

## AMORE++

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

Generated by Doxygen 1.7.4

Tue Jun 7 2011 18:42:35



# 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>Class Index</b>	<b>3</b>
2.1	Class Hierarchy . . . . .	3
<b>3</b>	<b>Class Index</b>	<b>5</b>
3.1	Class List . . . . .	5
<b>4</b>	<b>File Index</b>	<b>7</b>
4.1	File List . . . . .	7
<b>5</b>	<b>Class Documentation</b>	<b>9</b>
5.1	CompareId Struct Reference . . . . .	9
5.1.1	Detailed Description . . . . .	9
5.1.2	Member Function Documentation . . . . .	9
5.1.2.1	operator() . . . . .	9
5.1.2.2	operator() . . . . .	9
5.1.2.3	operator() . . . . .	10
5.1.2.4	operator() . . . . .	10
5.2	Con Class Reference . . . . .	10
5.2.1	Detailed Description . . . . .	11
5.2.2	Constructor & Destructor Documentation . . . . .	11
5.2.2.1	Con . . . . .	11
5.2.2.2	Con . . . . .	11

5.2.2.3	Con . . . . .	12
5.2.2.4	~Con . . . . .	12
5.2.3	Member Function Documentation . . . . .	12
5.2.3.1	getFrom . . . . .	12
5.2.3.2	getId . . . . .	13
5.2.3.3	getWeight . . . . .	14
5.2.3.4	setFrom . . . . .	15
5.2.3.5	setWeight . . . . .	15
5.2.3.6	show . . . . .	16
5.2.3.7	validate . . . . .	17
5.2.4	Member Data Documentation . . . . .	18
5.2.4.1	from . . . . .	18
5.2.4.2	weight . . . . .	18
5.3	Container< T > Class Template Reference . . . . .	18
5.3.1	Detailed Description . . . . .	21
5.3.2	Member Typedef Documentation . . . . .	21
5.3.2.1	const_iterator . . . . .	21
5.3.2.2	iterator . . . . .	22
5.3.3	Member Function Documentation . . . . .	22
5.3.3.1	append . . . . .	22
5.3.3.2	begin . . . . .	23
5.3.3.3	end . . . . .	24
5.3.3.4	load . . . . .	24
5.3.3.5	operator[] . . . . .	25
5.3.3.6	push_back . . . . .	25
5.3.3.7	reserve . . . . .	26
5.3.3.8	resize . . . . .	27
5.3.3.9	show . . . . .	27
5.3.3.10	size . . . . .	28
5.3.3.11	store . . . . .	29
5.3.3.12	validate . . . . .	29
5.3.4	Member Data Documentation . . . . .	30
5.3.4.1	collection . . . . .	30
5.4	Neuron Class Reference . . . . .	30

5.4.1	Detailed Description	32
5.4.2	Constructor & Destructor Documentation	32
5.4.2.1	Neuron	32
5.4.2.2	Neuron	33
5.4.2.3	Neuron	33
5.4.2.4	~Neuron	33
5.4.3	Member Function Documentation	33
5.4.3.1	getConId	33
5.4.3.2	getFrom	33
5.4.3.3	getId	34
5.4.3.4	getWeight	34
5.4.3.5	numOfCons	35
5.4.3.6	setFrom	36
5.4.3.7	setId	36
5.4.3.8	setWeight	36
5.4.3.9	show	37
5.4.3.10	validate	38
5.4.4	Member Data Documentation	38
5.4.4.1	con	38
5.4.4.2	id	38
5.4.4.3	outputValue	38
5.5	VecCon Class Reference	39
5.5.1	Detailed Description	42
5.5.2	Member Function Documentation	42
5.5.2.1	buildAndAppend	42
5.5.2.2	erase	44
5.5.2.3	getFrom	46
5.5.2.4	getId	47
5.5.2.5	getWeight	49
5.5.2.6	getWeight	51
5.5.2.7	numOfCons	52
5.5.2.8	select	54
5.5.2.9	setFrom	55
5.5.2.10	setWeight	57

5.5.2.11	setWeight	59
5.5.2.12	validate	61
5.6	vecMLPneuron Class Reference	62
5.6.1	Detailed Description	65
5.6.2	Member Function Documentation	65
5.6.2.1	buildAndAppend	65
5.7	vecNeuron Class Reference	65
5.7.1	Detailed Description	68
<b>6</b>	<b>File Documentation</b>	<b>69</b>
6.1	pkg/AMORE/src/AMORE.h File Reference	69
6.1.1	Define Documentation	70
6.1.1.1	foreach	70
6.1.1.2	size_type	70
6.1.2	Typedef Documentation	70
6.1.2.1	ConPtr	70
6.1.2.2	ContainerConPtr	70
6.1.2.3	ContainerNeuronPtr	71
6.1.2.4	NeuronPtr	71
6.1.2.5	NeuronWeakPtr	71
6.1.2.6	VecConPtr	71
6.1.2.7	VecNeuronPtr	71
6.2	pkg/AMORE/src/Con.cpp File Reference	71
6.3	pkg/AMORE/src/Con.h File Reference	72
6.4	pkg/AMORE/src/Container.cpp File Reference	73
6.5	pkg/AMORE/src/Container.h File Reference	73
6.6	pkg/AMORE/src/Neuron.cpp File Reference	73
6.7	pkg/AMORE/src/Neuron.h File Reference	75
6.8	pkg/AMORE/src/VecCon.cpp File Reference	75
6.9	pkg/AMORE/src/VecCon.h File Reference	76
6.10	pkg/AMORE/src/VecMLPneuron.h File Reference	76
6.11	pkg/AMORE/src/VecNeuron.h File Reference	77

## 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 > . . . . .	18
Container< Con > . . . . .	18
VecCon . . . . .	39
Container< Neuron > . . . . .	18
vecNeuron . . . . .	65
vecMLPneuron . . . . .	62
Neuron . . . . .	30



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">CompareId</a> . . . . .	9
<a href="#">Con</a> (A class to handle the information needed to describe an input connection )	10
<a href="#">Container&lt; T &gt;</a> . . . . .	18
<a href="#">Neuron</a> (A class to handle the information contained in a general <a href="#">Neuron</a> ) . .	30
<a href="#">VecCon</a> (A vector of connections ) . . . . .	39
<a href="#">vecMLPneuron</a> (A vector of connections ) . . . . .	62
<a href="#">vecNeuron</a> (A vector of neurons ) . . . . .	65



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

pkg/AMORE/src/ <a href="#">AMORE.h</a> . . . . .	69
pkg/AMORE/src/ <a href="#">Con.cpp</a> . . . . .	71
pkg/AMORE/src/ <a href="#">Con.h</a> . . . . .	72
pkg/AMORE/src/ <a href="#">Container.cpp</a> . . . . .	73
pkg/AMORE/src/ <a href="#">Container.h</a> . . . . .	73
pkg/AMORE/src/ <a href="#">Neuron.cpp</a> . . . . .	73
pkg/AMORE/src/ <a href="#">Neuron.h</a> . . . . .	75
pkg/AMORE/src/ <a href="#">VecCon.cpp</a> . . . . .	75
pkg/AMORE/src/ <a href="#">VecCon.h</a> . . . . .	76
pkg/AMORE/src/ <a href="#">VecMLPneuron.h</a> . . . . .	76
pkg/AMORE/src/ <a href="#">VecNeuron.h</a> . . . . .	77



## 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 369 of file `VecCon.cpp`.

#### 5.1.2 Member Function Documentation

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

Definition at line 371 of file `VecCon.cpp`.

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

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

Definition at line 383 of file `VecCon.cpp`.

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

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

Definition at line 379 of file VecCon.cpp.

```

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

```

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

Definition at line 375 of file VecCon.cpp.

```

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

```

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

- [pkg/AMORE/src/VecCon.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](#) f)  
*Constructor.*
- [Con](#) ([NeuronPtr](#) f, double w)  
*Constructor.*
- [~Con](#) ()  
*Default Destructor.*
- [NeuronPtr](#) [getFrom](#) ()  
*from field accessor.*
- void [setFrom](#) ([NeuronPtr](#) f)  
*from field accessor.*
- int [getId](#) ()  
*A getter of the Id of the [Neuron](#) pointed by the from field.*
- double [getWeight](#) ()  
*weight field accessor.*



- void `setWeight` (double 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.*

#### 5.2.1 Detailed Description

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

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

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

#### 5.2.2 Constructor & Destructor Documentation

##### 5.2.2.1 `Con::Con ( )`

Default Constructor.

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

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

##### 5.2.2.2 `Con::Con ( NeuronPtr f )`

Constructor.

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

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

### 5.2.2.3 Con::Con ( NeuronPtr f, double w )

Constructor.

Definition at line 28 of file Con.cpp.

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

### 5.2.2.4 Con::~Con ( )

Default Destructor.

Definition at line 41 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 65 of file Con.cpp.

References from.

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

## 5.2.3.2 int Con::getId ( )

A getter of the Id of the [Neuron](#) pointed by the from field.

