

AMORE++

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

Generated by Doxygen 1.7.4

Thu Jun 2 2011 19:25:59

Contents

1	The AMORE++ package	1
1.1	Introduction	1
1.2	Motivation	1
1.3	Road Map	1
2	Todo List	3
3	Class Index	5
3.1	Class Hierarchy	5
4	Class Index	7
4.1	Class List	7
5	File Index	9
5.1	File List	9
6	Class Documentation	11
6.1	Con Class Reference	11
6.1.1	Detailed Description	13
6.1.2	Constructor & Destructor Documentation	13
6.1.2.1	Con	13
6.1.2.2	Con	13
6.1.2.3	~Con	14
6.1.3	Member Function Documentation	14
6.1.3.1	getFromId	14
6.1.3.2	getFromNeuron	15
6.1.3.3	getWeight	16

6.1.3.4	setFromNeuron	17
6.1.3.5	setWeight	17
6.1.3.6	show	18
6.1.3.7	validate	19
6.1.4	Member Data Documentation	20
6.1.4.1	from	20
6.1.4.2	weight	20
6.2	Neuron Class Reference	20
6.2.1	Detailed Description	21
6.2.2	Member Function Documentation	21
6.2.2.1	getId	21
6.2.2.2	setId	22
6.2.3	Member Data Documentation	22
6.2.3.1	Id	22
6.2.3.2	outputValue	22
6.3	vecAMORE< T > Class Template Reference	22
6.3.1	Detailed Description	25
6.3.2	Member Function Documentation	25
6.3.2.1	append	25
6.3.2.2	getLdata	27
6.3.2.3	push_back	28
6.3.2.4	setLdata	29
6.3.2.5	show	29
6.3.2.6	size	30
6.3.2.7	validate	30
6.3.3	Member Data Documentation	31
6.3.3.1	ldata	31
6.4	vecCon Class Reference	31
6.4.1	Detailed Description	34
6.4.2	Member Function Documentation	34
6.4.2.1	getFromId	34
6.4.2.2	numOfCons	35
6.4.2.3	populate	36

7	File Documentation	37
7.1	pkg/AMORE/src/AMORE.h File Reference	37
7.1.1	Typedef Documentation	38
7.1.1.1	ConSharedPtr	38
7.1.1.2	NeuronSharedPtr	38
7.2	pkg/AMORE/src/Con.cpp File Reference	38
7.3	pkg/AMORE/src/Con.h File Reference	39
7.4	pkg/AMORE/src/Neuron.cpp File Reference	40
7.5	pkg/AMORE/src/Neuron.h File Reference	41
7.6	pkg/AMORE/src/vecAMORE.cpp File Reference	41
7.7	pkg/AMORE/src/vecAMORE.h File Reference	42
7.8	pkg/AMORE/src/vecCon.cpp File Reference	42
7.9	pkg/AMORE/src/vecCon.h File Reference	43

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:

Con	11
Neuron	20
vecAMORE< T >	22
vecAMORE< Con >	22
vecCon	31

Chapter 4

Class Index

4.1 Class List

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

Con (A class to handle the information needed to describe an input connection)	11
Neuron (A class to handle the information contained in a general Neuron)	20
vecAMORE< T >	22
vecCon (A vector of connections)	31

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

pkg/AMORE/src/ AMORE.h	37
pkg/AMORE/src/ Con.cpp	38
pkg/AMORE/src/ Con.h	39
pkg/AMORE/src/ Neuron.cpp	40
pkg/AMORE/src/ Neuron.h	41
pkg/AMORE/src/ vecAMORE.cpp	41
pkg/AMORE/src/ vecAMORE.h	42
pkg/AMORE/src/ vecCon.cpp	42
pkg/AMORE/src/ vecCon.h	43

Chapter 6

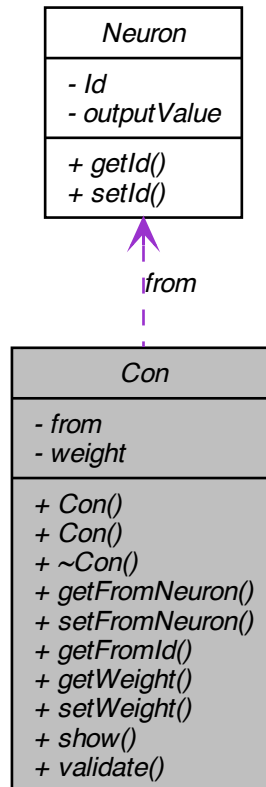
Class Documentation

6.1 Con Class Reference

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

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

Collaboration diagram for Con:



