AMORE++

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

Generated by Doxygen 1.7.4

Sat May 28 2011 12:49:27

Contents

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::vecCon restore vecCon<Con> listCon;

Chapter 3

Class Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

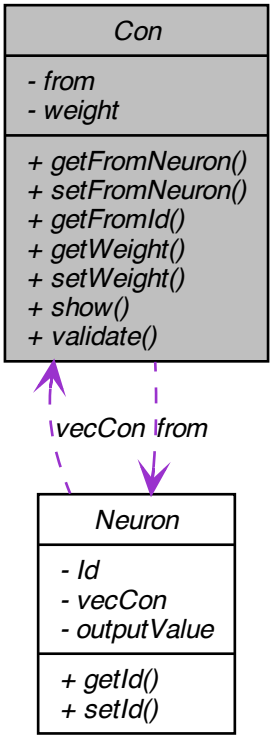
Class Documentation

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

- 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 Member Function Documentation

6.1.2.1 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:
//================
      Con myCon;
      Neuron MyNeuron;
      MyNeuron.setId(16);
      myCon.setFromNeuron(&MyNeuron);
      int result= myCon.getFromId();
```

After execution of the code shown above, MyNeuron::Id is set to the integer 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 73 of file Con.cpp.

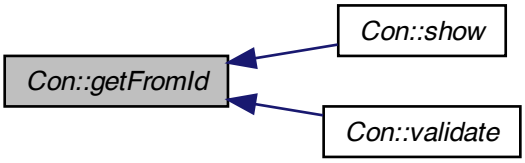
References from, and Neuron::getId().

Referenced by show(), and validate().

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

Here is the call graph for this function:

Here is the caller graph for this function:



6.1.2.2 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:
//================
      Con myCon;
      Neuron MyNeuron;
      Neuron * ptNeuron;
      MyNeuron.setId(1);
      myCon.setFromNeuron(&MyNeuron);
      ptNeuron = myCon.getFromNeuron();
      int result= ptNeuron->getId();
```

After execution of the code shown above, 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 37 of file Con.cpp.

References from.

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

6.1.2.3 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:
//================
      Con myCon;
      Neuron MyNeuron;
      MyNeuron.setId(16);
      myCon.setFromNeuron(&MyNeuron);
      myCon.setWeight(12.4);
      double result1= myCon.getWeight();
      myCon.setWeight(2.2);
      double result2= myCon.getWeight();
```

After execution of the code shown above, result1 is set to the double value 12.4 and result2 is set to the double value 2.2.

See also

setWeight and the unit test files, e.g., runit.Cpp.Con.R, for further examples.

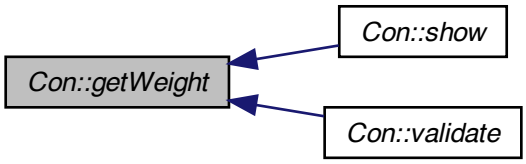
Definition at line 103 of file Con.cpp.

References weight.

Referenced by show(), and validate().

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

Here is the caller graph for this function:



6.1.2.4 void Con::setFromNeuron (Neuron * f)

from field accessor.

This method sets the value of the from field with the address used as parameter.

Parameters

f	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 48 of file Con.cpp.

References from.

```
                                                {
        from = f;
}
```

6.1.2.5 void Con::setWeight (double *w*)

weight field accessor.

This method sets the value of the weight field.

Parameters

w	The new value (double) to be set in the weight field.

