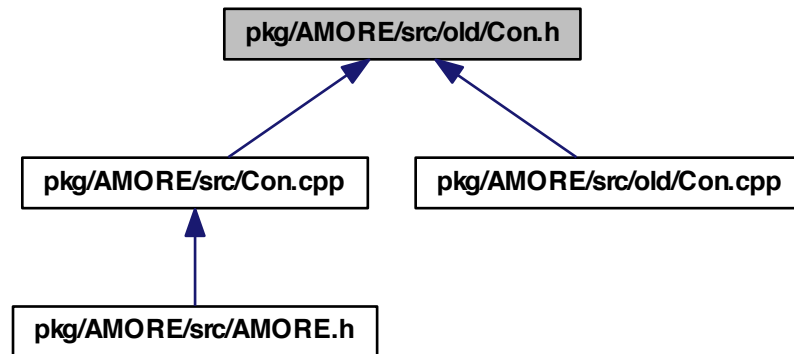
Classes

- class [Con](#)

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

6.5 pkg/AMORE/src/old/Con.h File Reference

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



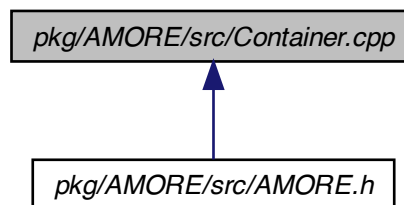
Classes

- class `Con`

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

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

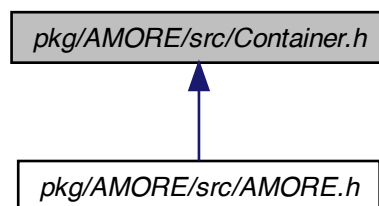
This graph shows which files directly or indirectly include this file:



6.7 pkg/AMORE/src/old/Container.cpp File Reference

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

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



Classes

- class `Container< T >`
class `Container` -

6.9 pkg/AMORE/src/old/Container.h File Reference

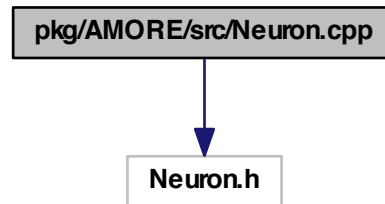
Classes

- class `Container< T >`
class `Container` -

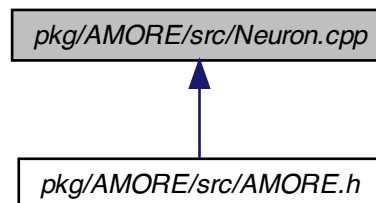
6.10 pkg/AMORE/src/Neuron.cpp File Reference

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

Include dependency graph for Neuron.cpp:



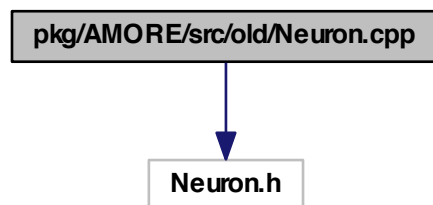
This graph shows which files directly or indirectly include this file:



6.11 pkg/AMORE/src/old/Neuron.cpp File Reference

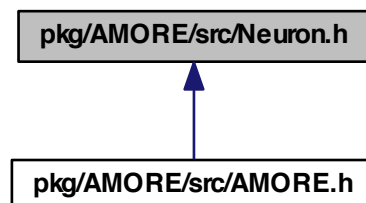
```
#include "Neuron.h"
```

Include dependency graph for Neuron.cpp:



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

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

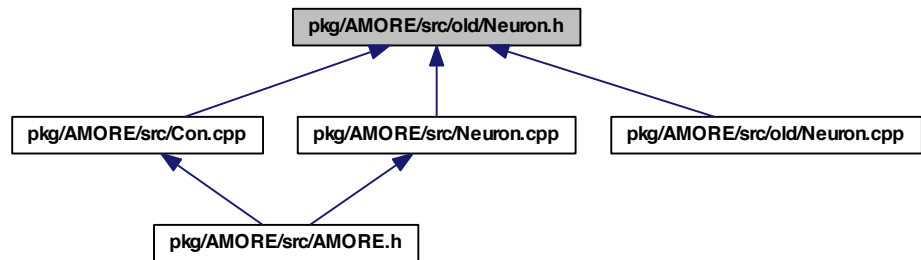


Classes

- class [Neuron](#)
class [Neuron](#) -

6.13 pkg/AMORE/src/old/Neuron.h File Reference

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



Classes

- class [Neuron](#)
class [Neuron](#) -

6.14 pkg/AMORE/src/old/ConContainer.cpp File Reference

Classes

- struct [CompareId](#)

6.15 pkg/AMORE/src/old/ConContainer.h File Reference

Classes

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

6.16 pkg/AMORE/src/old/MLPlayer.h File Reference

Classes

- class [MLPlayer](#)

6.17 pkg/AMORE/src/old/MLPlayerContainer.h File Reference

Classes

- class [MLPlayerContainer](#)

