

## AMORE++

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

Generated by Doxygen 1.7.4

Sun Jun 5 2011 19:58:32



# Contents

<b>1</b>	<b>The AMORE++ package</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Motivation . . . . .	1
1.3	Road Map . . . . .	1
<b>2</b>	<b>Todo List</b>	<b>3</b>
<b>3</b>	<b>Class Index</b>	<b>5</b>
3.1	Class Hierarchy . . . . .	5
<b>4</b>	<b>Class Index</b>	<b>7</b>
4.1	Class List . . . . .	7
<b>5</b>	<b>File Index</b>	<b>9</b>
5.1	File List . . . . .	9
<b>6</b>	<b>Class Documentation</b>	<b>11</b>
6.1	CompareId Struct Reference . . . . .	11
6.1.1	Detailed Description . . . . .	11
6.1.2	Member Function Documentation . . . . .	11
6.1.2.1	operator() . . . . .	11
6.1.2.2	operator() . . . . .	11
6.1.2.3	operator() . . . . .	12
6.1.2.4	operator() . . . . .	12
6.2	Con Class Reference . . . . .	12
6.2.1	Detailed Description . . . . .	13
6.2.2	Constructor & Destructor Documentation . . . . .	13

6.2.2.1	Con . . . . .	13
6.2.2.2	Con . . . . .	13
6.2.2.3	Con . . . . .	14
6.2.2.4	~Con . . . . .	14
6.2.3	Member Function Documentation . . . . .	14
6.2.3.1	getFromId . . . . .	14
6.2.3.2	getFromNeuron . . . . .	15
6.2.3.3	getWeight . . . . .	16
6.2.3.4	setFromNeuron . . . . .	17
6.2.3.5	setWeight . . . . .	18
6.2.3.6	show . . . . .	18
6.2.3.7	validate . . . . .	19
6.2.4	Member Data Documentation . . . . .	20
6.2.4.1	from . . . . .	20
6.2.4.2	weight . . . . .	20
6.3	Neuron Class Reference . . . . .	21
6.3.1	Detailed Description . . . . .	21
6.3.2	Constructor & Destructor Documentation . . . . .	21
6.3.2.1	Neuron . . . . .	21
6.3.2.2	Neuron . . . . .	21
6.3.2.3	~Neuron . . . . .	22
6.3.3	Member Function Documentation . . . . .	22
6.3.3.1	getId . . . . .	22
6.3.3.2	setId . . . . .	22
6.3.4	Member Data Documentation . . . . .	22
6.3.4.1	Id . . . . .	22
6.3.4.2	outputValue . . . . .	22
6.4	vecAMORE< T > Class Template Reference . . . . .	23
6.4.1	Detailed Description . . . . .	26
6.4.2	Member Function Documentation . . . . .	26
6.4.2.1	append . . . . .	26
6.4.2.2	getLdata . . . . .	28
6.4.2.3	push_back . . . . .	29
6.4.2.4	reserve . . . . .	30

6.4.2.5	setLdata	30
6.4.2.6	show	30
6.4.2.7	size	31
6.4.2.8	validate	32
6.4.3	Member Data Documentation	32
6.4.3.1	ldata	32
6.5	vecCon Class Reference	33
6.5.1	Detailed Description	36
6.5.2	Member Function Documentation	36
6.5.2.1	buildAndAppend	36
6.5.2.2	erase	37
6.5.2.3	getFromId	39
6.5.2.4	getFromNeuron	40
6.5.2.5	getWeight	41
6.5.2.6	numOfCons	42
6.5.2.7	select	44
6.5.2.8	setFromNeuron	44
6.5.2.9	setWeight	46
6.6	vecMLPneuron Class Reference	47
6.6.1	Detailed Description	50
6.6.2	Member Function Documentation	50
6.6.2.1	buildAndAppend	50
6.7	vecNeuron Class Reference	50
6.7.1	Detailed Description	53
<b>7</b>	<b>File Documentation</b>	<b>55</b>
7.1	pkg/AMORE/src/AMORE.h File Reference	55
7.1.1	Typedef Documentation	56
7.1.1.1	ConSharedPtr	56
7.1.1.2	NeuronSharedPtr	56
7.1.1.3	NeuronWeakPtr	56
7.1.1.4	vecAMOREconSharedPtr	56
7.1.1.5	vecAMOREneuronSharedPtr	56
7.1.1.6	vecConSharedPtr	56

---

7.2	<a href="#">pkg/AMORE/src/Con.cpp File Reference</a>	57
7.3	<a href="#">pkg/AMORE/src/Con.h File Reference</a>	58
7.4	<a href="#">pkg/AMORE/src/Neuron.cpp File Reference</a>	58
7.5	<a href="#">pkg/AMORE/src/Neuron.h File Reference</a>	60
7.6	<a href="#">pkg/AMORE/src/vecAMORE.cpp File Reference</a>	60
7.7	<a href="#">pkg/AMORE/src/vecAMORE.h File Reference</a>	61
7.8	<a href="#">pkg/AMORE/src/vecCon.cpp File Reference</a>	61
7.9	<a href="#">pkg/AMORE/src/vecCon.h File Reference</a>	62
7.10	<a href="#">pkg/AMORE/src/vecMLPneuron.h File Reference</a>	62
7.11	<a href="#">pkg/AMORE/src/vecNeuron.h File Reference</a>	62

## 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

# Todo List

**Member** `Neuron::outputValue` restore vecCon<Con> listCon;



## Chapter 3

# Class Index

### 3.1 Class Hierarchy

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

CompareId . . . . .	11
Con . . . . .	12
Neuron . . . . .	21
vecAMORE< T > . . . . .	23
vecAMORE< Con > . . . . .	23
vecCon . . . . .	33
vecAMORE< Neuron > . . . . .	23
vecNeuron . . . . .	50
vecMLPneuron . . . . .	47



## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">CompareId</a> . . . . .	11
<a href="#">Con</a> (A class to handle the information needed to describe an input connection ) . .	12
<a href="#">Neuron</a> (A class to handle the information contained in a general <a href="#">Neuron</a> ) . .	21
<a href="#">vecAMORE&lt; T &gt;</a> . . . . .	23
<a href="#">vecCon</a> (A vector of connections ) . . . . .	33
<a href="#">vecMLPneuron</a> (A vector of connections ) . . . . .	47
<a href="#">vecNeuron</a> (A vector of neurons ) . . . . .	50



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

pkg/AMORE/src/ <a href="#">AMORE.h</a> . . . . .	55
pkg/AMORE/src/ <a href="#">Con.cpp</a> . . . . .	57
pkg/AMORE/src/ <a href="#">Con.h</a> . . . . .	58
pkg/AMORE/src/ <a href="#">Neuron.cpp</a> . . . . .	58
pkg/AMORE/src/ <a href="#">Neuron.h</a> . . . . .	60
pkg/AMORE/src/ <a href="#">vecAMORE.cpp</a> . . . . .	60
pkg/AMORE/src/ <a href="#">vecAMORE.h</a> . . . . .	61
pkg/AMORE/src/ <a href="#">vecCon.cpp</a> . . . . .	61
pkg/AMORE/src/ <a href="#">vecCon.h</a> . . . . .	62
pkg/AMORE/src/ <a href="#">vecMLPneuron.h</a> . . . . .	62
pkg/AMORE/src/ <a href="#">vecNeuron.h</a> . . . . .	62





## Chapter 6

# Class Documentation

### 6.1 CompareId Struct Reference

#### Public Member Functions

- bool [operator\(\)](#) (const [ConSharedPtr](#) a, const [ConSharedPtr](#) b)
- bool [operator\(\)](#) (const [ConSharedPtr](#) a, const int b)
- bool [operator\(\)](#) (const int a, const [ConSharedPtr](#) b)
- bool [operator\(\)](#) (const int a, const int b)

#### 6.1.1 Detailed Description

Definition at line 359 of file vecCon.cpp.

#### 6.1.2 Member Function Documentation

**6.1.2.1** bool CompareId::operator() ( const ConSharedPtr a, const ConSharedPtr b )  
[inline]

Definition at line 361 of file vecCon.cpp.

```
return a->getFromId() < b->getFromId();  
};
```

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

Definition at line 373 of file vecCon.cpp.

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

### 6.1.2.3 `bool CompareId::operator() ( const int a, const ConSharedPtr b )` `[inline]`

Definition at line 369 of file `vecCon.cpp`.

```

{
    return a < b->getFromId();
};

```

### 6.1.2.4 `bool CompareId::operator() ( const ConSharedPtr a, const int b )` `[inline]`

Definition at line 365 of file `vecCon.cpp`.

```

{
    return a->getFromId() < b ;
};

```

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

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

## 6.2 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 \(NeuronSharedPtr f\)](#)  
*Constructor.*
- [Con \(NeuronSharedPtr f, double w\)](#)  
*Constructor.*
- [~Con \(\)](#)  
*Default Destructor.*
- [NeuronSharedPtr getFromNeuron \(\)](#)  
*from field accessor.*
- void [setFromNeuron \(NeuronSharedPtr f\)](#)  
*from field accessor.*
- int [getFromId \(\)](#)  
*A getter of the Id of the [Neuron](#) pointed by the from field.*
- double [getWeight \(\)](#)  
*weight field accessor.*

- void `setWeight` (double w)  
*weight field accessor.*
- bool `show` ()  
*Pretty print of the `Con` information.*
- bool `validate` ()  
*Object validator.*

### Private Attributes

- `NeuronWeakPtr from`  
*A smart pointer to the `Neuron` used as input during simulation or training.*
- double `weight`  
*A double variable that contains the weight of the connection.*

#### 6.2.1 Detailed Description

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

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

Definition at line 16 of file `Con.h`.