This method gets the Id of the [Neuron](#) referred to by the [from](#) field

**Returns**

The value of the Id (an integer).

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

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

**See also**

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

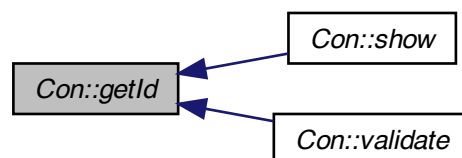
Definition at line 117 of file `Con.cpp`.

References from.

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

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

Here is the caller graph for this function:



### 5.2.3.3 double Con::getWeight ( )

weight field accessor.

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

#### Returns

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

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

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

#### See also

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

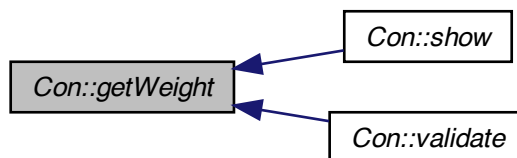
Definition at line 151 of file `Con.cpp`.

References [weight](#).

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

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

Here is the caller graph for this function:



## 5.2.3.4 void Con::setFrom ( NeuronPtr f )

from field accessor.

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

## Parameters

<a href="#">f</a>	A pointer to the neuron that is to be inserted in the <a href="#">from</a> 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 92 of file `Con.cpp`.

References from.

```

{
    from=f;
}
```

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

weight field accessor.

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

## Parameters

<a href="#">w</a>	The new value (double) to be set in the <a href="#">weight</a> 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 180 of file `Con.cpp`.

References `weight`.

```

{
    weight = w;
}

```

#### 5.2.3.6 `bool Con::show ( )`

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

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

### Returns

true in case everything works without throwing an exception

### See also

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

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

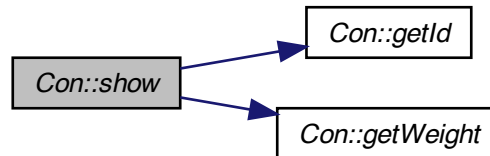
References `getId()`, and `getWeight()`.

```

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

```

Here is the call graph for this function:



#### 5.2.3.7 bool Con::validate ( )

Object validator.

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

#### Returns

true in case the checks are Ok.

#### Exceptions

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

Definition at line 211 of file Con.cpp.

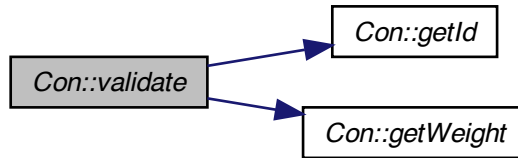
References `getId()`, and `getWeight()`.

```

    {
        BEGIN_RCPP
        if (! R_FINITE(getWeight()) )           throw std::range_error("weight is
not finite.");
        if (getId() == NA_INTEGER )             throw std::range_error("fromId is
not finite.");
        return(true);
        END_RCPP
    };

```

Here is the call graph for this function:



## 5.2.4 Member Data Documentation

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

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

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

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

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

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

A double variable that contains the weight of the connection.

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

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

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

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

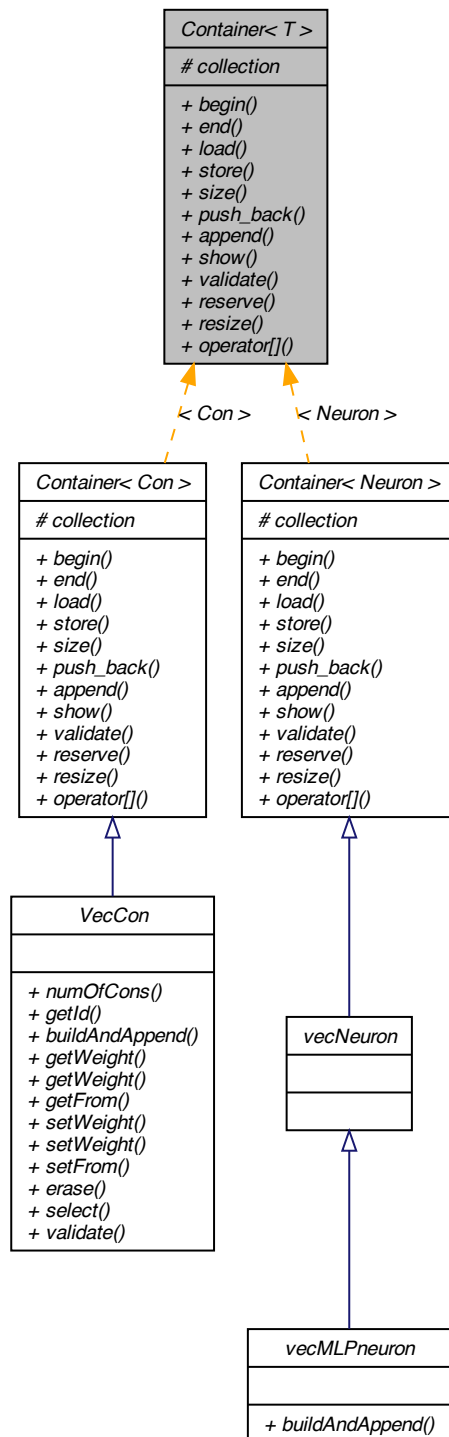
- `pkg/AMORE/src/Con.h`
- `pkg/AMORE/src/Con.cpp`

## 5.3 `Container< T >` Class Template Reference

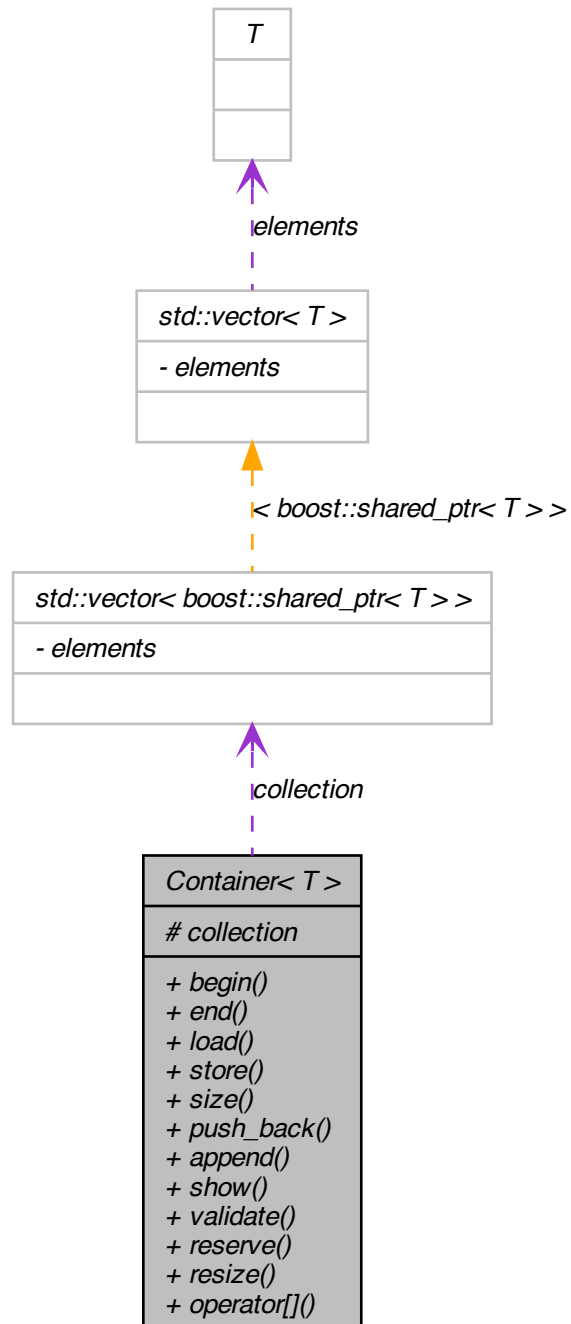
```
#include <Container.h>
```



Inheritance diagram for Container< T >:



Collaboration diagram for `Container< T >`:



## Public Types

- typedef std::vector< boost::shared\_ptr< T > >::iterator iterator
- typedef std::vector< boost::shared\_ptr< T > >::const\_iterator const\_iterator

## Public Member Functions

- iterator begin ()
- iterator end ()
- std::vector< boost::shared\_ptr< T > > load ()  
*collection field accessor function*
- void store (typename std::vector< boost::shared\_ptr< T > >)  
*collection field accessor function*
- size\_type 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 collection.*
- void append (Container< T > v)  
*Appends a Container<T> object.*
- bool show ()  
*Pretty print of the Container<T>*
- bool validate ()  
*Object validator.*
- void reserve (int n)
- void resize (int n)
- boost::shared\_ptr< T > & operator[] (size\_type offset)

## Protected Attributes

- std::vector< boost::shared\_ptr< T > > collection

### 5.3.1 Detailed Description

template<typename T>class Container< T >

Definition at line 12 of file Container.h.