Public Member Functions

- [Con](#) ()
Default Constructor.
- [Con](#) ([Neuron](#) *f, double w)
Constructor.
- [~Con](#) ()
Default Destructor.
- [Neuron](#) * [getFromNeuron](#) ()
from field accessor.
- void [setFromNeuron](#) ([Neuron](#) *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

- [Neuron](#) * `from`
A pointer to the [Neuron](#) used as input during simulation or training.
- double `weight`
A double variable that contains the weight of the connection.

6.1.1 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 the [Neuron](#) 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`.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 `Con::Con ()`

Default Constructor.

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

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

6.1.2.2 `Con::Con (Neuron * f, double w)`

Constructor.

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

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

6.1.2.3 Con::~Con ()

Default Destructor.

Definition at line 32 of file Con.cpp.

```
{};
```

6.1.3 Member Function Documentation

6.1.3.1 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
Con myCon;
Neuron MyNeuron;
MyNeuron.setId(16);
myCon.setFromNeuron (&MyNeuron);

// Test
int result= myCon.getFromId();
// After execution of the code shown above, MyNeuron::Id is set to the in
teger value 16 and, thus, 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 97 of file Con.cpp.

References from, and `Neuron::getId()`.

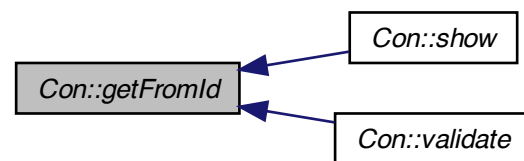
Referenced by `show()`, and `validate()`.

```
    {
    return(from->getId() );
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3.2 Neuron * 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
Con myCon;
Neuron MyNeuron;
Neuron* ptNeuron;
MyNeuron.setId(1);
myCon.setFromNeuron(&MyNeuron);

//Test
ptNeuron = myCon.getFromNeuron();
  
```

```
int result= ptNeuron->getId();
// Now, ptNeuron is pointing at MyNeuron and, thus, 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 59 of file `Con.cpp`.

References from.

```

{
    return(from);
}
```

6.1.3.3 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
Con myCon;
Neuron MyNeuron;
MyNeuron.setId(16);
myCon.setFromNeuron(&MyNeuron);
myCon.setWeight(12.4);
double result1= myCon.getWeight();

// Test
myCon.setWeight(2.2);
double result2= myCon.getWeight();
// Now, result1 is equal to 12.4 and result2 is equal to 2.2.
```

See also

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

Definition at line 127 of file `Con.cpp`.

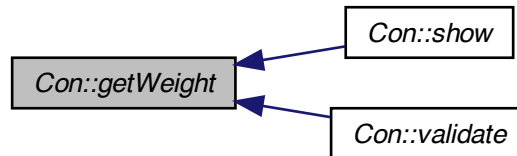
References weight.

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

```

{
    return(weight);
}
```

Here is the caller graph for this function:



6.1.3.4 void Con::setFromNeuron (Neuron * 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 from field.
----------	---

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 70 of file `Con.cpp`.

References from.

```

{
    from = f;
}

```

6.1.3.5 void Con::setWeight (double w)

weight field accessor.

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

Parameters

<i>w</i>	The new value (double) to be set in the weight field.
----------	---

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

```
//Usage example:
//=====
// Data set up
        Con myCon;
        Neuron n;
        n.setId(16);
        myCon.setFromNeuron(&n);

// Test
        myCon.setWeight(12.4);
        myCon.show();
// Now, the output at the R terminal would show:
//
// FROM=16           WEIGHT=12.4
//
```

See also

[getWeight](#) and the unit test files (e.g. `runit.Cpp.Con.R`)

Definition at line 159 of file `Con.cpp`.

References `weight`.

```

                                {
        weight = w;
}

```

6.1.3.6 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 171 of file `Con.cpp`.

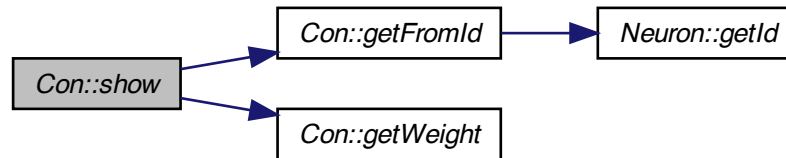
References `getFromId()`, and `getWeight()`.

```

        {
        Rprintf("From:\t %d \t Weight= \t %lf \n", getFromId() , getWeight());
        return(true);
}

```


Here is the call graph for this function:



6.1.3.7 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 185 of file Con.cpp.