```
//================
//Usage example:
//================
      Con myCon;
      Neuron n;
      n.setId(16);
      myCon.setFromNeuron(&n);
      myCon.setWeight(12.4);
      myCon.show();
```

After execution of the code shown above, 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 134 of file Con.cpp.

References weight.

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

6.1.2.6 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

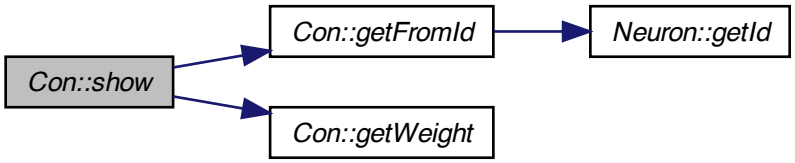
setWeight and the unit test files, e.g., runit.Cpp.Con.R, for usage examples.

Definition at line 146 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:



6.1.2.7 bool Con::validate ( )

Object validator.

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

Returns

true in case the checks are Ok.

Exceptions

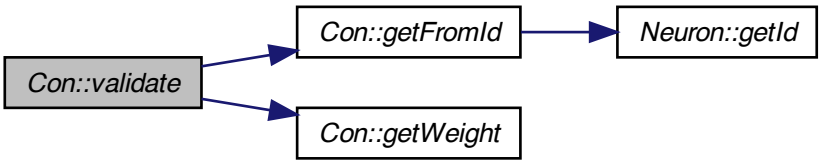
An	std::range error if weight or from are not finite.

Definition at line 160 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:



6.1.3 Member Data Documentation

6.1.3.1 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().

6.1.3.2 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:

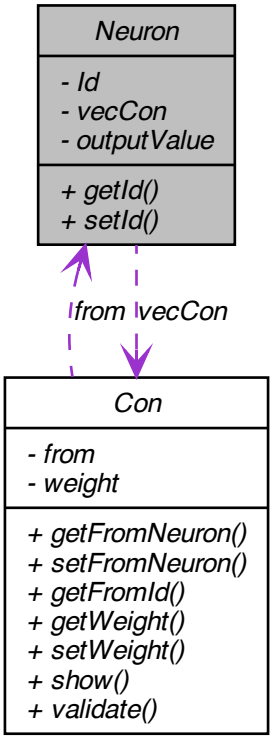
- pkg/AMORE/src/Con.h
- pkg/AMORE/src/Con.cpp

6.2 Neuron Class Reference

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

`#include <Neuron.h>`

Collaboration diagram for Neuron:



Public Member Functions

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

Private Attributes

- int Id

  An integer variable with the Neuron Id.
- Con vecCon

  A vector of input connections.
- double outputValue

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]

Definition at line 30 of file Neuron.h.

6.2.3.3 Con Neuron::vecCon [private]

A vector of input connections.

Todo

restore vecCon<Con> listCon;

Definition at line 29 of file Neuron.h.

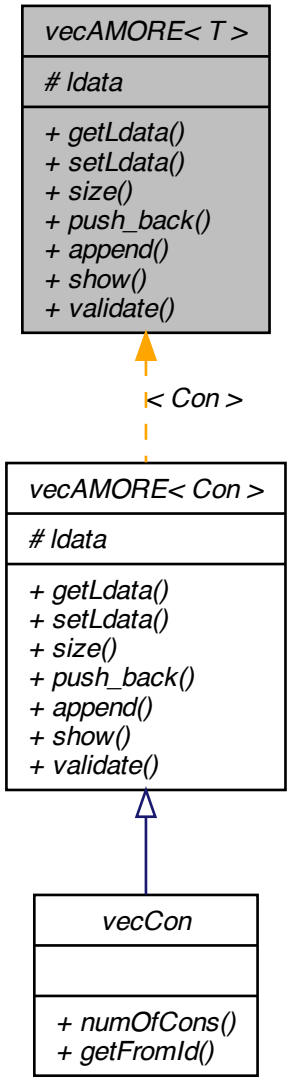
The documentation for this class was generated from the following files:

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

6.3 vecAMORE< T > Class Template Reference

`#include <vecAMORE.h>`

Inheritance diagram for vecAMORE< T >:



Public Member Functions

- std::vector< T > getLdata ()

  ldata field accessor function

- void setLdata (typename std::vector< T >)

  ldata field accessor function
- int size ()

  Returns the size or length of the vector.
- void push_back (T element)

  A method to append one element 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< T > ldata

6.3.1 Detailed Description

template<typename T>class vecAMORE< T >

Definition at line 11 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

v	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.

Definition at line 108 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< T > vecAMORE< T >::getLdata ( )

ldata field accessor function

This method allows access to the data stored in the ldata field (an std::vector<T> object).

Returns

The ldata vector.

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

// Data set up
        Con Con1, Con2, Con3;
        Neuron N1, N2, N3;
        vecAMORE<Con> MyvecCon;
        std::vector<int> result;
        std::vector<Con> vc;

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

        Con1.setFromNeuron(&N1);
        Con2.setFromNeuron(&N2);
        Con3.setFromNeuron(&N3);

        Con1.setWeight(1.01);
        Con2.setWeight(22.02);
        Con3.setWeight(333.03);

// Test

        MyvecCon.push_back(Con1);
        MyvecCon.push_back(Con2);
        MyvecCon.push_back(Con3);
        MyvecCon.show();
        MyvecCon.validate();

        vc = MyvecCon.getLdata();
```

```
result.push_back(vc.at(0).getFromId());
result.push_back(vc.at(1).getFromId());
result.push_back(vc.at(2).getFromId());
return wrap(result);
```