### 5.3.2 Member Typedef Documentation

5.3.2.1 template<typename T> typedef std::vector<boost::shared\_ptr<T>  
>::const\_iterator Container< T >::const\_iterator

Definition at line 19 of file Container.h.

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

Definition at line 18 of file Container.h.

## 5.3.3 Member Function Documentation

### 5.3.3.1 `template<typename T> void Container< T >::append ( Container< T > v )`

Appends a Container<T> object.

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

#### Parameters

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

#### See also

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

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

std::vector<int> result;
std::vector<ConPtr> vcA, vcB;
ContainerNeuronPtr ptShvNeuron( new
Container<Neuron>() );
ContainerConPtr ptShvConA( new Container<Con>() )
;
ContainerConPtr ptShvConB( new Container<Con>() )
;

ConPtr ptC;
NeuronPtr ptN;
int ids[] = {1, 2, 3, 4, 5, 6};
double weights[] = {1.13, 2.22, 3.33, 5.6, 4.2, 3
.6 };

for (int i=0; i<=5 ; i++) {
/ Let's create a vector with six neurons
ptN.reset( new Neuron( ids[i] ) );
ptShvNeuron->push_back(ptN);
}
for (int i=0; i<=2 ; i++) {
/ A vector with three connections
ptC.reset( new Con( ptShvNeuron->load().a
t(i), weights[i]) );
ptShvConA->push_back(ptC);
}
for (int i=3; i<=5 ; i++) {
/ Another vector with three connections
ptC.reset( new Con( ptShvNeuron->load().a
t(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**

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

Definition at line 172 of file `Container.cpp`.

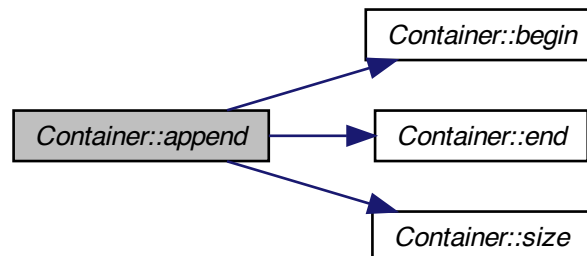
References `Container< T >::begin()`, `Container< T >::end()`, and `Container< T >::size()`.

```

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

```

Here is the call graph for this function:



### 5.3.3.2 `template<typename T> iterator Container< T >::begin ( ) [inline]`

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

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

```

{
    return collection.begin(); }

```

Here is the caller graph for this function:



### 5.3.3.3 `template<typename T> iterator Container< T >::end ( ) [inline]`

Definition at line 22 of file Container.h.

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

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

Here is the caller graph for this function:



### 5.3.3.4 `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 ptShvNeuron( new
Container<Neuron>() );

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

// Test
    ptShvCon->store(vcA);
    vcB = ptShvCon->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 217 of file `Container.cpp`.

```

return collection;
};

```

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

Definition at line 255 of file `Container.cpp`.

```

{
    return collection[offset];
}

```

### 5.3.3.6 `template<typename T> void Container< T>::push_back ( boost::shared_ptr< T> TsharedPtr )`

Append a `shared_ptr` at the end of collection.

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

**Parameters**

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

```

//=====
//Usage example:
//=====
// Data set up
    Neuron N1, N2, N3;
    Container<Con> MyVecCon;
    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
    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.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 44 of file `Container.cpp`.

```

{
    collection.push_back(TsharedPtr);
};

```

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

Definition at line 250 of file `Container.cpp`.

```

{
    collection.reserve(n) ;
};

```



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

Definition at line 245 of file Container.cpp.

```

{
    collection.resize(n);
}

```

5.3.3.9 `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      ptShvNeuron( new
Container<Neuron>() );
ContainerConPtr ptShvCon( 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] ) );
        ptShvNeuron->push_back(ptN);
    }

    for (int i=0; i<=2 ; i++) {
/ and a vector with three connections
        ptC.reset( new Con( ptShvNeuron->load().at(i), we
ights[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.Container.R`, for usage examples.

Definition at line 93 of file Container.cpp.

Referenced by Neuron::show().

```

{

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

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

```

Here is the caller graph for this function:



### 5.3.3.10 `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 240 of file Container.cpp.

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

```

{

    return collection.size() ;
};

```

Here is the caller graph for this function:



**5.3.3.11** `template<typename T> void Container< T >::store ( typename std::vector< boost::shared_ptr< T > > v )`

collection field accessor function

This method sets the value of the data stored in the [collection](#) field.

#### Parameters

<code>v</code>	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 229 of file `Container.cpp`.

```

{
    collection=v;
};

```

**5.3.3.12** `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 [VecCon](#).

Definition at line 112 of file `Container.cpp`.

```

{
    foreach (typename boost::shared_ptr<T> itr, *this){

```

```
        itr->validate();  
    }  
    return true;  
};
```

### 5.3.4 Member Data Documentation

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

Definition at line 14 of file Container.h.

Referenced by `Container< Neuron >::begin()`, and `Container< Neuron >::end()`.

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

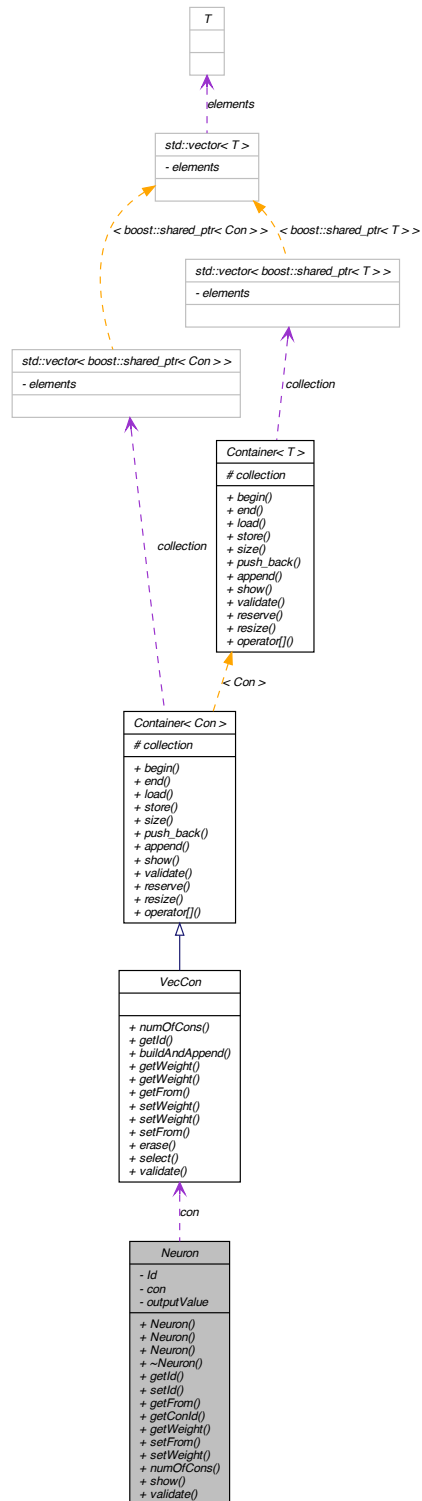
- pkg/AMORE/src/[Container.h](#)
- pkg/AMORE/src/[Container.cpp](#)

## 5.4 Neuron Class Reference

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

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

Collaboration diagram for Neuron:



## Public Member Functions

- [Neuron](#) ()
- [Neuron](#) (int [Id](#))
- [Neuron](#) (int [Id](#), [VecCon](#) [Con](#))
- [~Neuron](#) ()
- int [getId](#) ()
- void [setId](#) (int [Id](#))
- std::vector< [NeuronPtr](#) > [getFrom](#) ()
- std::vector< int > [getConId](#) ()
- std::vector< double > [getWeight](#) ()
- bool [setFrom](#) (std::vector< [NeuronPtr](#) > [vFrom](#))
- bool [setWeight](#) (std::vector< double > [vWeight](#))
- int [numOfCons](#) ()
- bool [show](#) ()
- bool [validate](#) ()

## Private Attributes

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

### 5.4.1 Detailed Description

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

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

Definition at line 16 of file Neuron.h.

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 [Neuron::Neuron](#) ( )

Definition at line 12 of file Neuron.cpp.

```
: Id (NA_INTEGER), con () {};
```

**5.4.2.2 Neuron::Neuron ( int *Id* )**

Definition at line 13 of file Neuron.cpp.

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

**5.4.2.3 Neuron::Neuron ( int *Id*, VecCon *Con* )**

Definition at line 14 of file Neuron.cpp.

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

**5.4.2.4 Neuron::~Neuron ( )**

Definition at line 15 of file Neuron.cpp.

```
{};
```

**5.4.3 Member Function Documentation****5.4.3.1 std::vector< int > Neuron::getConId ( )**

Definition at line 32 of file Neuron.cpp.

References con, and VecCon::getId().

```

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

Here is the call graph for this function:

**5.4.3.2 std::vector< NeuronPtr > Neuron::getFrom ( )**

Definition at line 27 of file Neuron.cpp.