#### 6.2.2 Constructor & Destructor Documentation

##### 6.2.2.1 `Con::Con ( )`

Default Constructor.

Definition at line 18 of file `Con.cpp`.

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

##### 6.2.2.2 `Con::Con ( NeuronSharedPtr f )`

Constructor.

Definition at line 36 of file `Con.cpp`.

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

### 6.2.2.3 Con::Con ( NeuronSharedPtr f, double w )

Constructor.

Definition at line 28 of file Con.cpp.

```
: from(f), weight(w) {};
```

### 6.2.2.4 Con::~Con ( )

Default Destructor.

Definition at line 41 of file Con.cpp.

```
{};
```

## 6.2.3 Member Function Documentation

### 6.2.3.1 int Con::getFromId ( )

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
NeuronSharedPtr ptShNeuron ( new Neuron(16) ); // Neuron
Id is set to 16
ConSharedPtr ptShCon( new Con(ptShNeuron) ); // from p
oints to ptShNeuron and weight is set to 0
// Test
int result = ptShCon->getFromId();

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

#### See also

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

Definition at line 117 of file Con.cpp.

References from.

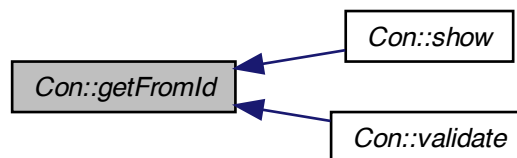
Referenced by `show()`, and `validate()`.

```

    {
        if (from.use_count() !=0 ){
            NeuronSharedPtr ptNeuron(from);
            return( ptNeuron->getId() );
        } else {
            return(NA_INTEGER);
        }
    }
}

```

Here is the caller graph for this function:



### 6.2.3.2 NeuronSharedPtr Con::getFromNeuron ( )

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
NeuronSharedPtr ptShNeuron ( new Neuron(1) ); // Neuron
Id is set 1
ConSharedPtr ptShCon( new Con(ptShNeuron) ); // from p
oints to ptShNeuron and weight is set to 0
// Test
ptShNeuron = ptShCon->getFromNeuron() ;
int result = ptShNeuron->getId();

// Now, result is equal to 1.

```

#### See also

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

Definition at line 65 of file Con.cpp.

References from.

```

    {
        return (from.lock());
    }

```

### 6.2.3.3 double Con::getWeight ( )

weight field accessor.

This method allows access to the value stored in the private field [weight](#)

#### Returns

The value of [weight](#) (double)

```

//=====
//Usage example:
//=====
// Data set up
        std::vector<double> result;
        NeuronSharedPtr ptShNeuron ( new Neuron(16) );           /
/ Neuron Id is set to 16
        ConSharedPtr ptShCon( new Con(ptShNeuron, 12.4) ); // fr
om points 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 151 of file Con.cpp.

References [weight](#).

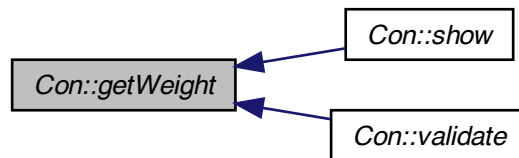
Referenced by [show\(\)](#), and [validate\(\)](#).

```

    {
        return (weight);
    }

```

Here is the caller graph for this function:



#### 6.2.3.4 void Con::setFromNeuron ( NeuronSharedPtr f )

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 <a href="#">from</a> field.
----------	---

```

//=====
//Usage example:
//=====
// Data set up
NeuronSharedPtr ptShNeuron ( new Neuron(1) ); // Neuron
Id is set to 1
ConSharedPtr ptShCon( new Con() );
ptShCon->setFromNeuron( ptShNeuron );

// Test
ptShNeuron = ptShCon->getFromNeuron() ;
int result = ptShNeuron->getId();

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

##### See also

[getFromNeuron](#) and [getFromId](#) contain usage examples. For further examples see the unit test files, e.g., `runit.Cpp.Con.R`

Definition at line 92 of file `Con.cpp`.

References from.

```

{
    from=f;
}
  
```

### 6.2.3.5 void Con::setWeight ( double w )

weight field accessor.

This method sets the value of the [weight](#) field.

#### Parameters

<b>w</b>	The new value (double) to be set in the <a href="#">weight</a> field.
----------	---

```
//=====
//Usage example:
//=====
// Data set up
                                std::vector<double> result;
                                NeuronSharedPtr ptShNeuron ( new Neuron(16) );           /
/ Neuron Id is set to 16
                                ConSharedPtr ptShCon( new Con(ptShNeuron, 12.4) ); // fr
om points 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 180 of file `Con.cpp`.

References `weight`.

```
                                {
                                weight = w;
                                }
```

### 6.2.3.6 bool Con::show ( )

Pretty print of the [Con](#) information.

This method outputs in the R terminal the contents of the [Con](#) fields.

#### Returns

true in case everything works without throwing an exception

#### See also

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

Definition at line 192 of file `Con.cpp`.

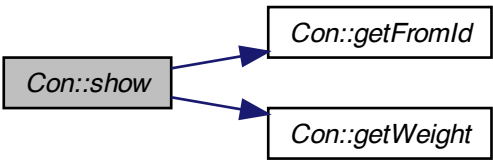
References `getFromId()`, and `getWeight()`.



```

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

Here is the call graph for this function:



6.2.3.7 bool Con::validate ( )

Object validator.

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

Returns

true in case the checks are Ok.

Exceptions

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

Definition at line 211 of file Con.cpp.

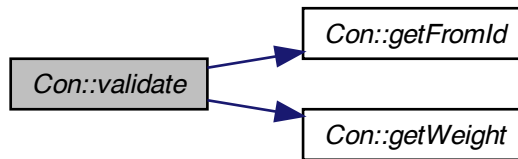
References [getFromId\(\)](#), and [getWeight\(\)](#).

```

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

```
    return(true);  
    END_RCPP  
};
```

Here is the call graph for this function:



## 6.2.4 Member Data Documentation

### 6.2.4.1 `NeuronWeakPtr Con::from` [private]

A smart pointer to the [Neuron](#) used as input during simulation or training.

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

Definition at line 21 of file `Con.h`.

Referenced by `getFromId()`, `getFromNeuron()`, and `setFromNeuron()`.

### 6.2.4.2 `double Con::weight` [private]

A double variable that contains the weight of the connection.

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

Definition at line 26 of file `Con.h`.

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

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

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

## 6.3 Neuron Class Reference

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

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

### Public Member Functions

- [Neuron](#) ()
- [Neuron](#) (int [Id](#))
- [~Neuron](#) ()
- int [getId](#) ()
- void [setId](#) (int [id](#))

### Private Attributes

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

#### 6.3.1 Detailed Description

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

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

Definition at line 16 of file Neuron.h.

#### 6.3.2 Constructor & Destructor Documentation

##### 6.3.2.1 [Neuron::Neuron](#) ( )

Definition at line 12 of file Neuron.cpp.

```
{};
```

##### 6.3.2.2 [Neuron::Neuron](#) ( int [Id](#) )

Definition at line 13 of file Neuron.cpp.

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

### 6.3.2.3 Neuron::~~Neuron ( )

Definition at line 14 of file Neuron.cpp.

```
{};
```

## 6.3.3 Member Function Documentation

### 6.3.3.1 int Neuron::getId ( )

Definition at line 17 of file Neuron.cpp.

References [Id](#).

```
    {
        return Id;
    }
```

### 6.3.3.2 void Neuron::setId ( int id )

Definition at line 21 of file Neuron.cpp.

References [Id](#).

```
    {
        Id=id;
    }
```

## 6.3.4 Member Data Documentation

### 6.3.4.1 int Neuron::Id [private]

An integer variable with the [Neuron](#) Id.

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

Definition at line 21 of file Neuron.h.

Referenced by [getId\(\)](#), and [setId\(\)](#).

### 6.3.4.2 double Neuron::outputValue [private]

A vector of input connections.

### Todo

```
restore vecCon<Con> listCon;
```

Definition at line 30 of file Neuron.h.