6.18 pkg/AMORE/src/old/MLPNeuralNet.h File Reference

Classes

- class [MLPNeuralNet](#)

6.19 pkg/AMORE/src/old/MLPNeuralNetFactory.cpp File Reference

Functions

- [MLPNeuralNet CreateMLPNeuralNet](#) (std::vector< int > numberOfNeuronsPerLayer)

6.19.1 Function Documentation

6.19.1.1 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();  
    }  
}
```

6.20 pkg/AMORE/src/old/MLPneuron.h File Reference

Classes

- class [MLPneuron](#)

6.21 pkg/AMORE/src/old/MLPneuronContainer.h File Reference

Classes

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

6.22 pkg/AMORE/src/old/NeuralNet.h File Reference

Classes

- class [NeuralNet](#)

6.23 pkg/AMORE/src/old/NeuronContainer.cpp File Reference

6.24 pkg/AMORE/src/old/NeuronContainer.h File Reference

Classes

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

6.25 pkg/AMORE/src/old/RBFneuralNet.h File Reference

Classes

- class [RBFneuralNet](#)

Index

- ~Con
 - Con, [12](#)
- ~Neuron
 - Neuron, [75](#)
- ~NeuronContainer
 - NeuronContainer, [83](#)
- AMORE.h
 - ConContainer, [90](#)
 - ConContainer_const_iterator, [90](#)
 - ConContainer_iterator, [90](#)
 - ConContainerPtr, [91](#)
 - ConPtr, [91](#)
 - foreach, [90](#)
 - NeuronContainer, [91](#)
 - NeuronContainer_const_iterator, [91](#)
 - NeuronContainer_iterator, [91](#)
 - NeuronContainerPtr, [91](#)
 - NeuronPtr, [91](#)
 - NeuronWeakPtr, [91](#)
 - size_type, [90](#)
- append
 - Container, [43](#)
- begin
 - Container, [45](#), [46](#)
- bias
 - MLPneuron, [66](#)
- buildAndAppend
 - Container, [46](#)
 - MLPneuronContainer, [69](#)
- clear
 - Container, [47](#)
- collection
 - Container, [54](#)
- CompareId, [9](#)
 - operator(), [9](#), [10](#)
- Con, [10](#)
 - ~Con, [12](#)
 - Con, [11](#), [12](#)
 - from, [19](#)
 - getFrom, [12](#), [13](#)
 - getId, [13](#), [14](#)
 - getWeight, [14](#)
 - setFrom, [15](#), [16](#)
 - setWeight, [16](#), [17](#)
 - show, [17](#)
 - validate, [18](#), [19](#)
 - weight, [19](#)
- con
 - Neuron, [79](#)
- ConContainer, [20](#)
 - AMORE.h, [90](#)
 - ConContainer, [24](#)
 - const_iterator, [23](#)
 - const_reference, [23](#)
 - erase, [24](#)
 - getId, [26](#)
 - iterator, [24](#)
 - numOfCons, [28](#)
 - select, [29](#)
 - setFrom, [31](#)
 - setWeight, [33](#), [34](#)
 - validate, [36](#)
 - value_type, [24](#)
- ConContainer_const_iterator
 - AMORE.h, [90](#)
- ConContainer_iterator
 - AMORE.h, [90](#)
- ConContainerPtr
 - AMORE.h, [91](#)
- ConPtr
 - AMORE.h, [91](#)
- const_iterator
 - ConContainer, [23](#)
 - Container, [41](#)
 - NeuronContainer, [82](#)
- const_iterators
 - Neuron, [75](#)
- const_reference
 - ConContainer, [23](#)

- Container, 41
- NeuronContainer, 82
- Container, 37
 - append, 43
 - begin, 45, 46
 - buildAndAppend, 46
 - clear, 47
 - collection, 54
 - const_iterator, 41
 - const_reference, 41
 - Container, 42, 43
 - empty, 47
 - end, 47, 48
 - erase, 48
 - insert, 48
 - iterator, 42
 - load, 48
 - push_back, 50, 51
 - reserve, 51
 - resize, 51
 - show, 51, 52
 - size, 53
 - store, 53
 - validate, 54
 - value_type, 42
- CreateMLPNeuralNet
 - MLPNeuralNetFactory.cpp, 99
- empty
 - Container, 47
- end
 - Container, 47, 48
- erase
 - ConContainer, 24
 - Container, 48
- foreach
 - AMORE.h, 90
- from
 - Con, 19
- getConId
 - Neuron, 75
 - NeuronContainer, 83
- getFrom
 - Con, 12, 13
 - NeuronContainer, 84
- getId
 - Con, 13, 14
 - ConContainer, 26
 - MLPNeuronContainer, 69
 - Neuron, 75, 76
 - NeuronContainer, 84
- getWeight
 - Con, 14
 - Neuron, 76