References con, and VecCon::getFrom().

```

    {
        return con.getFrom();
    }

```

Here is the call graph for this function:



#### 5.4.3.3 int *Neuron::getId* ( )

Definition at line 18 of file *Neuron.cpp*.

References *Id*.

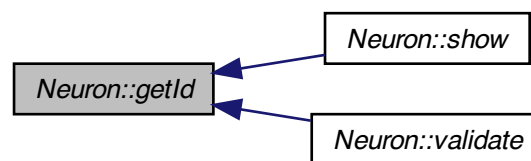
Referenced by *show()*, and *validate()*.

```

    {
        return Id;
    }

```

Here is the caller graph for this function:



#### 5.4.3.4 std::vector< double > *Neuron::getWeight* ( )

Definition at line 37 of file *Neuron.cpp*.

References *con*, and *VecCon::getWeight()*.

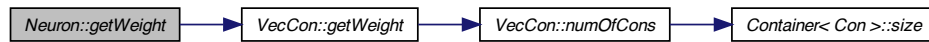


```

    {
        return con.getWeight();
    }

```

Here is the call graph for this function:



#### 5.4.3.5 int Neuron::numOfCons ( )

Definition at line 51 of file Neuron.cpp.

References `con`, and `VecCon::numOfCons()`.

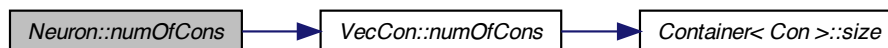
Referenced by `show()`.

```

    {
        return con.numOfCons();
    }

```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.4.3.6 `bool Neuron::setFrom ( std::vector< NeuronPtr > vFrom )`

Definition at line 42 of file Neuron.cpp.

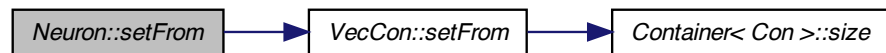
References `con`, and `VecCon::setFrom()`.

```

    {
        con.setFrom(vFrom);
    }

```

Here is the call graph for this function:



#### 5.4.3.7 `void Neuron::setId ( int Id )`

Definition at line 22 of file Neuron.cpp.

References `Id`.

```

    {
        Id=id;
    }

```

#### 5.4.3.8 `bool Neuron::setWeight ( std::vector< double > vWeight )`

Definition at line 47 of file Neuron.cpp.

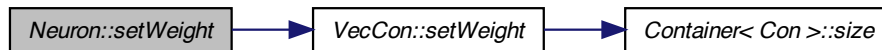
References `con`, and `VecCon::setWeight()`.

```

    {
        con.setWeight(vWeight);
    }

```

Here is the call graph for this function:



#### 5.4.3.9 bool Neuron::show ( )

Definition at line 55 of file Neuron.cpp.

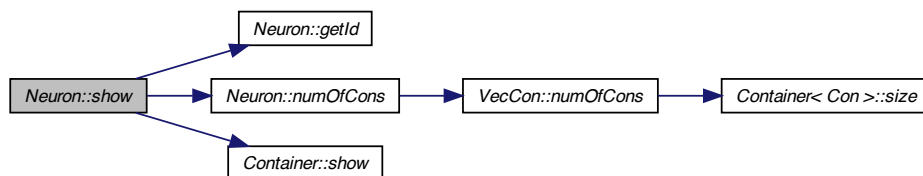
References `con`, `getId()`, `numOfCons()`, and `Container< T >::show()`.

```

{
    int id=getId();
    Rprintf("\n-----\n");
    if (id==NA_INTEGER) {
        Rprintf("\n Id: NA, Invalid neuron Id");
    } else {
        Rprintf("\n Id: %d", id);
    }
    Rprintf("\n-----\n");
    if (numOfCons()==0) {
        Rprintf("\n No connections defined");
    } else {
        con.show();
    }
    Rprintf("\n-----\n");
    return true;
}

```

Here is the call graph for this function:



#### 5.4.3.10 bool Neuron::validate ( )

Definition at line 75 of file Neuron.cpp.

References con, getId(), and VecCon::validate().

```

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

```

Here is the call graph for this function:



### 5.4.4 Member Data Documentation

#### 5.4.4.1 VecCon Neuron::con [private]

A vector of input connections.

Definition at line 28 of file Neuron.h.

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

#### 5.4.4.2 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()`.

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

Definition at line 29 of file Neuron.h.

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

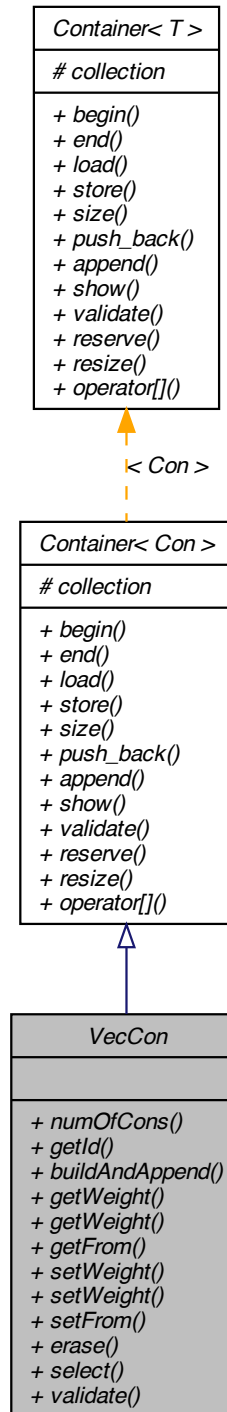
- pkg/AMORE/src/[Neuron.h](#)
- pkg/AMORE/src/[Neuron.cpp](#)

## 5.5 VecCon Class Reference

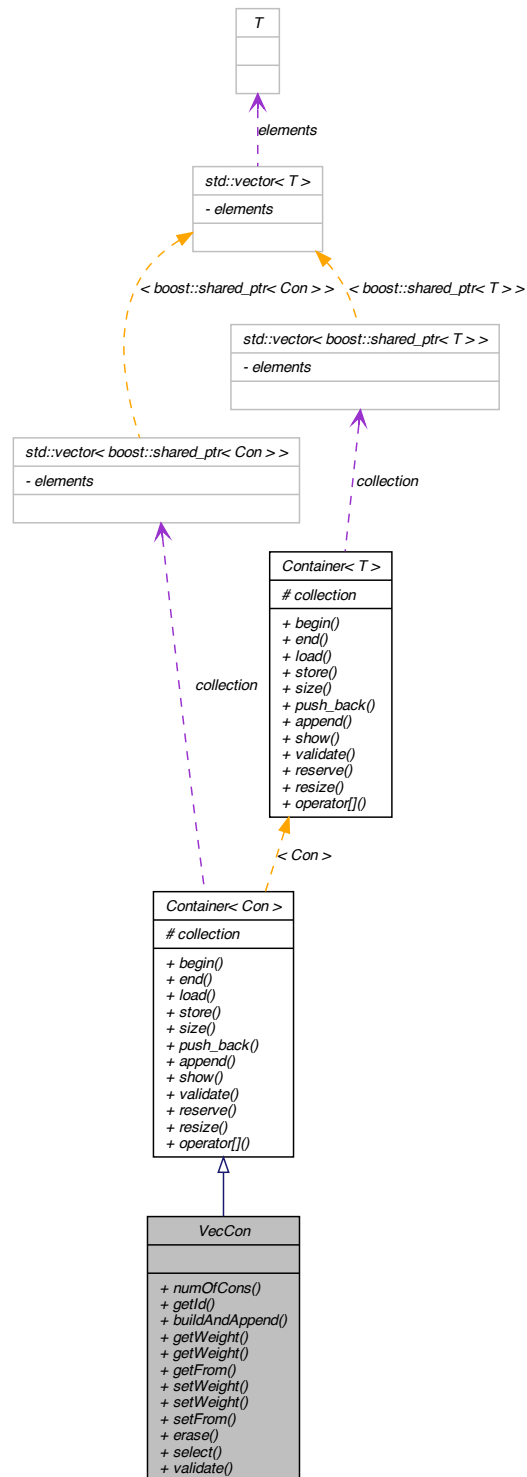
A vector of connections.

```
#include <VecCon.h>
```

Inheritance diagram for VecCon:



Collaboration diagram for VecCon:



## Public Member Functions

- int [numOfCons](#) ()  
*Size of the [VecCon](#) object.*
- std::vector< int > [getId](#) ()  
*Getter of the Id values of the vector of Cons.*
- bool [buildAndAppend](#) (std::vector< [NeuronPtr](#) > vFrom, std::vector< double > vWeight)  
*Builds [Con](#) objects and appends them to collection.*
- std::vector< double > [getWeight](#) ()  
*Getter of the weight field of the [Con](#) objects related to [VecCon](#).*
- std::vector< double > [getWeight](#) (std::vector< int > vFrom)  
*Getter of the weights of the specified elements from the [vecCom](#) object.*
- std::vector< [NeuronPtr](#) > [getFrom](#) ()  
*Getter of the from field of the [Con](#) objects related to [VecCon](#).*
- bool [setWeight](#) (std::vector< double > vWeight)  
*Setter of the weight field of the [Con](#) objects related to [VecCon](#).*
- bool [setWeight](#) (std::vector< double > vWeight, std::vector< int > vFrom)  
*Setter of the weights of the specified elements from the [VecCon](#) object.*
- bool [setFrom](#) (std::vector< [NeuronPtr](#) > vFrom)  
*Setter of the from fields of the [Con](#) objects related to [VecCon](#).*
- void [erase](#) (std::vector< int > vFrom)  
*Erase the specified elements from the [vecCom](#) object.*
- [VecConPtr](#) [select](#) (std::vector< int > vFrom)  
*Selects the specified elements from the [vecCom](#) object.*
- bool [validate](#) ()  
*Object validator.*

### 5.5.1 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](#).

### 5.5.2 Member Function Documentation

#### 5.5.2.1 bool [VecCon::buildAndAppend](#) ( std::vector< [NeuronPtr](#) > vFrom, std::vector< double > vWeight )

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

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



## 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<NeuronPtr> vNeuron;
    std::vector<double> vWeight;

// Test
    NeuronPtr 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.getId();

// 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.