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

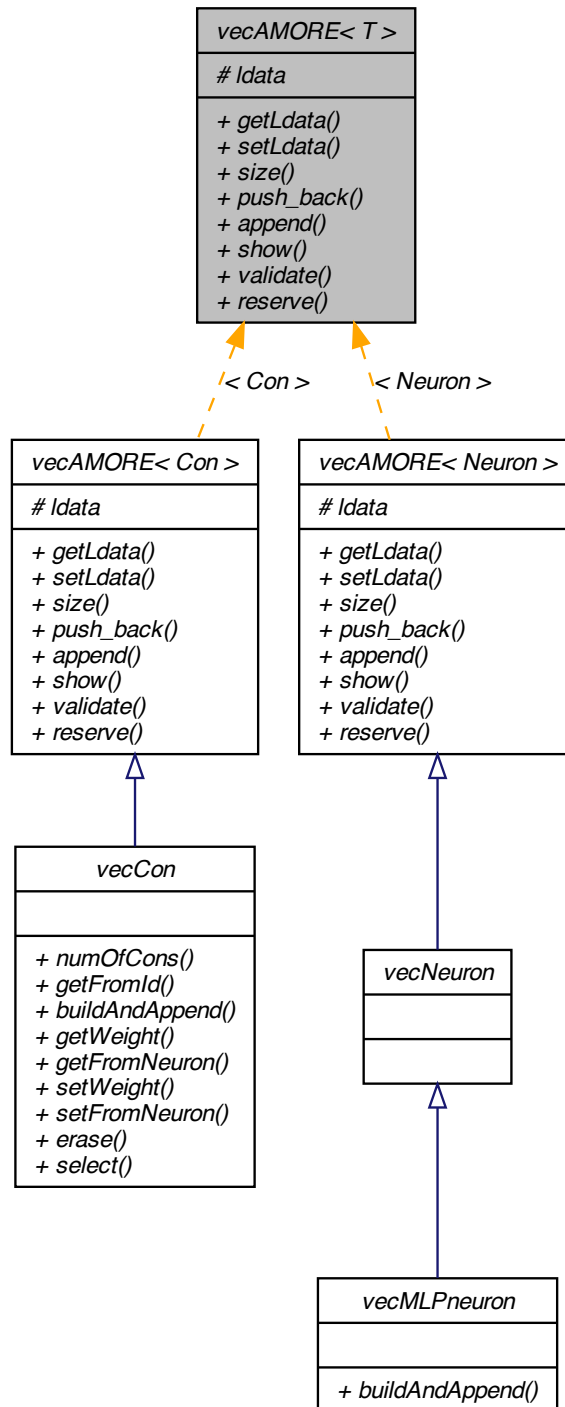
- pkg/AMORE/src/[Neuron.h](#)

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

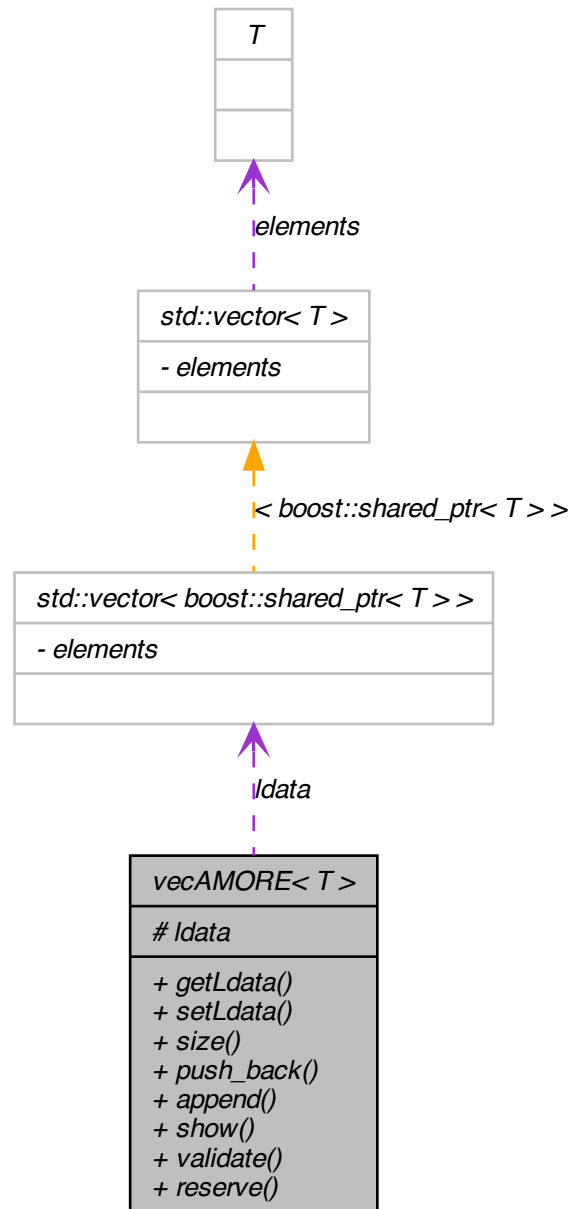
## 6.4 vecAMORE< T > Class Template Reference

```
#include <vecAMORE.h>
```

Inheritance diagram for vecAMORE< T >:



Collaboration diagram for vecAMORE< T >:



## Public Member Functions

- `std::vector< boost::shared_ptr< T > > getLdata ()`  
*ldata field accessor function*
- `void setLdata (typename std::vector< boost::shared_ptr< T > >)`  
*ldata field accessor function*
- `int size ()`  
*Returns the size or length of the vector.*
- `void push\_back (boost::shared_ptr< T > element)`  
*Append a shared\_ptr at the end of ldata.*
- `void append (vecAMORE< T > v)`  
*Appends a vecAMORE<T> object.*
- `bool show ()`  
*Pretty print of the vecAMORE<T>*
- `bool validate ()`  
*Object validator.*
- `void reserve (int n)`

## Protected Attributes

- `std::vector< boost::shared_ptr< T > > ldata`

### 6.4.1 Detailed Description

`template<typename T>class vecAMORE< T >`

Definition at line 12 of file vecAMORE.h.

### 6.4.2 Member Function Documentation

#### 6.4.2.1 `template<typename T> void vecAMORE< T >::append ( vecAMORE< T > v )`

Appends a vecAMORE<T> object.

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

#### Parameters

<code>v</code>	The vecAMORE<T> object to be added to the current one
----------------	---

#### See also

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

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



```

//=====
// Data set up

std::vector<int> result;
std::vector<ConSharedPtr> vcA, vcB;
vecAMOREneuronSharedPtr ptShvNeuron( new

vecAMORE<Neuron>() );
vecAMOREconSharedPtr ptShvConA( new
vecAMOREconSharedPtr ptShvConB( new

ConSharedPtr ptC;
NeuronSharedPtr 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] ) );
ptShvNeuron->push_back(ptN);
}
for (int i=0; i<=2 ; i++) {
/
/ A vector with three connections
ptC.reset( new Con( ptShvNeuron->getLdata
().at(i), weights[i] ) );
ptShvConA->push_back(ptC);
}
for (int i=3; i<=5 ; i++) {
/
/ Another vector with three connections
ptC.reset( new Con( ptShvNeuron->getLdata
().at(i), weights[i] ) );
ptShvConB->push_back(ptC);
}

// Test
ptShvConA->append(*ptShvConB);
ptShvConA->validate();
ptShvConA->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**

[vecAMORE::setLdata](#), [vecAMORE::push\\_back](#) and the unit test files, e.g., `runit.Cpp.vecAMORE.R`, for usage examples.

Definition at line 166 of file `vecAMORE.cpp`.

References `vecAMORE< T >::ldata`, and `vecAMORE< T >::size()`.

```

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

```

Here is the call graph for this function:



#### 6.4.2.2 `template<typename T> std::vector< boost::shared_ptr< T> > vecAMORE< T>::getLdata ( )`

ldata field accessor function

This method allows access to the data stored in the `ldata` field.

#### Returns

The ldata vector.

```

//=====
//Usage example:
//=====
// Data set up
std::vector<int> result;
std::vector<ConSharedPtr> vcA, vcB;
vecAMOREneuronSharedPtr ptShvNeuron( new
vecAMORE<Neuron>( ) );
vecAMOREconSharedPtr ptShvCon( new
vecAMORE<Con>( ) );

ConSharedPtr ptC;
NeuronSharedPtr 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] ) );
ptShvNeuron->push_back(ptN);
}
for (int i=0; i<=2 ; i++) {
/ and a vector with three connections
ptC.reset( new Con( ptShvNeuron->getLdata
().at(i), weights[i] ) );
vcA.push_back(ptC);
}

// Test
ptShvCon->setLdata(vcA);
vcB = ptShvCon->getLdata();
for (int i=0; i<=2 ; i++) {
/ get Ids. vecAMORE does not have getFromId defined
result.push_back( vcB.at(i)->getFromId())
;

```

```

    }

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

```

**See also**

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

Definition at line 211 of file `vecAMORE.cpp`.