After execution of the code shown above, vc is a vector containing Con1, Con2 and Con3 and, thus, 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 164 of file vecAMORE.cpp.

```
                                                                {
        return ldata;
};
```

6.3.2.3 template<typename T> void vecAMORE< T >::push_back (T *element*)

A method to append one element at the end of ldata.

This function implements the std:vector member push_back for the vecAMORE<T> class

Parameters

T	element The element to be inserted at the end of ldata

```
//================
//Usage example:
//================
Con Con1, Con2, Con3, Con4, Con5, Con6;
Neuron N1, N2, N3, N4, N5, N6;
vecAMORE<Con> vc1, vc2;
std::vector<int> result;

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

Con1.setFromNeuron(&N1);
Con2.setFromNeuron(&N2);
Con3.setFromNeuron(&N3);
Con4.setFromNeuron(&N4);
Con5.setFromNeuron(&N5);
Con6.setFromNeuron(&N6);

Con1.setWeight(1.01);
Con2.setWeight(22.02);
Con3.setWeight(333.03);
Con4.setWeight(5.4);
Con5.setWeight(2.22);
```

```
                Con6.setWeight(33.03);

                vc1.push_back(Con1);
                vc1.push_back(Con2);
                vc1.push_back(Con3);
                vc2.push_back(Con4);
                vc2.push_back(Con5);
                vc2.push_back(Con6);

                Rprintf("vc1 contents:");
                vc1.show() ;
                Rprintf("vc2 contents:");
                vc2.show() ;

                vc1.join(vc2);
                Rprintf("vc2 contents:");
                vc1.show() ;
                vc1.validate();

                for(std::vector<Con>::iterator itr = (vc1.getLdata()).begin();
      itr != (vc1.getLdata()).end();   itr++) { result.push_back(itr->getFromId()); }
                return wrap(result);
```

After execution of this code, return contains a numeric vector with values 10, 20, 30, 40, 50 and 60.

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 69 of file vecAMORE.cpp.

```
                                                                {
        this->ldata.push_back(element);
};
```

6.3.2.4 template<typename T> void vecAMORE< T >::setLdata (typename std::vector< T > *v*)

ldata field accessor function

This method sets the value of the data stored in the ldata field (an std::vector<T> object).

Parameters

v	The std::vector<T> object to be stored in the ldata field

See also

getLdata and the unit test files, e.g., runit.Cpp.vecAMORE.R, for usage examples.

Definition at line 176 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 the vecAMORE<T> fields.