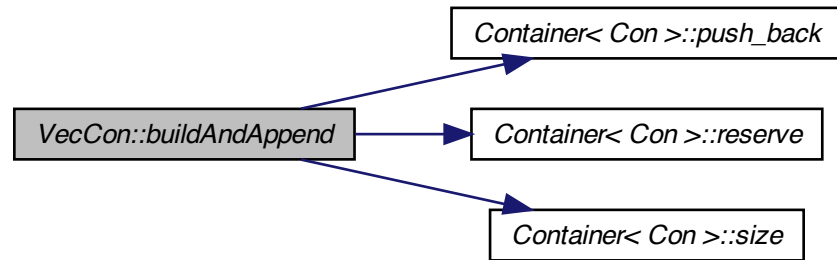
Definition at line 134 of file `VecCon.cpp`.

References `Container< Con >::push_back()`, `Container< Con >::reserve()`, and `Container< Con >::size()`.

```
{
    BEGIN_RCPP
    if (vFrom.empty()) { throw std::range_error("[VecCon::append]: Error, vFrom is empty"); }
    if (vFrom.size() != vWeight.size() ) { throw std::range_error("[VecCon::buildAndAppend]: Error, vFrom.size() != vWeight.size()"); }
    reserve(size() + vFrom.size());
    ConPtr ptCon;
    std::vector<double>::iterator itrWeight = vWeight.begin();

    foreach (NeuronPtr itrFrom, vFrom){
        ptCon.reset( new Con( itrFrom, *itrWeight) );
        push_back(ptCon);
        itrWeight++;
    }
    return true;
    END_RCPP
}
```

Here is the call graph for this function:



#### 5.5.2.2 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 collection field of the [VecCon](#) 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> vNeuron;
VecConPtr ptShvCon( new VecCon() );
VecConPtr vErased;
ConPtr ptC;
NeuronPtr 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->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.VecCon.R`, for further examples.

Definition at line 455 of file `VecCon.cpp`.

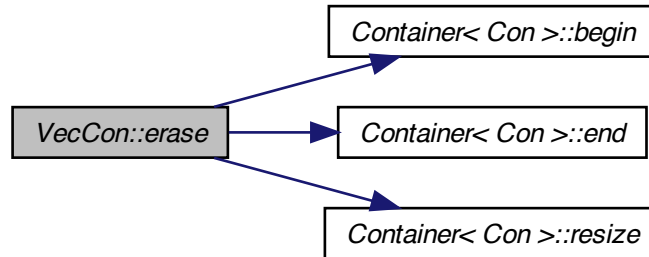
References `Container< Con >::begin()`, `Container< Con >::end()`, and `Container< Con >::resize()`.

```

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

```

Here is the call graph for this function:



#### 5.5.2.3 std::vector< NeuronPtr > VecCon::getFrom ( )

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<NeuronPtr> 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<NeuronPtr> vNeuron;
std::vector<double> vWeight;
NeuronPtr 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.getFrom();
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**

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

Definition at line 298 of file `VecCon.cpp`.

References `numOfCons()`.

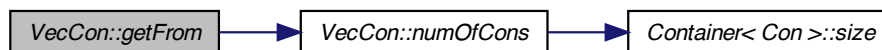
Referenced by `Neuron::getFrom()`.

```

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

```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.5.2.4 `std::vector< int > VecCon::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;
VecCon MyVecCon;
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
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.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.VecCon.R`, for further examples.

Definition at line 85 of file `VecCon.cpp`.

References `numOfCons()`.

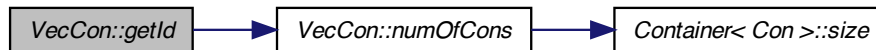
Referenced by `Neuron::getConId()`, and `validate()`.

```

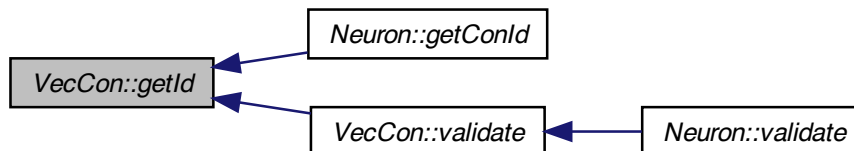
{
std::vector<int> result;
result.reserve(numOfCons());
foreach (ConPtr itr, *this){
    result.push_back(itr->getId());
}
return result;
}

```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.5.2.5 `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<NeuronPtr> vNeuron;
std::vector<double> vWeight;

// Test
NeuronPtr 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**

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

Definition at line 191 of file `VecCon.cpp`.

References `numOfCons()`.

Referenced by `Neuron::getWeight()`.

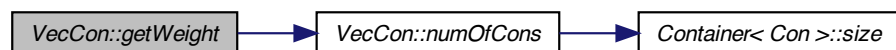
```

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

    return result;
}

```

Here is the call graph for this function:



Here is the caller graph for this function:





5.5.2.6 `std::vector< double > VecCon::getWeight ( std::vector< int > vFrom )`

Getter of the weights of the specified elements from the vecCom object.

Provides a convenient way of getting the weights of some [Con](#) objects from the collection field of the [VecCon](#) object.

**Parameters**

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

**Returns**

An `std::vector<double>` with the weights of the selected connections

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

// Data set up

    std::vector<double> result;
    std::vector<NeuronPtr> vNeuron;
    VecConPtr      ptShvCon( new VecCon() );
    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> vWeight;
    for (int i=0; i<11; i++) {
        vWeight.push_back(weights[i]);
    }
    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> toSelect;
    toSelect.push_back(1);
    toSelect.push_back(3);
    toSelect.push_back(5);
    toSelect.push_back(7);

    result=ptShvCon->getWeight(toSelect);

// Now, result is a numeric vector with the values 10.31, 3.16, 5.19 and 7.16.
```

**See also**

`setWeigh` and the unit test files, e.g. `runit.Cpp.VecCon.R`, for further examples.

Definition at line 568 of file `VecCon.cpp`.

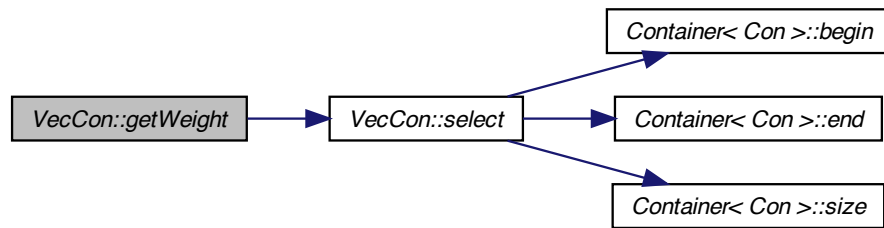
References `select()`.

```

    return  select (vFrom)->getWeight();
}

```

Here is the call graph for this function:



#### 5.5.2.7 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

Container<Neuron>( ) );

std::vector<int> result;
std::vector<ConPtr> vcA, vcB;
ContainerNeuronPtr ptShvNeuron( new
VecConPtr ptShvCon( new VecCon() );
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] ) );
    ptShvNeuron->push_back(ptN);
}

// Test
for (int i=0; i<=2 ; i++) {
    result.push_back(ptShvCon->numOfCons());
}

```

```

/ Append numOfCons to result, create new Con and push_back into MyVecCon
                                ptC.reset( new Con( ptShvNeuron->load().a
t(i), weights[i]) );                                ptShvCon->push_back(ptC);
                                }

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

```

**See also**

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

Definition at line 42 of file VecCon.cpp.