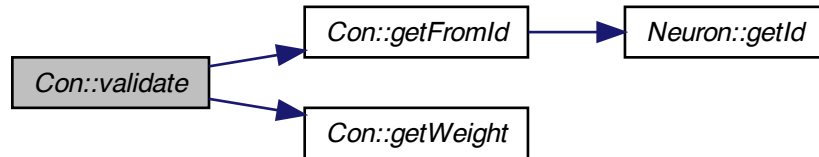
References `getFromId()`, and `getWeight()`.

```

    {
        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:



6.1.4 Member Data Documentation

6.1.4.1 `Neuron* Con::from` [private]

A 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()`.

6.1.4.2 `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`

6.2 Neuron Class Reference

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

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

Public Member Functions

- int `getId` ()
- void `setId` (int id)

Private Attributes

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

6.2.1 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.

6.2.2 Member Function Documentation

6.2.2.1 int Neuron::getId ()

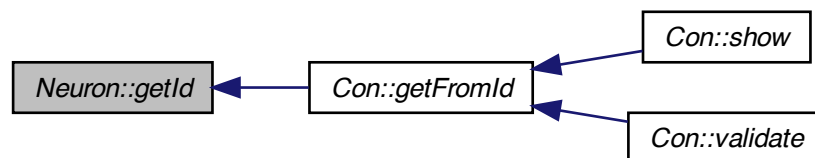
Definition at line 15 of file Neuron.cpp.

References `Id`.

Referenced by `Con::getFromId()`.

```
{  
    return Id;  
}
```

Here is the caller graph for this function:



6.2.2.2 void Neuron::setId (int *id*)

Definition at line 19 of file Neuron.cpp.

References `Id`.

```

        {
            Id=id;
        }

```

6.2.3 Member Data Documentation

6.2.3.1 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()`.

6.2.3.2 double Neuron::outputValue [private]

A vector of input connections.

Todo

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

Definition at line 30 of file Neuron.h.

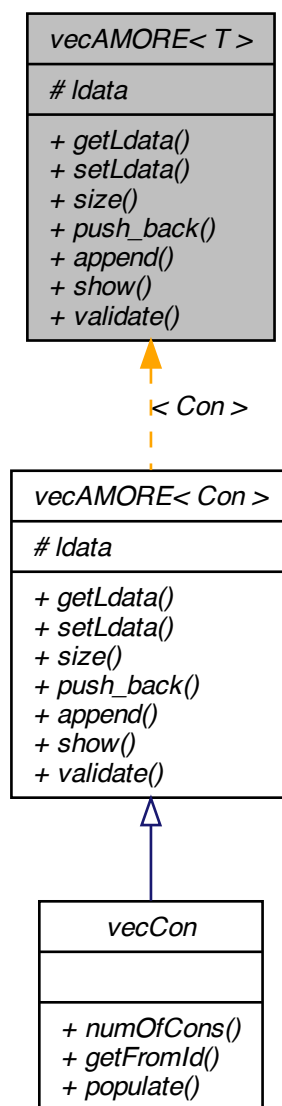
The documentation for this class was generated from the following files:

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

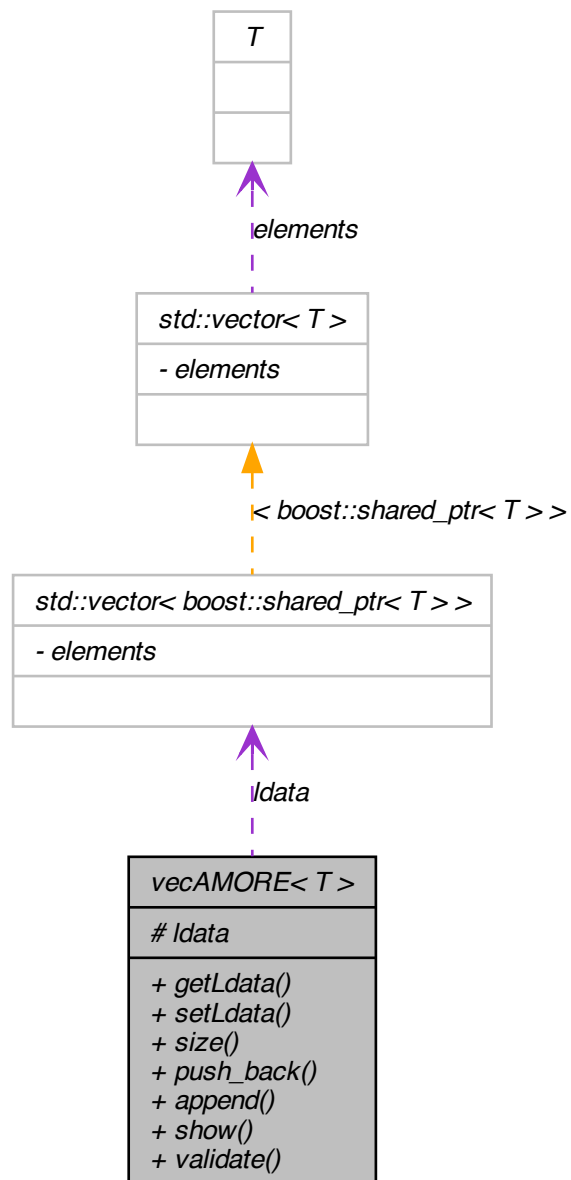
6.3 vecAMORE< T > Class Template Reference