```

    {
        return ldata;
    };

```

#### 6.4.2.3 `template<typename T> void vecAMORE< T >::push_back ( boost::shared_ptr< T > TsharedPtr )`

Append a `shared_ptr` at the end of `ldata`.

Implements `push_back` for the [vecAMORE](#) class

**Parameters**

<i>TsharedPtr</i>	A <code>shared_ptr</code> pointer to be inserted at the end of <code>ldata</code>
-------------------	---

```

//=====
//Usage example:
//=====
// Data set up
    Neuron N1, N2, N3;
    vecAMORE<Con> MyvecCon;
    std::vector<ConSharedPtr> vc;
    std::vector<int> result;
    N1.setId(10);
    N2.setId(20);
    N3.setId(30);

// Test
    ConSharedPtr ptCon( new Con(&N1, 1.13) );           // Create
new Con and initialize ptCon
    MyvecCon.push_back(ptCon);                          /
/ push_back
    ptCon.reset( new Con(&N2, 2.22) );                 // create
new Con and assign to ptCon
    MyvecCon.push_back(ptCon);                          /
/ push_back
    ptCon.reset( new Con(&N3, 3.33) );                 // create
new Con and assign to ptCon
    MyvecCon.push_back(ptCon);                          /
/ push_back

    vc = MyvecCon.getLdata();

    result.push_back(vc.at(0)->getFromId());
    result.push_back(vc.at(1)->getFromId());
    result.push_back(vc.at(2)->getFromId());

// 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.vecAMORE.R`, for usage examples.

Definition at line 44 of file `vecAMORE.cpp`.

```

{
    ldata.push_back(TsharedPtr);
};

```

**6.4.2.4** `template<typename T> void vecAMORE< T >::reserve ( int n )`

Definition at line 239 of file `vecAMORE.cpp`.

```

{
    ldata.reserve(n) ;
};

```

**6.4.2.5** `template<typename T> void vecAMORE< T >::setLdata ( typename std::vector< boost::shared_ptr< T > > v )`

`ldata` field accessor function

This method sets the value of the data stored in the `ldata` field.

**Parameters**

<code>v</code>	The vector of smart pointers to be stored in the <code>ldata</code> field
----------------	---

**See also**

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

Definition at line 223 of file `vecAMORE.cpp`.

```

{
    ldata=v;
};

```

**6.4.2.6** `template<typename T> bool vecAMORE< T >::show ( )`

Pretty print of the `vecAMORE<T>`

This method outputs in the R terminal the contents of `vecAMORE::ldata`.

**Returns**

true in case everything works without throwing an exception

\*

```
//=====
//Usage example:
//=====
// Data set up
vecAMOREneuronSharedPtr ptShvNeuron( new
vecAMORE<Neuron>() );
vecAMOREconSharedPtr    ptShvCon( new vecAMORE<Con>() );
ConSharedPtr            ptC;
NeuronSharedPtr          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] ) );
        ptShvNeuron->push_back(ptN);
    }

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

// Test
ptShvCon->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.vecAMORE.R`, for usage examples.

Definition at line 93 of file `vecAMORE.cpp`.

```
{
// This is equivalent to:
// for( auto x : ldata) { x.show(); }
// Waiting for C++0x
for(typename std::vector< boost::shared_ptr<T> >::iterator itr = ldata.b
egin();  itr != ldata.end();  itr++) { (*itr)->show(); }
return true;
};
```

#### 6.4.2.7 `template<typename T> int vecAMORE< T >::size ( )`

Returns the size or length of the vector.

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

Definition at line 234 of file vecAMORE.cpp.

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

```

    {
        return ldata.size() ;
    };

```

Here is the caller graph for this function:



#### 6.4.2.8 template<typename T> bool vecAMORE< T >::validate ( )

Object validator.

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

#### See also

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

Definition at line 108 of file vecAMORE.cpp.

```

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

```

### 6.4.3 Member Data Documentation

#### 6.4.3.1 template<typename T> std::vector<boost::shared\_ptr<T> > vecAMORE< T >::ldata [protected]

Definition at line 14 of file vecAMORE.h.

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

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

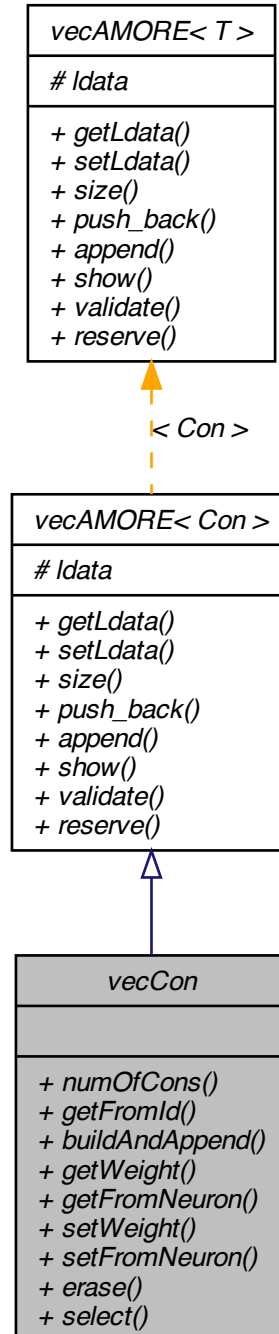
- pkg/AMORE/src/[vecAMORE.h](#)
- pkg/AMORE/src/[vecAMORE.cpp](#)

## 6.5 vecCon Class Reference

A vector of connections.

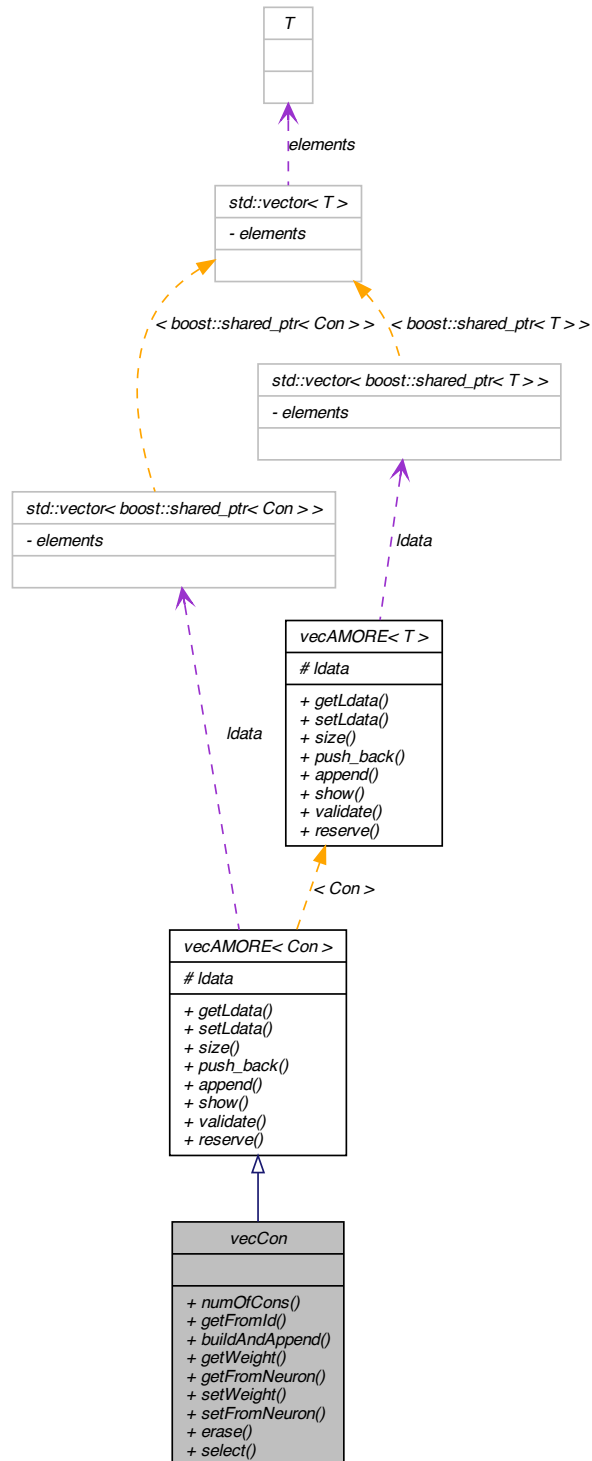
```
#include <vecCon.h>
```

Inheritance diagram for vecCon:





Collaboration diagram for vecCon:



## Public Member Functions

- int [numOfCons](#) ()  
*Size of the [vecCon](#) object.*
- std::vector< int > [getFromId](#) ()  
*Getter of the Id values of the vector of Cons.*
- bool [buildAndAppend](#) (std::vector< [NeuronSharedPtr](#) > FROM, std::vector< double > WEIGHT)  
*Builds [Con](#) objects and appends them to ldata.*
- std::vector< double > [getWeight](#) ()  
*Getter of the weight field of the [Con](#) objects related to [vecCon](#).*
- std::vector< [NeuronSharedPtr](#) > [getFromNeuron](#) ()  
*Getter of the from field of the [Con](#) objects related to [vecCon](#).*
- bool [setWeight](#) (std::vector< double > value)  
*Setter of the weight field of the [Con](#) objects related to [vecCon](#).*
- bool [setFromNeuron](#) (std::vector< [NeuronSharedPtr](#) > FROM)  
*Setter of the from fields of the [Con](#) objects related to [vecCon](#).*
- void [erase](#) (std::vector< int > FROM)  
*Erase the specified elements from the [vecCom](#) object.*
- [vecConSharedPtr](#) [select](#) (std::vector< int > FROM)

### 6.5.1 Detailed Description

A vector of connections.

The [vecCon](#) class provides a simple class for a vector of connections. It's named after the R equivalent Reference Class.

Definition at line 17 of file [vecCon.h](#).

### 6.5.2 Member Function Documentation

#### 6.5.2.1 bool [vecCon::buildAndAppend](#) ( std::vector< [NeuronSharedPtr](#) > FROM, std::vector< double > WEIGHT )

Builds [Con](#) objects and appends them to ldata.

This function provides a convenient way of populating a [vecCon](#) object by building and appending [Con](#) objects to ldata.

#### Parameters

<i>FROM</i>	A vector of smart pointers to the neurons to be used in the <a href="#">Con::from</a> fields
<i>WEIGHT</i>	A vector of values to be set in the <a href="#">Con::weight</a> fields

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