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

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

```

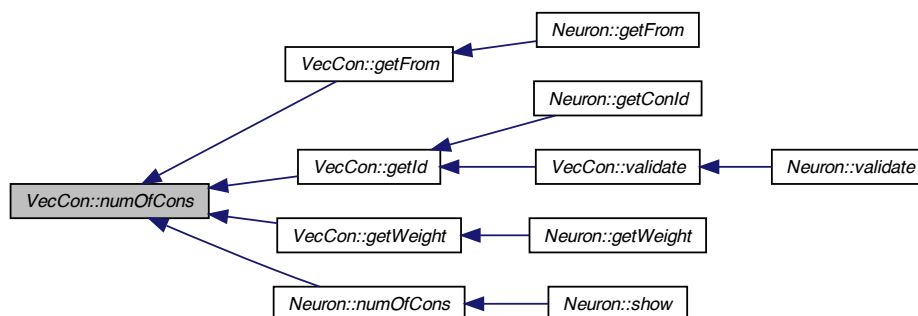
{
    return size();
}

```

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.5.2.8 VecConPtr VecCon::select ( std::vector< int > vFrom )

Selects the specified elements from the vecCom object.

Provides a convenient way of selecting some [Con](#) objects from the collection field of the [VecCon](#) object.

#### Parameters

<i>vFrom</i>	An std::vector<int> with the ids of the connections to select.
--------------	--

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

// Data set up
std::vector<int> result;
std::vector<NeuronPtr> vNeuron;
VecConPtr          ptShvCon( new VecCon() );
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> vWeight;
for (int i=0; i<11; i++) {
    vWeight.push_back(weights[i]);
}
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> toSelect;
toSelect.push_back(1);
toSelect.push_back(3);
toSelect.push_back(5);
toSelect.push_back(7);

VecConPtr vSelect ( ptShvCon->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.VecCon.R`, for further examples.

Definition at line 507 of file `VecCon.cpp`.

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

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

{

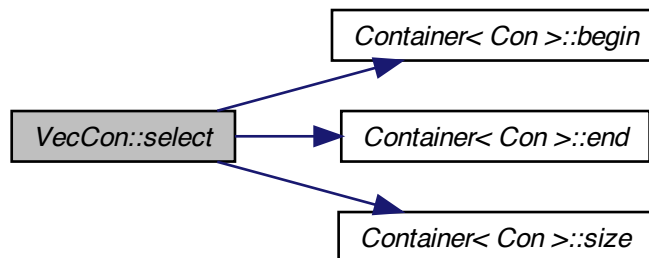
```

VecConPtr result(new VecCon );
result->reserve(size());
sort (begin(), end(), CompareId());
sort (vFrom.begin(), vFrom.end());
set_intersection(begin(), end(), vFrom.begin(), vFrom.end(), back_inserter(
result->collection) , CompareId());

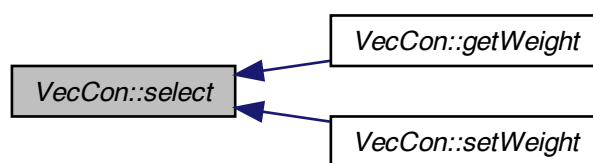
return result;
}

```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.5.2.9 bool VecCon::setFrom ( std::vector< NeuronPtr > 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;NeuronPtr&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;
ContainerNeuronPtr ptShvNeuron( new Container<Neuron>() );
VecConPtr ptShvCon( new VecCon() );
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] ) );
    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->setFrom(ptShvNeuron->load()) ;
ptShvCon->show();
result=ptShvCon->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.VecCon.R`, for further examples.

Definition at line 353 of file `VecCon.cpp`.

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

Referenced by `Neuron::setFrom()`.

```
{
BEGIN_RCPP
    if (vFrom.empty()) { throw std::range_error("[ C++ VecCon::setFrom]: Error, w is empty"); }
    if (vFrom.size() != size() ) { throw std::range_error("[C++ VecCon::setFrom]: Error, w.size() != collection.size()"); }
    std::vector<NeuronPtr>::iterator itrFrom = vFrom.begin();
    foreach(ConPtr itr , *this) {
        itr->setFrom( *itrFrom );
        itrFrom++;
    }
}
```

```

        return true;
    END_RCPP
}

```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.5.2.10 bool VecCon::setWeight ( std::vector< double > vWeight, std::vector< int > vFrom )

Setter of the weights of the specified elements from the [VecCon](#) object.

Provides a convenient way of setting the weights of some [Con](#) objects from the collection field of the [VecCon](#) object.

##### Parameters

<i>vWeight</i>	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.
<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> vNeuron;
VecConPtr ptShvCon( new VecCon() );
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> vWeight;
for (int i=0; i<11; i++) {
vWeight.push_back(weights[i]);
}
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);

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);
ptShvCon->setWeight(vNewWeights, toSelect);

// Test
result = ptShvCon->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

getWeigh and the unit test files, e.g. `runit.Cpp.VecCon.R`, for further examples.

Definition at line 630 of file `VecCon.cpp`.

References `select()`.

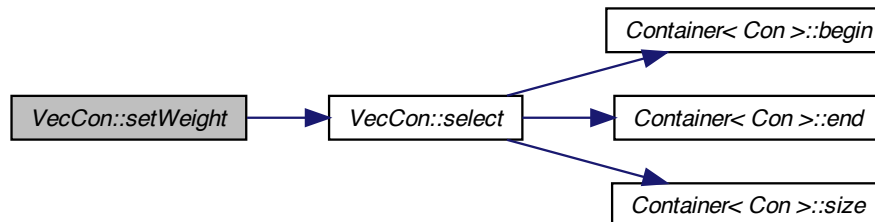
```

{
BEGIN_RCPP
return select(vFrom)->setWeight(vWeight);
END_RCPP
}

```



Here is the call graph for this function:



#### 5.5.2.11 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

<i>vWeight</i>	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<NeuronPtr> vNeuron;
std::vector<double> vWeight;
NeuronPtr 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 244 of file `VecCon.cpp`.

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

Referenced by `Neuron::setWeight()`.

```

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

```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.5.2.12 bool VecCon::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 [VecCon](#) 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.VecCon.R`, for usage examples.

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

Definition at line 655 of file `VecCon.cpp`.

References `getId()`.

Referenced by `Neuron::validate()`.

```

{
BEGIN_RCPP

std::vector<int>::iterator itr;

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

```

```
};
```

Here is the call graph for this function:



Here is the caller graph for this function:



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

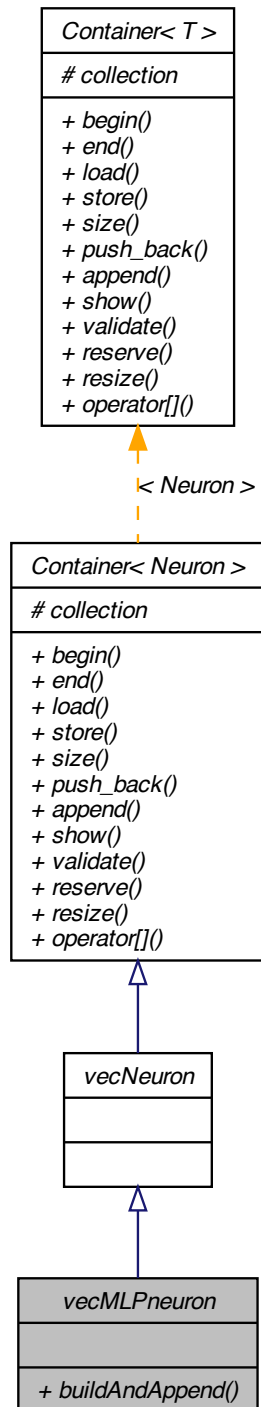
- pkg/AMORE/src/[VecCon.h](#)
- pkg/AMORE/src/[VecCon.cpp](#)