```
#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.

Protected Attributes

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

6.3.1 Detailed Description

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

Definition at line 12 of file vecAMORE.h.

6.3.2 Member Function Documentation

6.3.2.1 `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
Neuron N1, N2, N3, N4, N5, N6;
vecAMORE<Con> vcA, vcB;
std::vector<int> result;

N1.setId(10);
N2.setId(20);
N3.setId(30);
N4.setId(40);
N5.setId(50);
N6.setId(60);

// Test
ConSharedPtr ptCon( new Con(&N1, 1.13) );           // Create and sto
re in vcA three Cons
vcA.push_back(ptCon);
ptCon.reset( new Con(&N2, 2.22) );
vcA.push_back(ptCon);
ptCon.reset( new Con(&N3, 3.33) );
vcA.push_back(ptCon);

ptCon.reset( new Con(&N4, 1.13) );           // Create and store in vc
B three more Cons
vcB.push_back(ptCon);
ptCon.reset( new Con(&N5, 2.22) );
vcB.push_back(ptCon);
ptCon.reset( new Con(&N6, 3.33) );
vcB.push_back(ptCon);

// Append test
vcA.append(vcB);
vcA.validate();
vcA.show() ;

// After execution of the code above, the output at the R terminal would
display:
// From:      10      Weight=      1.130000
// From:      20      Weight=      2.220000
// From:      30      Weight=      3.330000
// From:      40      Weight=      1.130000
// From:      50      Weight=      2.220000
// From:      60      Weight=      3.330000

```

See also

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

Definition at line 133 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:



6.3.2.2 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
Neuron N1, N2, N3;
vecAMORE<Con> MyvecCon;
std::vector<int> result;
std::vector<ConSharedPtr> vcA, vcB;

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

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

MyvecCon.setLdata(vcA);
vcB = MyvecCon.getLdata();

result.push_back(vcB.at(0)->getFromId());
result.push_back(vcB.at(1)->getFromId());
  
```

```

        result.push_back(vcB.at(2)->getFromId());
        // After execution of the code shown above, 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 177 of file `vecAMORE.cpp`.

```

};

        return ldata;
    }

```

6.3.2.3 `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 values 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 45 of file `vecAMORE.cpp`.

```

{
    this->ldata.push_back(TsharedPtr);
};

```

6.3.2.4 `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 189 of file `vecAMORE.cpp`.

```

{
    ldata=v;
};

```

6.3.2.5 `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

See also

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

Definition at line 56 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.begin();
        itr != ldata.end(); itr++) { (*itr)->show(); }
    return true;
};

```

6.3.2.6 `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 200 of file `vecAMORE.cpp`.