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 80 of file vecAMORE.cpp.

```
                                          {
        //      for_each(ldata.begin(), ldata.end(), showCon );
        for(typename std::vector<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 vecAMORE<T> derived classes this is aliased as numOfCons, numOfNeurons and numOfLayers. The unit test files, e.g., runit.Cpp.vecAMORE.R, for usage examples.

Definition at line 187 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 of the ldata std::vector<T>,

See also

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

Definition at line 94 of file vecAMORE.cpp.

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

6.3.3 Member Data Documentation

6.3.3.1 template<typename T> std::vector<T> vecAMORE< T >::ldata [protected]

Definition at line 13 of file vecAMORE.h.

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

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

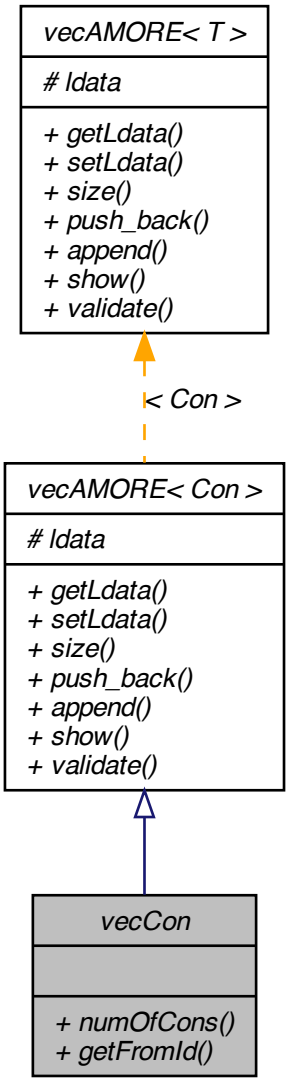
- pkg/AMORE/src/vecAMORE.h
- pkg/AMORE/src/vecAMORE.cpp

6.4 vecCon Class Reference

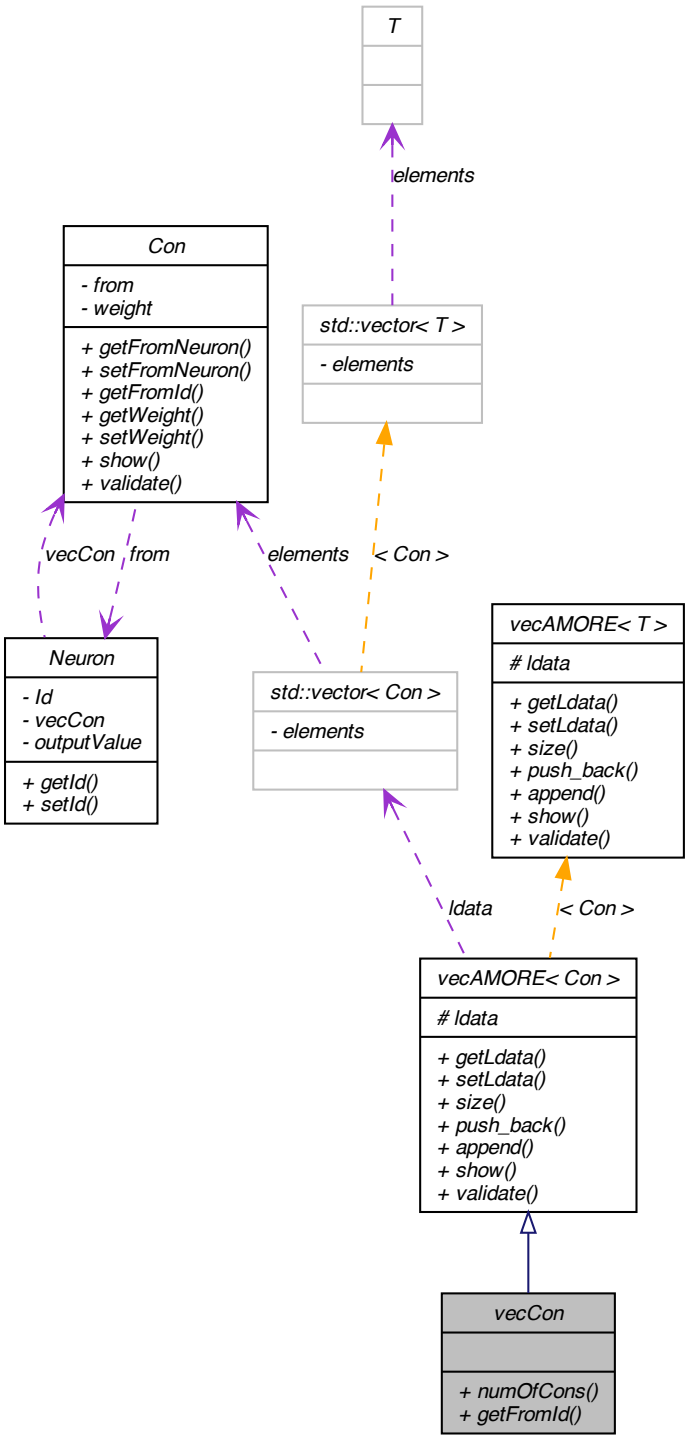
A vector of connections.

`#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.

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
              Con Con1, Con2, Con3;
              Neuron N1, N2, N3;
              vecCon MyvecCon;
              std::vector<int> result;

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

              Con1.setFromNeuron(&N1);
              Con2.setFromNeuron(&N2);
              Con3.setFromNeuron(&N3);

              Con1.setWeight(1.01);
              Con2.setWeight(22.02);
              Con3.setWeight(333.03);

              MyvecCon.push_back(Con1);
              MyvecCon.push_back(Con2);
              MyvecCon.push_back(Con3);

              MyvecCon.show() ;
              MyvecCon.validate();
```

```
        // Test
                result=MyvecCon.getFromId();
```