```

// Data set up
std::vector<int> result;
vecCon MyvecCon;
std::vector<NeuronSharedPtr> vNeuron;
std::vector<double> vWeight;

// Test
NeuronSharedPtr ptNeuron( new Neuron(11) );
vNeuron.push_back(ptNeuron);
ptNeuron.reset( new Neuron(22) );
vNeuron.push_back(ptNeuron);
ptNeuron.reset( new Neuron(33) );
vNeuron.push_back(ptNeuron);

vWeight.push_back(12.3);
vWeight.push_back(1.2);
vWeight.push_back(2.1);

MyvecCon.buildAndAppend(vNeuron, vWeight);

result=MyvecCon.getFromId();

// Now result is a vector that contains the values 11, 22 and 32.

```

**See also**

[append](#) and the unit test files, e.g. `runit.Cpp.vecCon.R`, for further examples.

Definition at line 130 of file `vecCon.cpp`.

References `vecAMORE< Con >::ldata`.

```

{
BEGIN_RCPP
if (FROM.empty()) { throw std::range_error("[vecCon::append]: Error, FROM
is empty"); }
if (FROM.size() != WEIGHT.size()) { throw std::range_error("[vecCon::bui
ldAndAppend]: Error, FROM.size() != WEIGHT.size()"); }
ldata.reserve(ldata.size() + FROM.size());
ConSharedPtr ptCon;
std::vector<double>::iterator itrWEIGHT = WEIGHT.begin();
for( std::vector<NeuronSharedPtr>::iterator itrFROM=FROM.begin(); itrFR
OM != FROM.end(); itrFROM++ , itrWEIGHT++) {
    ptCon.reset( new Con( *itrFROM, *itrWEIGHT) );
    ldata.push_back(ptCon);
}
return true;
END_RCPP
}

```

**6.5.2.2 void vecCon::erase ( std::vector< int > vFrom )**

Erase the specified elements from the `vecCom` object.

Provides a convenient way of removing some [Con](#) objects from the `ldata` field of the [vecCon](#) object.

## Parameters

<i>vFrom</i>	An <code>std::vector&lt;int&gt;</code> with the ids of the connections to remove.
--------------	---

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

// Data set up
std::vector<int> result;
std::vector<NeuronSharedPtr> vNeuron;
vecConSharedPtr ptShvCon( new vecCon() );
vecConSharedPtr vErased;
ConSharedPtr ptC;
NeuronSharedPtr ptN;
int ids[]= {11, 10, 9, 3, 4, 5, 6, 7, 8, 2, 1};
std::vector<double> vWeight;
vWeight.push_back(11.32);
vWeight.push_back(1.26);
vWeight.push_back(2.14);
vWeight.push_back(3.16);
vWeight.push_back(4.14);
vWeight.push_back(5.19);
vWeight.push_back(6.18);
vWeight.push_back(7.16);
vWeight.push_back(8.14);
vWeight.push_back(9.12);
vWeight.push_back(10.31);

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

// Test

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

ptShvCon->erase(toRemove);
ptShvCon->show();
result=ptShvCon->getFromId();

// 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.vecCon.R`, for further examples.

Definition at line 445 of file vecCon.cpp.

References `vecAMORE< Con >::ldata`.

```

    {
        std::vector<ConSharedPtr>::iterator itr;
        sort (ldata.begin(), ldata.end(), CompareId());
        sort (vFrom.begin(), vFrom.end());
        itr=set_difference (ldata.begin(), ldata.end(), vFrom.begin(), vFrom.end(
), ldata.begin(), CompareId());
        ldata.resize(itr-ldata.begin());
    }

```

### 6.5.2.3 `std::vector< int > vecCon::getFromId ( )`

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;
    vecCon MyvecCon;
    std::vector<int> result;

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

    ConSharedPtr ptCon( new Con(&N1, 1.13) );           // Create
new Con and initialize ptCon
    MyvecCon.push_back(ptCon);                           /
/ push_back
    ptCon.reset( new Con(&N2, 2.22) );                 // create
new Con and assign to ptCon
    MyvecCon.push_back(ptCon);                           /
/ push_back
    ptCon.reset( new Con(&N3, 3.33) );                 // create
new Con and assign to ptCon
    MyvecCon.push_back(ptCon);                           /
/ push_back

    // Test
    MyvecCon.show() ;
    MyvecCon.validate();
    result=MyvecCon.getFromId();

    // 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.vecCon.R`, for further examples.

Definition at line 83 of file vecCon.cpp.

References `vecAMORE< Con >::ldata`, and `numOfCons()`.

```

    {
        std::vector<int> result;
        result.reserve(numOfCons());
        for(std::vector<ConSharedPtr>::iterator itr = ldata.begin();   itr !=
ldata.end();   itr++)    { result.push_back((*itr)->getFromId()); }
        return result;
    }

```

Here is the call graph for this function:



#### 6.5.2.4 `std::vector< NeuronSharedPtr > vecCon::getFromNeuron ( )`

Getter of the from field of the [Con](#) objects related to [vecCon](#).

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 [vecCon](#) object.

#### Returns

An `std::vector<NeuronSharedPtr>` with the pointer to the incoming neurons.

```

//=====
//Usage example:
//=====
// Data set up
std::vector<double> result;
int ids[] = {1, 2, 3};
double weights[] = {12.3, 1.2, 2.1 };
vecCon MyvecCon;
std::vector<NeuronSharedPtr> vNeuron;
std::vector<double> vWeight;
NeuronSharedPtr ptNeuron;

    for (int i=0; i<=2; i++) {
        ptNeuron.reset( new Neuron(ids[i]) );
        vNeuron.push_back(ptNeuron);
        vWeight.push_back(weights[i]);
    }
    MyvecCon.buildAndAppend(vNeuron, vWeight);
// Test

```

```

vNeuron=MyvecCon.getFromNeuron();
for (int i=0; i<=2; i++) {
    result.push_back(vNeuron.at(i)->getId());
}

// Now result is a vector that contains the values 1, 2 and 3 .

```

**See also**

[getFromId](#) and the unit test files, e.g. `runit.Cpp.vecCon.R`, for further examples.

Definition at line 288 of file `vecCon.cpp`.

References `vecAMORE< Con >::ldata`, and `numOfCons()`.

```

{
    std::vector<NeuronSharedPtr> result;
    result.reserve(numOfCons());
    for(std::vector<ConSharedPtr>::iterator itr = ldata.begin(); itr !=
ldata.end(); itr++) {
        result.push_back((*itr)->getFromNeuron());
    }
    return result;
}

```

Here is the call graph for this function:



#### 6.5.2.5 `std::vector< double > vecCon::getWeight ( )`

Getter of the weight field of the [Con](#) objects related to [vecCon](#).

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 [vecCon](#) object.

**Returns**

A numeric (double) vector with the weights

```

//=====
//Usage example:
//=====
// Data set up
std::vector<double> result;

```

```

vecCon MyvecCon;
std::vector<NeuronSharedPtr> vNeuron;
std::vector<double> vWeight;

// Test
NeuronSharedPtr ptNeuron( new Neuron(11) );
vNeuron.push_back(ptNeuron);
ptNeuron.reset( new Neuron(22) );
vNeuron.push_back(ptNeuron);
ptNeuron.reset( new Neuron(33) );
vNeuron.push_back(ptNeuron);

vWeight.push_back(12.3);
vWeight.push_back(1.2);
vWeight.push_back(2.1);

MyvecCon.buildAndAppend(vNeuron, vWeight);

result=MyvecCon.getWeight();

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

```

### See also

[getFromId](#) and the unit test files, e.g. `runit.Cpp.vecCon.R`, for further examples.

Definition at line 185 of file `vecCon.cpp`.

References `vecAMORE< Con >::ldata`, and `numOfCons()`.