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

```

{
    return ldata.size() ;
};

```

Here is the caller graph for this function:



6.3.2.7 `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 71 of file `vecAMORE.cpp`.

```

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

```

6.3.3 Member Data Documentation

6.3.3.1 `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:

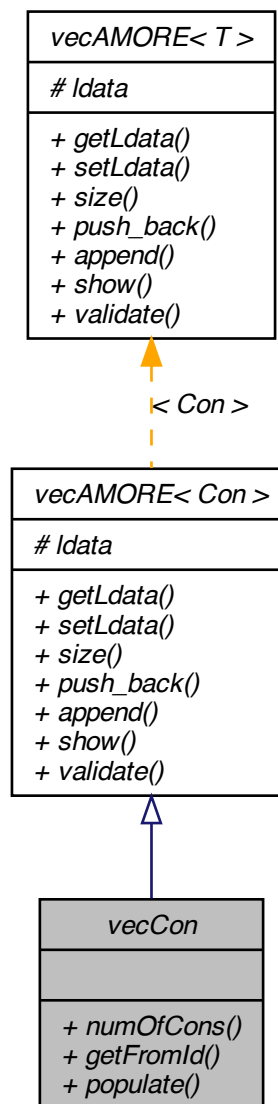
- `pkg/AMORE/src/vecAMORE.h`
- `pkg/AMORE/src/vecAMORE.cpp`

6.4 vecCon Class Reference

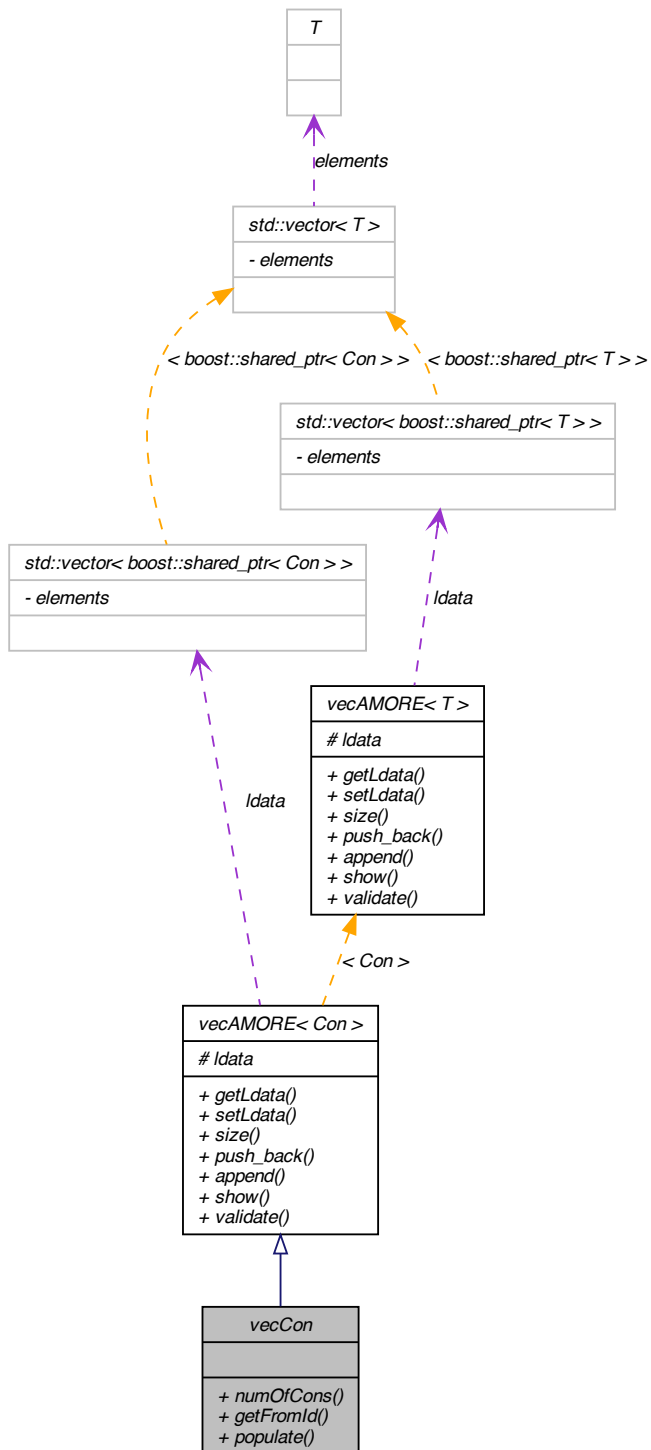
A vector of connections.

```
#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.
- void `populate` (`std::vector< NeuronSharedPtr >` FROM, `std::vector< double >` WEIGHT)

6.4.1 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`.

6.4.2 Member Function Documentation

6.4.2.1 `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
// 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.