## 5.6 vecMLPneuron Class Reference

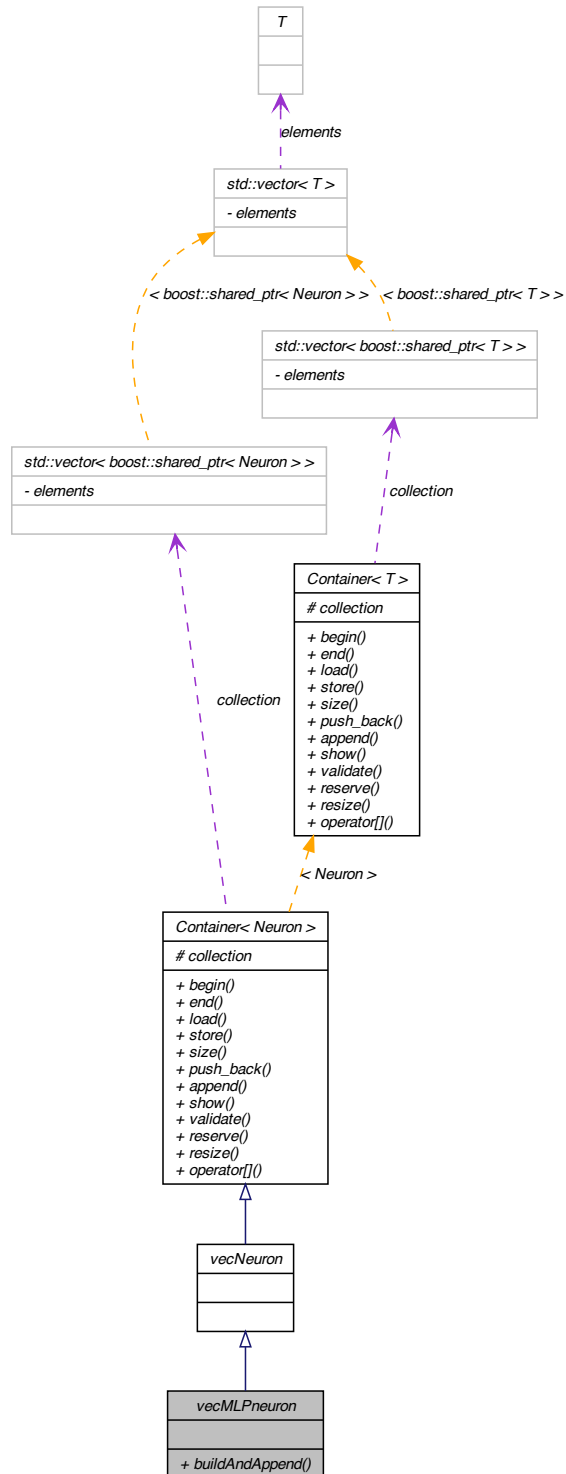
A vector of connections.

```
#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)

### 5.6.1 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.

### 5.6.2 Member Function Documentation

5.6.2.1 bool vecMLPneuron::buildAndAppend ( std::vector< int > *IDS*, std::vector< int > *BIAS*, [VecCon](#) *VC* )

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

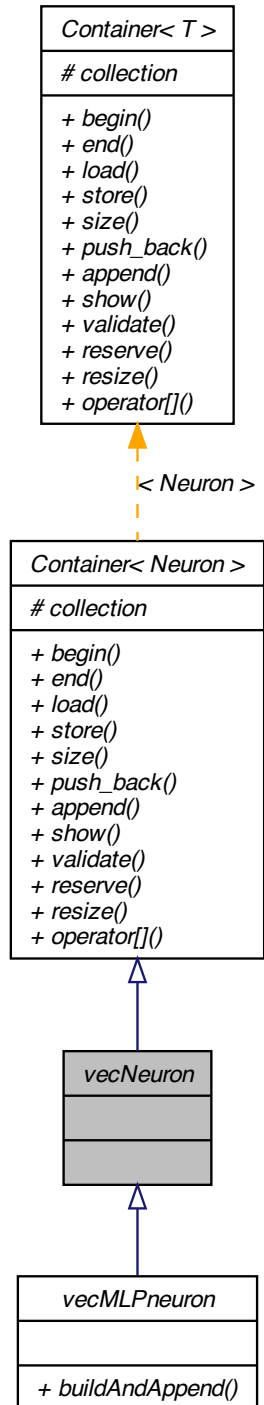
- pkg/AMORE/src/[VecMLPneuron.h](#)

## 5.7 vecNeuron Class Reference

A vector of neurons.

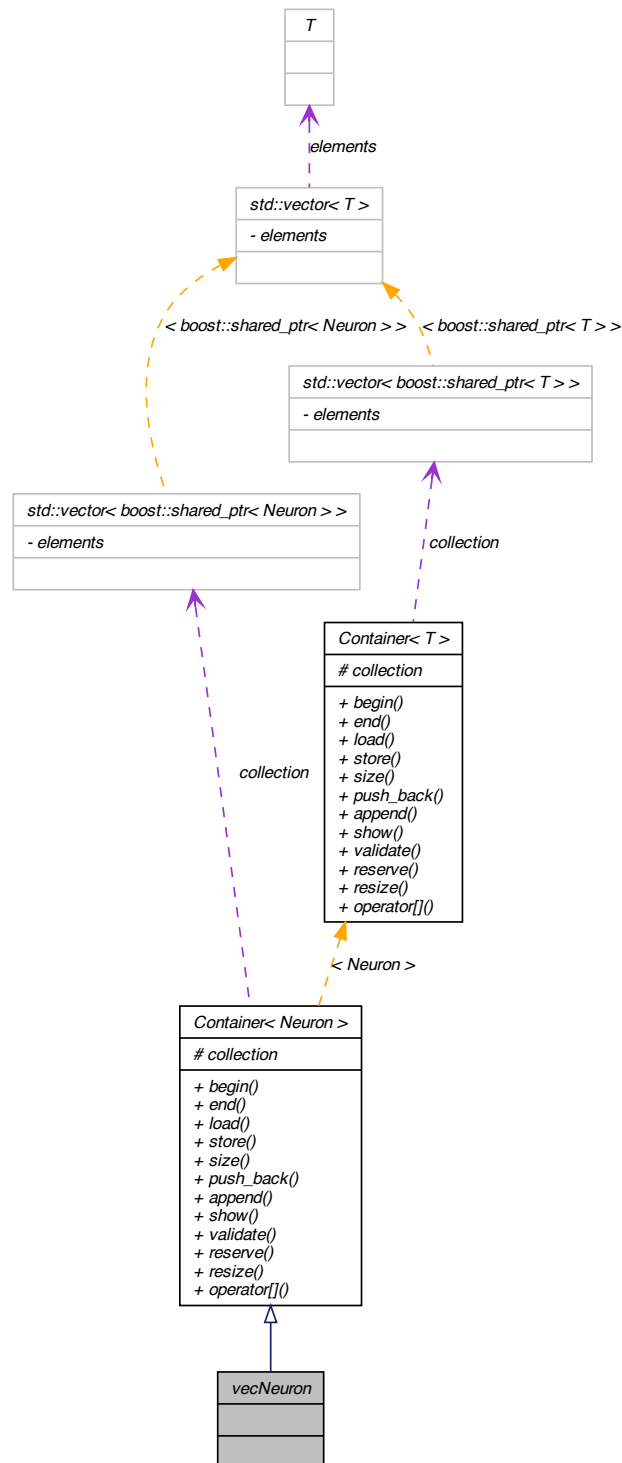
```
#include <VecNeuron.h>
```

Inheritance diagram for vecNeuron:





Collaboration diagram for vecNeuron:



### 5.7.1 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](#)

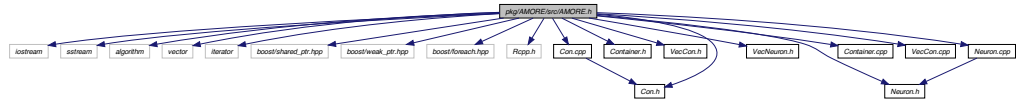
## Chapter 6

# 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 "VecCon.h"
#include "Neuron.h"
#include "VecNeuron.h"
#include "Con.cpp"
#include "Container.cpp"
#include "VecCon.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 boost::shared_ptr< Container< Con > > ContainerConPtr`
- `typedef boost::shared_ptr< Container< Neuron > > ContainerNeuronPtr`
- `typedef boost::shared_ptr< VecCon > VecConPtr`
- `typedef boost::shared_ptr< VecNeuron > VecNeuronPtr`

### 6.1.1 Define Documentation

#### 6.1.1.1 `#define` [foreach](#) BOOST\_FOREACH

Definition at line 38 of file AMORE.h.

#### 6.1.1.2 `#define` [size\\_type](#) unsigned int

Definition at line 41 of file AMORE.h.

### 6.1.2 Typedef Documentation

#### 6.1.2.1 `typedef` `boost::shared_ptr<Con> ConPtr`

Definition at line 44 of file AMORE.h.

#### 6.1.2.2 `typedef` `boost::shared_ptr< Container<Con> > ContainerConPtr`

Definition at line 47 of file AMORE.h.

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

Definition at line 48 of file AMORE.h.

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

Definition at line 45 of file AMORE.h.

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

Definition at line 46 of file AMORE.h.

6.1.2.6 `typedef boost::shared_ptr< VecCon > VecConPtr`

Definition at line 49 of file AMORE.h.

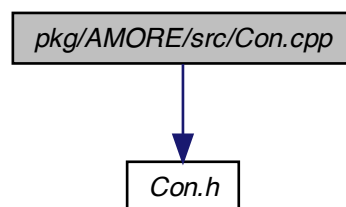
6.1.2.7 `typedef boost::shared_ptr< VecNeuron > VecNeuronPtr`

Definition at line 50 of file AMORE.h.

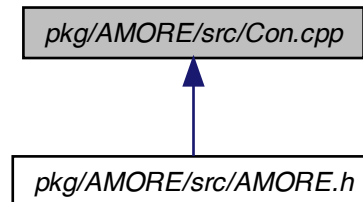
## 6.2 pkg/AMORE/src/Con.cpp File Reference

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

Include dependency graph for Con.cpp:

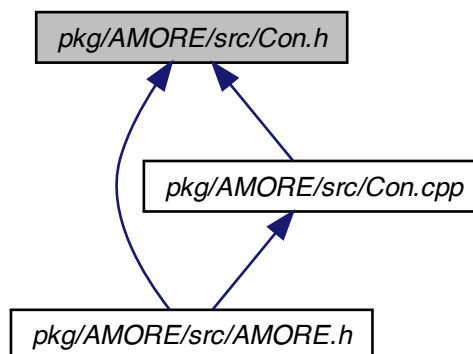


This graph shows which files directly or indirectly include this file:



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

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



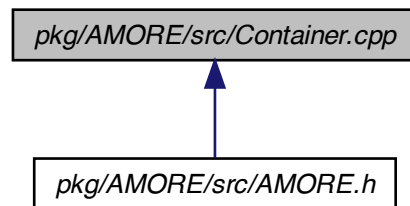
#### Classes

- class [Con](#)

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

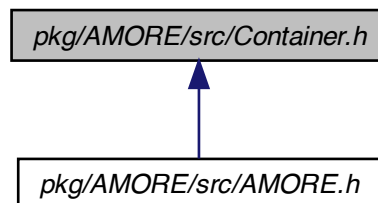
## 6.4 pkg/AMORE/src/Container.cpp File Reference

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



## 6.5 pkg/AMORE/src/Container.h File Reference

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



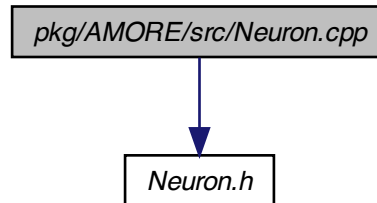
### Classes

- class `Container< T >`

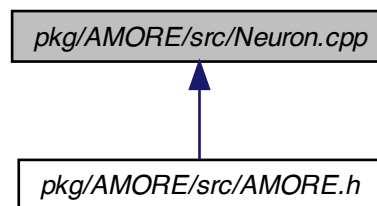
## 6.6 pkg/AMORE/src/Neuron.cpp File Reference

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

Include dependency graph for `Neuron.cpp`:



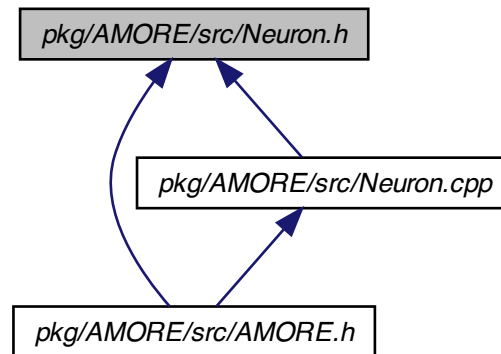
This graph shows which files directly or indirectly include this file:





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

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



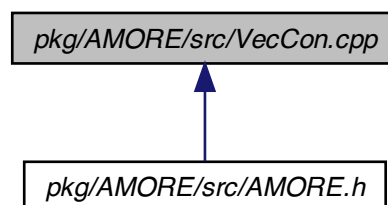
### Classes

- class [Neuron](#)

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

## 6.8 pkg/AMORE/src/VecCon.cpp File Reference

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

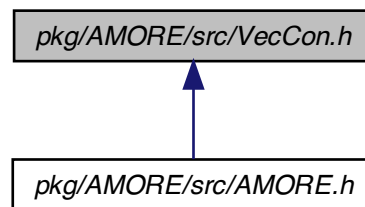


## Classes

- struct [CompareId](#)