 - NeuronContainer, 84
- Id
 - Neuron, 79
- insert
 - Container, 48
- iterator
 - ConContainer, 24
 - Container, 42
 - NeuronContainer, 82
- iterators
 - Neuron, 76
- load
 - Container, 48
- MLPlayer, 55
- MLPlayerContainer, 58
- MLPNeuralNet, 61
 - nLayers, 63
- MLPNeuralNetFactory.cpp
 - CreateMLPNeuralNet, 99
- MLPNeuron, 63
 - bias, 66
- MLPNeuronContainer, 66
 - buildAndAppend, 69
 - getId, 69
- nCons
 - Neuron, 79
- NeuralNet, 69
 - train, 70
- Neuron, 70
 - ~Neuron, 75
 - con, 79
 - const_iterators, 75
 - getConId, 75
 - getId, 75, 76
 - getWeight, 76
 - Id, 79
 - iterators, 76
 - nCons, 79
 - Neuron, 74, 75
 - numOfCons, 76

- outputValue, [79](#)
- setFrom, [77](#)
- setId, [77](#)
- setWeight, [77](#)
- show, [77](#), [78](#)
- validate, [78](#)
- NeuronContainer, [79](#)
 - ~NeuronContainer, [83](#)
 - AMORE.h, [91](#)
 - const_iterator, [82](#)
 - const_reference, [82](#)
 - getConId, [83](#)
 - getFrom, [84](#)
 - getId, [84](#)
 - getWeight, [84](#)
 - iterator, [82](#)
 - NeuronContainer, [83](#)
 - numOfCons, [84](#)
 - numOfNeurons, [84](#)
 - setFrom, [85](#)
 - setId, [85](#)
 - setWeight, [85](#)
 - value_type, [83](#)
- NeuronContainer_const_iterator
 - AMORE.h, [91](#)
- NeuronContainer_iterator
 - AMORE.h, [91](#)
- NeuronContainerPtr
 - AMORE.h, [91](#)
- NeuronPtr
 - AMORE.h, [91](#)
- NeuronWeakPtr
 - AMORE.h, [91](#)
- nLayers
 - MLPneuralNet, [63](#)
- numOfCons
 - ConContainer, [28](#)
 - Neuron, [76](#)
 - NeuronContainer, [84](#)
- numOfNeurons
 - NeuronContainer, [84](#)
- operator()
 - CompareId, [9](#), [10](#)
- outputValue
 - Neuron, [79](#)
- pkg/AMORE/src/AMORE.h, [89](#)
- pkg/AMORE/src/Con.cpp, [91](#)
- pkg/AMORE/src/Con.h, [93](#)
- pkg/AMORE/src/Container.cpp, [94](#)
- pkg/AMORE/src/Container.h, [95](#)
- pkg/AMORE/src/Neuron.cpp, [95](#)
- pkg/AMORE/src/Neuron.h, [97](#)
- pkg/AMORE/src/old/Con.cpp, [92](#)
- pkg/AMORE/src/old/Con.h, [94](#)
- pkg/AMORE/src/old/ConContainer.cpp, [98](#)
- pkg/AMORE/src/old/ConContainer.h, [98](#)
- pkg/AMORE/src/old/Container.cpp, [95](#)
- pkg/AMORE/src/old/Container.h, [95](#)
- pkg/AMORE/src/old/MLPlayer.h, [98](#)
- pkg/AMORE/src/old/MLPlayerContainer.h, [99](#)
- pkg/AMORE/src/old/MLPneuralNet.h, [99](#)
- pkg/AMORE/src/old/MLPneuralNetFactory.cpp, [99](#)
- pkg/AMORE/src/old/MLPneuron.h, [100](#)
- pkg/AMORE/src/old/MLPneuronContainer.h, [100](#)
- pkg/AMORE/src/old/NeuralNet.h, [100](#)
- pkg/AMORE/src/old/Neuron.cpp, [96](#)
- pkg/AMORE/src/old/Neuron.h, [98](#)
- pkg/AMORE/src/old/NeuronContainer.cpp, [100](#)
- pkg/AMORE/src/old/NeuronContainer.h, [100](#)
- pkg/AMORE/src/old/RBFneuralNet.h, [100](#)
- push_back
 - Container, [50](#), [51](#)
- RBFneuralNet, [86](#)
- reserve
 - Container, [51](#)
- resize
 - Container, [51](#)
- select
 - ConContainer, [29](#)
- setFrom
 - Con, [15](#), [16](#)
 - ConContainer, [31](#)
 - Neuron, [77](#)
 - NeuronContainer, [85](#)
- setId
 - Neuron, [77](#)
 - NeuronContainer, [85](#)
- setWeight
 - Con, [16](#), [17](#)
 - ConContainer, [33](#), [34](#)
 - Neuron, [77](#)
 - NeuronContainer, [85](#)

- show
 - Con, [17](#)
 - Container, [51](#), [52](#)
 - Neuron, [77](#), [78](#)
- size
 - Container, [53](#)
- size_type
 - AMORE.h, [90](#)
- store
 - Container, [53](#)
- train
 - NeuralNet, [70](#)
- validate
 - Con, [18](#), [19](#)
 - ConContainer, [36](#)
 - Container, [54](#)
 - Neuron, [78](#)
- value_type
 - ConContainer, [24](#)
 - Container, [42](#)
 - NeuronContainer, [83](#)
- weight
 - Con, [19](#)