```

Definition at line 85 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:



6.4.2.2 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
Neuron N1, N2, N3;
vecCon MyvecCon;
std::vector<int> result;

N1.setId(10);

```

```

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

        // Test
        result.push_back(MyvecCon.numOfCons()); // Append
        numOfCons to result, create new Con and push_back into MyvecCon
        ConSharedPtr ptCon( new Con(&N1, 1.13) ); // and re
peat twice

        MyvecCon.push_back(ptCon);
        result.push_back(MyvecCon.numOfCons());

        ptCon.reset( new Con(&N2, 2.22) );
        MyvecCon.push_back(ptCon);
        result.push_back(MyvecCon.numOfCons());

        ptCon.reset( new Con(&N3, 3.33) );
        MyvecCon.push_back(ptCon);
        result.push_back(MyvecCon.numOfCons());

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

```

See also

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

Definition at line 45 of file `vecCon.cpp`.

References `vecAMORE< Con >::ldata`.

Referenced by `getFromId()`.

```

        {
        return ldata.size();
        }

```

Here is the caller graph for this function:



6.4.2.3 `void vecCon::populate (std::vector< NeuronSharedPtr > FROM, std::vector< double > WEIGHT)`

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

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

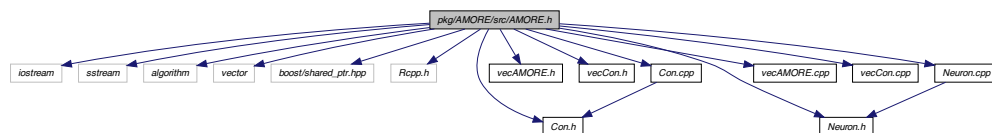
Chapter 7

File Documentation

7.1 pkg/AMORE/src/AMORE.h File Reference

```
#include <iostream>
#include <sstream>
#include <algorithm>
#include <vector>
#include <boost/shared_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`

7.1.1 Typedef Documentation

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

Definition at line 31 of file AMORE.h.

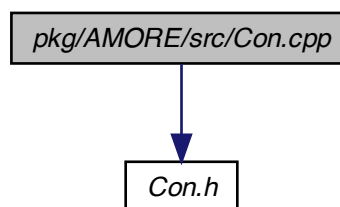
7.1.1.2 `typedef boost::shared_ptr<Neuron> NeuronSharedPtr`

Definition at line 35 of file AMORE.h.

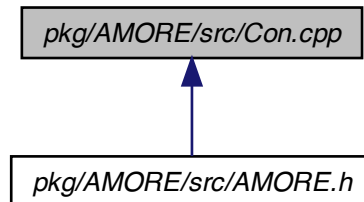
7.2 pkg/AMORE/src/Con.cpp File Reference

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

Include dependency graph for Con.cpp:

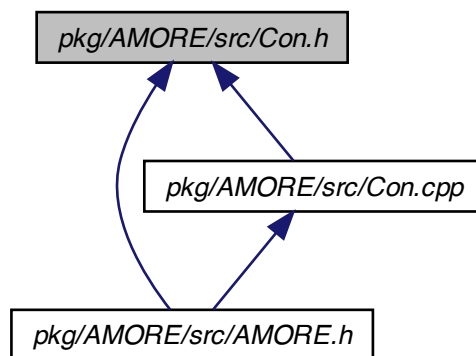


This graph shows which files directly or indirectly include this file:



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

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



Classes

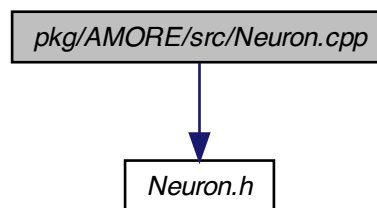
- class [Con](#)

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

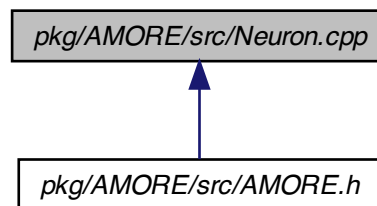
7.4 pkg/AMORE/src/Neuron.cpp File Reference

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

Include dependency graph for Neuron.cpp:

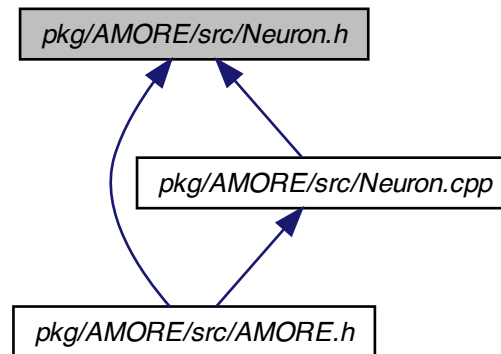


This graph shows which files directly or indirectly include this file:



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

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



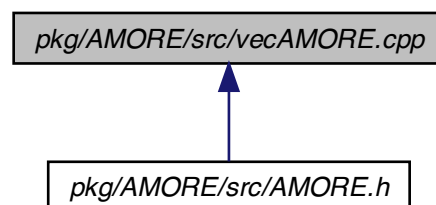
Classes

- class `Neuron`

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

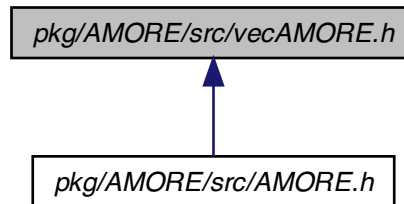
7.6 pkg/AMORE/src/vecAMORE.cpp File Reference

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:

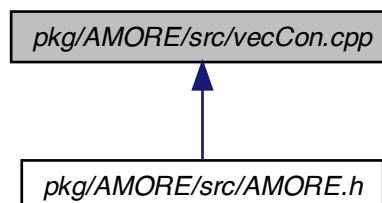


Classes

- class `vecAMORE< T >`

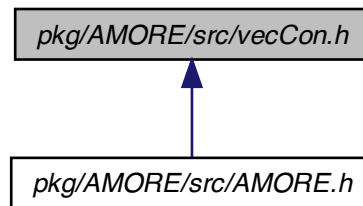
7.8 pkg/AMORE/src/vecCon.cpp File Reference

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



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.

Index

~Con
Con, [13](#)

AMORE.h
ConSharedPtr, [38](#)
NeuronSharedPtr, [38](#)

append
vecAMORE, [25](#)

Con, [11](#)
~Con, [13](#)
Con, [13](#)
from, [20](#)
getFromId, [14](#)
getFromNeuron, [15](#)
getWeight, [16](#)
setFromNeuron, [17](#)
setWeight, [17](#)
show, [18](#)
validate, [19](#)
weight, [20](#)

ConSharedPtr
AMORE.h, [38](#)

from
Con, [20](#)

getFromId
Con, [14](#)
vecCon, [34](#)

getFromNeuron
Con, [15](#)

getId
Neuron, [21](#)

getLdata
vecAMORE, [27](#)

getWeight
Con, [16](#)

Id
Neuron, [22](#)

ldata
vecAMORE, [31](#)

Neuron, [20](#)
getId, [21](#)
Id, [22](#)
outputValue, [22](#)
setId, [22](#)

NeuronSharedPtr
AMORE.h, [38](#)

numOfCons
vecCon, [35](#)

outputValue
Neuron, [22](#)

pkg/AMORE/src/AMORE.h, [37](#)
pkg/AMORE/src/Con.cpp, [38](#)
pkg/AMORE/src/Con.h, [39](#)
pkg/AMORE/src/Neuron.cpp, [40](#)
pkg/AMORE/src/Neuron.h, [41](#)
pkg/AMORE/src/vecAMORE.cpp, [41](#)
pkg/AMORE/src/vecAMORE.h, [42](#)
pkg/AMORE/src/vecCon.cpp, [42](#)
pkg/AMORE/src/vecCon.h, [43](#)
populate
vecCon, [36](#)
push_back
vecAMORE, [28](#)

setFromNeuron
Con, [17](#)

setId
Neuron, [22](#)

setLdata
vecAMORE, [29](#)

setWeight
Con, [17](#)

show
Con, [18](#)
vecAMORE, [29](#)

size

- vecAMORE, [30](#)
- validate
 - Con, [19](#)
 - vecAMORE, [30](#)
- vecAMORE, [22](#)
 - append, [25](#)
 - getLdata, [27](#)
 - ldata, [31](#)
 - push_back, [28](#)
 - setLdata, [29](#)
 - show, [29](#)
 - size, [30](#)
 - validate, [30](#)
- vecCon, [31](#)
 - getFromId, [34](#)
 - numOfCons, [35](#)
 - populate, [36](#)
- weight
 - Con, [20](#)