```

{
    std::vector<double> result;
    result.reserve(numOfCons());
    for(std::vector<ConSharedPtr>::iterator itr = ldata.begin();   itr !=
ldata.end();   itr++)   { result.push_back((*itr)->getWeight()); }
    return result;
}

```

Here is the call graph for this function:



#### 6.5.2.6 int vecCon::numOfCons ( )

Size of the [vecCon](#) object.



This function returns the size of the [vecCon](#) object, that is to say, the number of [Con](#) objects it contains.

### Returns

The size of the vector

```
//=====
//Usage example:
//=====
// Data set up

vecAMORE<Neuron>() );

std::vector<int> result;
std::vector<ConSharedPtr> vcA, vcB;
vecAMOREneuronSharedPtr ptShvNeuron( new

vecConSharedPtr ptShvCon( new vecCon() );
ConSharedPtr      ptC;
NeuronSharedPtr 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] ) );
    ptShvNeuron->push_back( ptN );
}

// Test
for (int i=0; i<=2 ; i++) {
/
/ and a vector with three connections
    result.push_back( ptShvCon->numOfCons() );
/
/ Append numOfCons to result, create new Con and push_back into MyvecCon
    ptC.reset( new Con( ptShvNeuron->getLdata
().at(i), weights[i] ) );
    ptShvCon->push_back( ptC );
}

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

### See also

[vecAMORE::size](#) (alias)

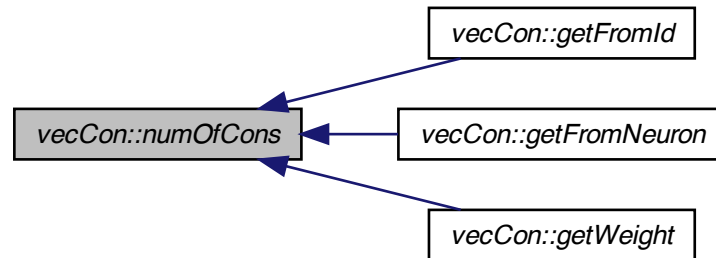
Definition at line 42 of file `vecCon.cpp`.

References `vecAMORE< Con >::ldata`.

Referenced by `getFromId()`, `getFromNeuron()`, and `getWeight()`.

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

Here is the caller graph for this function:



#### 6.5.2.7 `vecConSharedPtr vecCon::select ( std::vector< int > FROM )`

Definition at line 454 of file `vecCon.cpp`.

```

{

    /*
    vecConSharedPtr result(new vecCon );
    result->reserve(ldata.size());
    sort (ldata,ldata.size());
    sort (vFrom,vFrom.size());
    set_intersection(ldata, ldata.size(), vFrom, vFrom.size(), result.begin()
, compareId());
    return result;
    */
}

```

#### 6.5.2.8 `bool vecCon::setFromNeuron ( std::vector< NeuronSharedPtr > vFrom )`

Setter of the from fields of the [Con](#) objects related to [vecCon](#).

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 [vecCon](#) object.

##### Parameters

<i>vFrom</i>	An <code>std::vector&lt;NeuronSharedPtr&gt;</code> with the pointers to be set in the from fields of the <a href="#">vecCon</a> object.
--------------	---

**Returns**

true if not exception is thrown

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

// Data set up
std::vector<int> result;
vecAMOREneuronSharedPtr ptShvNeuron( new vecAMORE<Neuron>() );
vecConSharedPtr ptShvCon( new vecCon() );
ConSharedPtr ptC;
NeuronSharedPtr 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] ) );
    ptShvNeuron->push_back(ptN);
}
for (int i=0; i<=2 ; i++) { // and a
vector with three connections
    ptC.reset( new Con() );
    ptShvCon->push_back(ptC);
}

// Test
ptShvCon->setFromNeuron(ptShvNeuron->getLdata() );
ptShvCon->show();
result=ptShvCon->getFromId();

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

**See also**

[getFromNeuron](#) and the unit test files, e.g. `runit.Cpp.vecCon.R`, for further examples.

Definition at line 343 of file `vecCon.cpp`.

References `vecAMORE< Con >::ldata`.

```

{
    BEGIN_RCPP
        if (vFrom.empty()) { throw std::range_error("[ C++ vecCon::setFromNeuron]: Error, w is empty"); }
        if (vFrom.size() != ldata.size() ) { throw std::range_error("[C++ vecCon::setFromNeuron]: Error, w.size() != ldata.size()"); }
        std::vector<NeuronSharedPtr>::iterator itrFrom = vFrom.begin();
        for(std::vector<ConSharedPtr>::iterator itr = ldata.begin(); itr != ldata.end(); itr++, itrFrom++) {
            (*itr)->setFromNeuron( *itrFrom );
        }

        return true;
    END_RCPP
}

```

### 6.5.2.9 bool vecCon::setWeight ( std::vector< double > vWeight )

Setter of the weight field of the [Con](#) objects related to [vecCon](#).

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 [vecCon](#) object.

#### Parameters

<b>w</b>	A numeric (double) vector with the weights to be set in the <a href="#">Con</a> objects contained in the <a href="#">vecCon</a> 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 };
vecCon MyvecCon;
std::vector<NeuronSharedPtr> vNeuron;
std::vector<double> vWeight;
NeuronSharedPtr ptNeuron;

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

for (int i=0; i<=2; i++) {
    vWeight.at(i)=weights[i];
}

// Test
MyvecCon.setWeight(vWeight); // weight
s are set to 12.3, 1.2 and 2.1
result=MyvecCon.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.vecCon.R`, for further examples.

Definition at line 235 of file `vecCon.cpp`.

References `vecAMORE< Con >::ldata`.

```
{
BEGIN_RCPP
if (vWeight.empty()) { throw std::range_error("[ C++ vecCon::setWeight]:
Error, vWeight is empty"); }
```

```
        if (vWeight.size() != ldata.size() ) { throw std::range_error("[C++ vecCon::setWeight]: Error, vWeight.size() != ldata.size()"); }
        std::vector<double>::iterator itrWeight = vWeight.begin();
        for(std::vector<ConSharedPtr>::iterator itr = ldata.begin();   itr !=
        ldata.end();   itr++, itrWeight++)      {
            (*itr)->setWeight( *itrWeight );
        }
        return true;
    END_RCPP
}
```

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

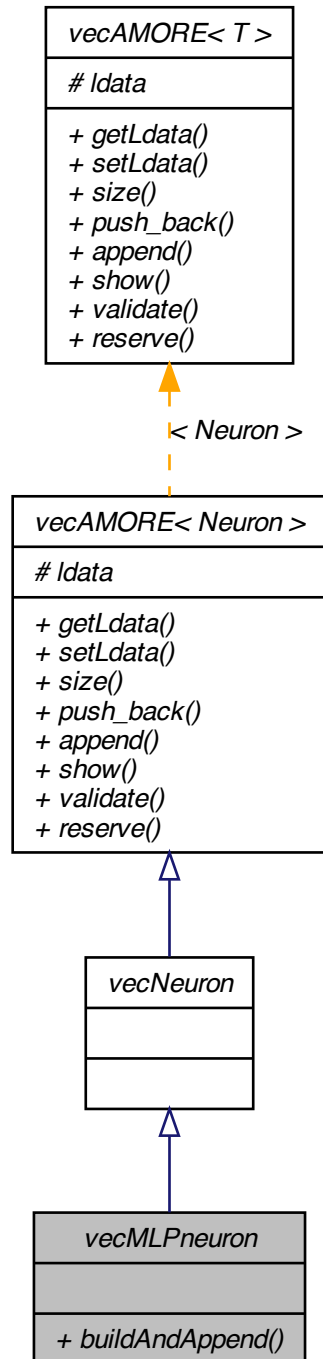
- pkg/AMORE/src/[vecCon.h](#)
- pkg/AMORE/src/[vecCon.cpp](#)

## 6.6 vecMLPneuron Class Reference

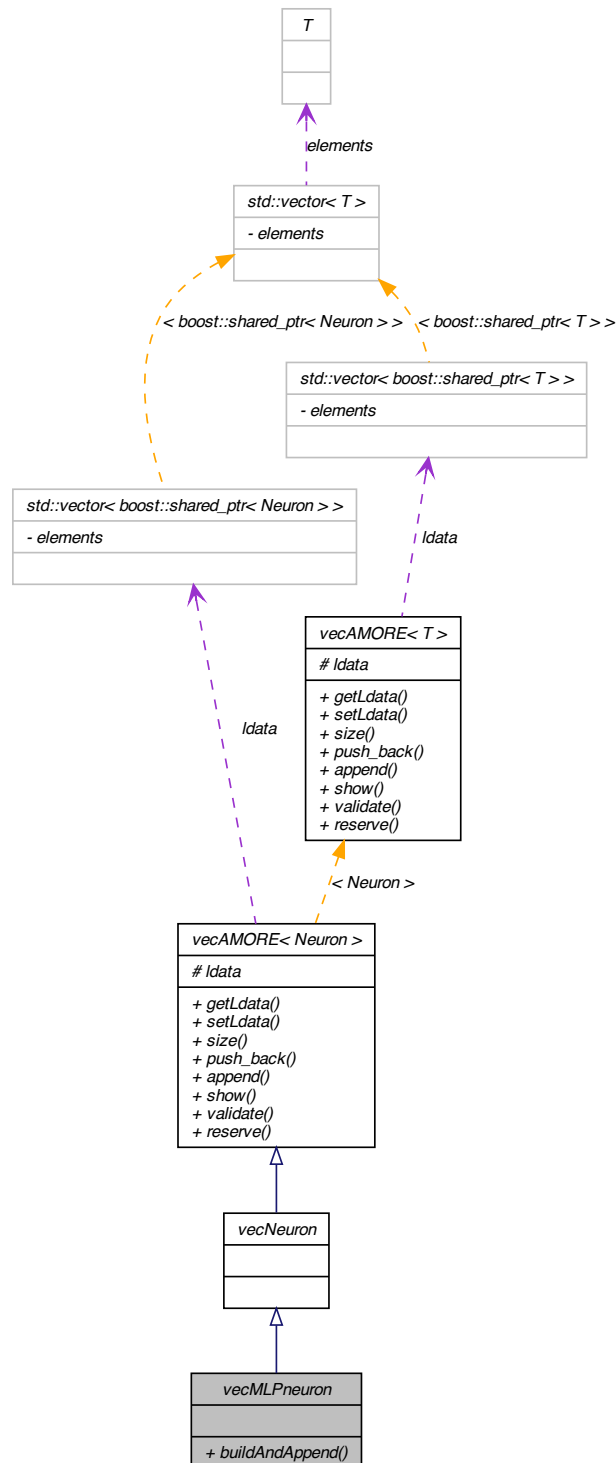
A vector of connections.

```
#include <vecMLPneuron.h>
```