## 6.9 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.*

## 6.10 pkg/AMORE/src/VecMLPneuron.h File Reference

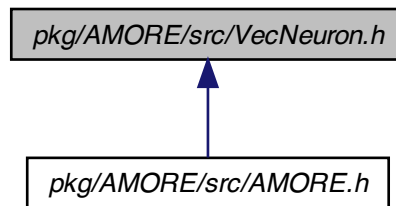
## Classes

- class [vecMLPneuron](#)

*A vector of connections.*

## 6.11 pkg/AMORE/src/VecNeuron.h File Reference

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



### Classes

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

# Index

- ~Con
  - Con, [12](#)
- ~Neuron
  - Neuron, [33](#)
- AMORE.h
  - ConPtr, [70](#)
  - ContainerConPtr, [70](#)
  - ContainerNeuronPtr, [70](#)
  - foreach, [70](#)
  - NeuronPtr, [71](#)
  - NeuronWeakPtr, [71](#)
  - size\_type, [70](#)
  - VecConPtr, [71](#)
  - VecNeuronPtr, [71](#)
- append
  - Container, [22](#)
- begin
  - Container, [23](#)
- buildAndAppend
  - VecCon, [42](#)
  - vecMLPneuron, [65](#)
- collection
  - Container, [30](#)
- CompareId, [9](#)
  - operator(), [9](#), [10](#)
- Con, [10](#)
  - ~Con, [12](#)
  - Con, [11](#)
  - from, [18](#)
  - getFrom, [12](#)
  - getId, [12](#)
  - getWeight, [13](#)
  - setFrom, [14](#)
  - setWeight, [15](#)
  - show, [16](#)
  - validate, [17](#)
  - weight, [18](#)
- con
  - Neuron, [38](#)
- ConPtr
  - AMORE.h, [70](#)
- const\_iterator
  - Container, [21](#)
- Container, [18](#)
  - append, [22](#)
  - begin, [23](#)
  - collection, [30](#)
  - const\_iterator, [21](#)
  - end, [24](#)
  - iterator, [21](#)
  - load, [24](#)
  - push\_back, [25](#)
  - reserve, [26](#)
  - resize, [26](#)
  - show, [27](#)
  - size, [28](#)
  - store, [29](#)
  - validate, [29](#)
- ContainerConPtr
  - AMORE.h, [70](#)
- ContainerNeuronPtr
  - AMORE.h, [70](#)
- end
  - Container, [24](#)
- erase
  - VecCon, [44](#)
- foreach
  - AMORE.h, [70](#)
- from
  - Con, [18](#)
- getConId
  - Neuron, [33](#)
- getFrom
  - Con, [12](#)
  - Neuron, [33](#)
  - VecCon, [46](#)

- getId
  - Con, [12](#)
  - Neuron, [34](#)
  - VecCon, [47](#)
- getWeight
  - Con, [13](#)
  - Neuron, [34](#)
  - VecCon, [49](#), [51](#)
- Id
  - Neuron, [38](#)
- iterator
  - Container, [21](#)
- load
  - Container, [24](#)
- Neuron, [30](#)
  - ~Neuron, [33](#)
  - con, [38](#)
  - getId, [33](#)
  - getFrom, [33](#)
  - getId, [34](#)
  - getWeight, [34](#)
  - Id, [38](#)
  - Neuron, [32](#), [33](#)
  - numOfCons, [35](#)
  - outputValue, [38](#)
  - setFrom, [35](#)
  - setId, [36](#)
  - setWeight, [36](#)
  - show, [37](#)
  - validate, [37](#)
- NeuronPtr
  - AMORE.h, [71](#)
- NeuronWeakPtr
  - AMORE.h, [71](#)
- numOfCons
  - Neuron, [35](#)
  - VecCon, [52](#)
- operator()
  - CompareId, [9](#), [10](#)
- outputValue
  - Neuron, [38](#)
- pkg/AMORE/src/AMORE.h, [69](#)
- pkg/AMORE/src/Con.cpp, [71](#)
- pkg/AMORE/src/Con.h, [72](#)
- pkg/AMORE/src/Container.cpp, [73](#)
- pkg/AMORE/src/Container.h, [73](#)
- pkg/AMORE/src/Neuron.cpp, [73](#)
- pkg/AMORE/src/Neuron.h, [75](#)
- pkg/AMORE/src/VecCon.cpp, [75](#)
- pkg/AMORE/src/VecCon.h, [76](#)
- pkg/AMORE/src/VecMLPneuron.h, [76](#)
- pkg/AMORE/src/VecNeuron.h, [77](#)
- push\_back
  - Container, [25](#)
- reserve
  - Container, [26](#)
- resize
  - Container, [26](#)
- select
  - VecCon, [53](#)
- setFrom
  - Con, [14](#)
  - Neuron, [35](#)
  - VecCon, [55](#)
- setId
  - Neuron, [36](#)
- setWeight
  - Con, [15](#)
  - Neuron, [36](#)
  - VecCon, [57](#), [59](#)
- show
  - Con, [16](#)
  - Container, [27](#)
  - Neuron, [37](#)
- size
  - Container, [28](#)
- size\_type
  - AMORE.h, [70](#)
- store
  - Container, [29](#)
- validate
  - Con, [17](#)
  - Container, [29](#)
  - Neuron, [37](#)
  - VecCon, [61](#)
- VecCon, [39](#)
  - buildAndAppend, [42](#)
  - erase, [44](#)
  - getFrom, [46](#)
  - getId, [47](#)
  - getWeight, [49](#), [51](#)
  - numOfCons, [52](#)
  - select, [53](#)

- setFrom, [55](#)
  - setWeight, [57](#), [59](#)
  - validate, [61](#)
- VecConPtr
  - AMORE.h, [71](#)
- vecMLPneuron, [62](#)
  - buildAndAppend, [65](#)
- vecNeuron, [65](#)
- VecNeuronPtr
  - AMORE.h, [71](#)
- weight
  - Con, [18](#)