Inheritance diagram for vecMLPneuron:



Collaboration diagram for vecMLPneuron:



## Public Member Functions

- bool [buildAndAppend](#) (std::vector< int > IDS, std::vector< int > BIAS, [vecCon VC](#))

### 6.6.1 Detailed Description

A vector of connections.

The [vecCon](#) class provides a simple class for a vector of connections. It's named after the R equivalent Reference Class.

Definition at line 17 of file [vecMLPneuron.h](#).

### 6.6.2 Member Function Documentation

- 6.6.2.1 bool [vecMLPneuron::buildAndAppend](#) ( std::vector< int > *IDS*, std::vector< int > *BIAS*, [vecCon VC](#) )

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

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

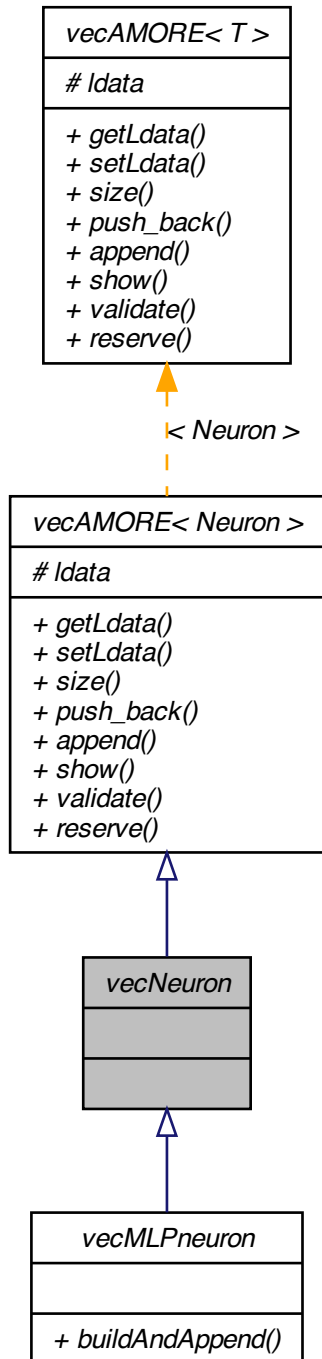
## 6.7 vecNeuron Class Reference

A vector of neurons.

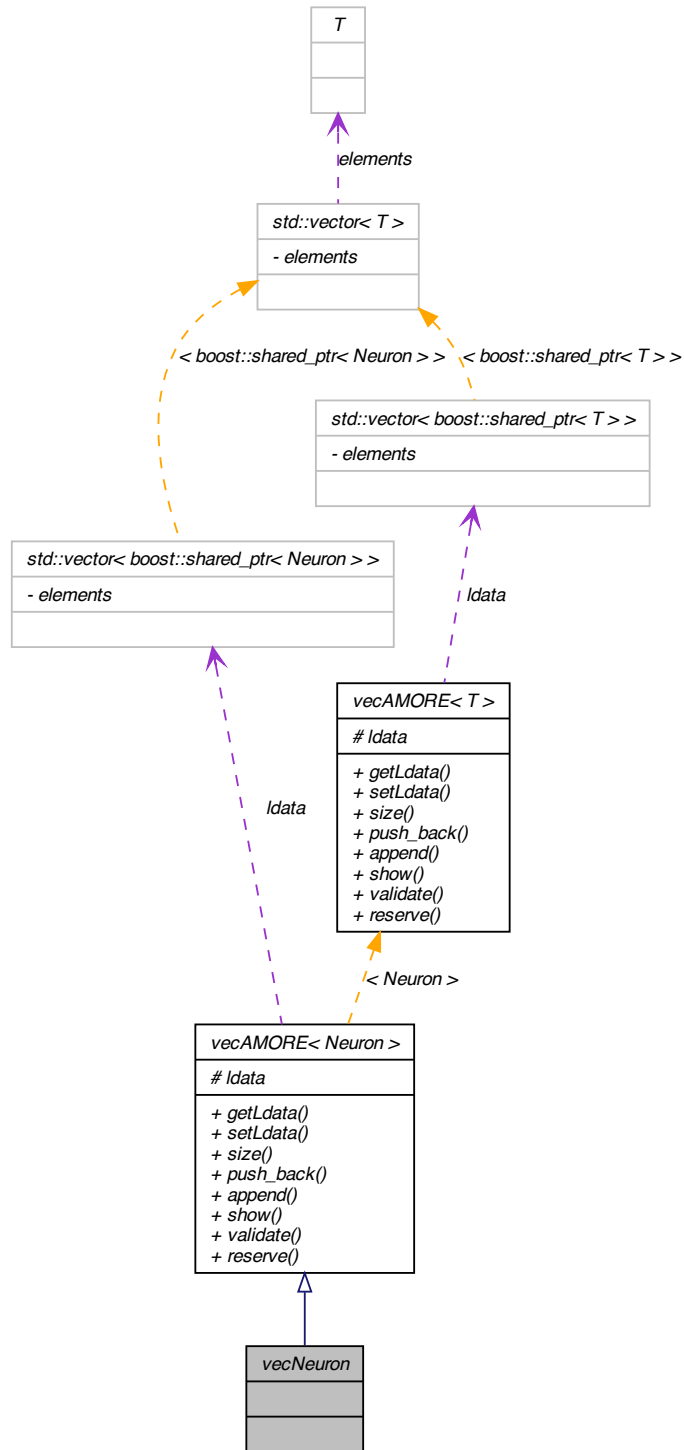
```
#include <vecNeuron.h>
```



Inheritance diagram for vecNeuron:



Collaboration diagram for vecNeuron:



### 6.7.1 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 18 of file [vecNeuron.h](#).

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

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



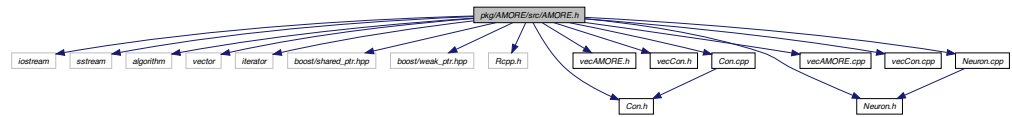
## Chapter 7

# File Documentation

### 7.1 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 <Rcpp.h>
#include "Con.h"
#include "vecAMORE.h"
#include "vecCon.h"
#include "Neuron.h"
#include "Con.cpp"
#include "vecAMORE.cpp"
#include "vecCon.cpp"
#include "Neuron.cpp"
```

Include dependency graph for AMORE.h:



## Typedefs

- `typedef boost::shared_ptr< Con > ConSharedPtr`
- `typedef boost::shared_ptr< Neuron > NeuronSharedPtr`
- `typedef boost::weak_ptr< Neuron > NeuronWeakPtr`
- `typedef boost::shared_ptr< vecAMORE< Con > > vecAMOREconSharedPtr`
- `typedef boost::shared_ptr< vecAMORE< Neuron > > vecAMOREneuronSharedPtr`
- `typedef boost::shared_ptr< vecCon > vecConSharedPtr`

### 7.1.1 Typedef Documentation

#### 7.1.1.1 `typedef boost::shared_ptr<Con> ConSharedPtr`

Definition at line 33 of file AMORE.h.

#### 7.1.1.2 `typedef boost::shared_ptr<Neuron> NeuronSharedPtr`

Definition at line 37 of file AMORE.h.

#### 7.1.1.3 `typedef boost::weak_ptr<Neuron> NeuronWeakPtr`

Definition at line 38 of file AMORE.h.

#### 7.1.1.4 `typedef boost::shared_ptr< vecAMORE<Con> > vecAMOREconSharedPtr`

Definition at line 39 of file AMORE.h.

#### 7.1.1.5 `typedef boost::shared_ptr< vecAMORE<Neuron> > vecAMOREneuronSharedPtr`

Definition at line 40 of file AMORE.h.

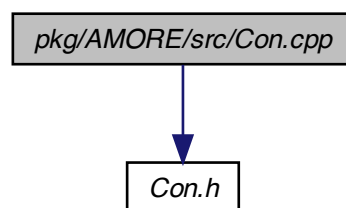
#### 7.1.1.6 `typedef boost::shared_ptr< vecCon > vecConSharedPtr`

Definition at line 41 of file AMORE.h.

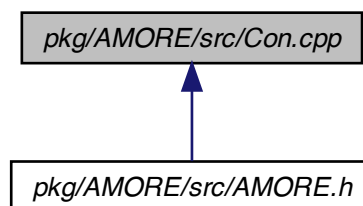
## 7.2 pkg/AMORE/src/Con.cpp File Reference

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

Include dependency graph for Con.cpp:

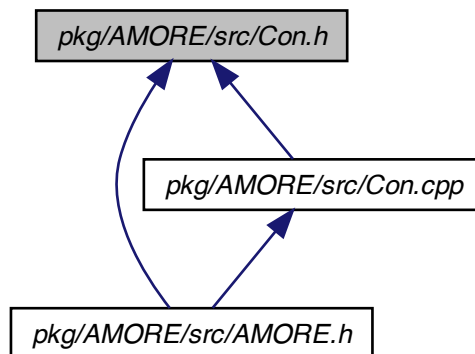


This graph shows which files directly or indirectly include this file:



### 7.3 pkg/AMORE/src/Con.h File Reference

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



#### Classes

- class [Con](#)

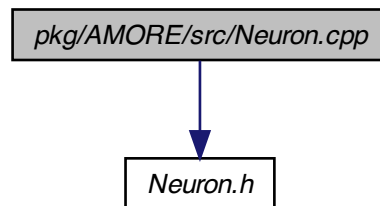
*A class to handle the information needed to describe an input connection.*

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

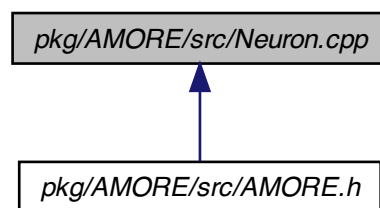
```
#include "Neuron.h"
```



Include dependency graph for Neuron.cpp:

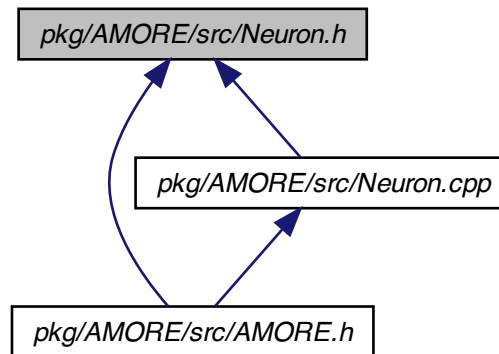


This graph shows which files directly or indirectly include this file:



## 7.5 pkg/AMORE/src/Neuron.h File Reference

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



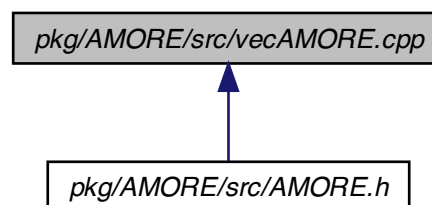
### Classes

- class [Neuron](#)

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

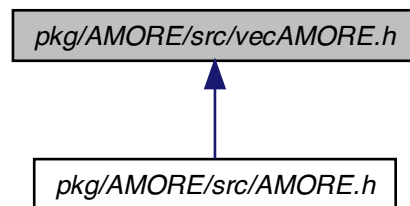
## 7.6 pkg/AMORE/src/vecAMORE.cpp File Reference

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



## 7.7 pkg/AMORE/src/vecAMORE.h File Reference

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

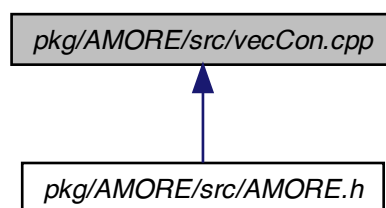


### Classes

- class `vecAMORE< T >`

## 7.8 pkg/AMORE/src/vecCon.cpp File Reference

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

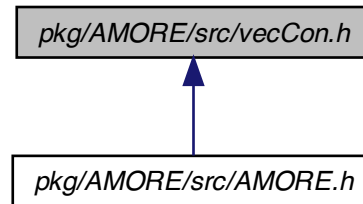


### Classes

- struct `CompareId`

## 7.9 pkg/AMORE/src/vecCon.h File Reference

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



### Classes

- class `vecCon`  
*A vector of connections.*

## 7.10 pkg/AMORE/src/vecMLPneuron.h File Reference

### Classes

- class `vecMLPneuron`  
*A vector of connections.*

## 7.11 pkg/AMORE/src/vecNeuron.h File Reference

### Classes

- class `vecNeuron`  
*A vector of neurons.*

# Index

- ~Con
  - Con, [14](#)
- ~Neuron
  - Neuron, [21](#)
- AMORE.h
  - ConSharedPtr, [56](#)
  - NeuronSharedPtr, [56](#)
  - NeuronWeakPtr, [56](#)
  - vecAMOREconSharedPtr, [56](#)
  - vecAMOREneuronSharedPtr, [56](#)
  - vecConSharedPtr, [56](#)
- append
  - vecAMORE, [26](#)
- buildAndAppend
  - vecCon, [36](#)
  - vecMLPneuron, [50](#)
- CompareId, [11](#)
  - operator(), [11](#), [12](#)
- Con, [12](#)
  - ~Con, [14](#)
  - Con, [13](#)
  - from, [20](#)
  - getFromId, [14](#)
  - getFromNeuron, [15](#)
  - getWeight, [16](#)
  - setFromNeuron, [17](#)
  - setWeight, [17](#)
  - show, [18](#)
  - validate, [19](#)
  - weight, [20](#)
- ConSharedPtr
  - AMORE.h, [56](#)
- erase
  - vecCon, [37](#)
- from
  - Con, [20](#)
- getFromId
  - Con, [14](#)
  - vecCon, [39](#)
- getFromNeuron
  - Con, [15](#)
  - vecCon, [40](#)
- getId
  - Neuron, [22](#)
- getLdata
  - vecAMORE, [28](#)
- getWeight
  - Con, [16](#)
  - vecCon, [41](#)
- Id
  - Neuron, [22](#)
- ldata
  - vecAMORE, [32](#)
- Neuron, [21](#)
  - ~Neuron, [21](#)
  - getId, [22](#)
  - Id, [22](#)
  - Neuron, [21](#)
  - outputValue, [22](#)
  - setId, [22](#)
- NeuronSharedPtr
  - AMORE.h, [56](#)
- NeuronWeakPtr
  - AMORE.h, [56](#)
- numOfCons
  - vecCon, [42](#)
- operator()
  - CompareId, [11](#), [12](#)
- outputValue
  - Neuron, [22](#)
- pkg/AMORE/src/AMORE.h, [55](#)
- pkg/AMORE/src/Con.cpp, [57](#)
- pkg/AMORE/src/Con.h, [58](#)

- pkg/AMORE/src/Neuron.cpp, 58
- pkg/AMORE/src/Neuron.h, 60
- pkg/AMORE/src/vecAMORE.cpp, 60
- pkg/AMORE/src/vecAMORE.h, 61
- pkg/AMORE/src/vecCon.cpp, 61
- pkg/AMORE/src/vecCon.h, 62
- pkg/AMORE/src/vecMLPneuron.h, 62
- pkg/AMORE/src/vecNeuron.h, 62
- push\_back
  - vecAMORE, 29
- reserve
  - vecAMORE, 30
- select
  - vecCon, 44
- setFromNeuron
  - Con, 17
  - vecCon, 44
- setId
  - Neuron, 22
- setLdata
  - vecAMORE, 30
- setWeight
  - Con, 17
  - vecCon, 45
- show
  - Con, 18
  - vecAMORE, 30
- size
  - vecAMORE, 31
- validate
  - Con, 19
  - vecAMORE, 32
- vecAMORE, 23
  - append, 26
  - getLdata, 28
  - ldata, 32
  - push\_back, 29
  - reserve, 30
  - setLdata, 30
  - show, 30
  - size, 31
  - validate, 32
- vecAMOREconSharedPtr
  - AMORE.h, 56
- vecAMOREneuronSharedPtr
  - AMORE.h, 56
- vecCon, 33
  - buildAndAppend, 36
  - erase, 37
  - getFromId, 39
  - getFromNeuron, 40
  - getWeight, 41
  - numOfCons, 42
  - select, 44
  - setFromNeuron, 44
  - setWeight, 45
- vecConSharedPtr
  - AMORE.h, 56
- vecMLPneuron, 47
  - buildAndAppend, 50
- vecNeuron, 50
- weight
  - Con, 20