After execution of this code, result is a vector that contains the values 10, 20 and 30.

Definition at line 99 of file vecCon.cpp.

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

```
                                {
        std::vector<int> result;
        result.reserve(numOfCons());
        for(std::vector<Con>::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
              Con Con1, Con2, Con3;
              Neuron N1, N2, N3;
              vecCon MyvecCon;
              std::vector<int> result;

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

```
        Con1.setFromNeuron(&N1);
        Con2.setFromNeuron(&N2);
        Con3.setFromNeuron(&N3);

        Con1.setWeight(1.01);
        Con2.setWeight(22.02);
        Con3.setWeight(333.03);

// Test
        result.push_back(MyvecCon.numOfCons());
        MyvecCon.push_back(Con1);
        result.push_back(MyvecCon.numOfCons());
        MyvecCon.push_back(Con2);
        result.push_back(MyvecCon.numOfCons());
        MyvecCon.push_back(Con3);
        result.push_back(MyvecCon.numOfCons());
```

After execution of this code, result contains a numeric vector with values 0, 1, 2, and 3.

See also

vecAMORE::size (alias)

Definition at line 52 of file vecCon.cpp.

References vecAMORE< Con >::ldata.

Referenced by getFromId().

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

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

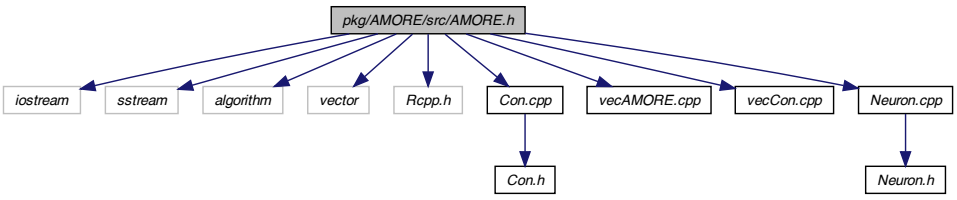
Chapter 7

File Documentation

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

```
#include <iostream>
#include <sstream>
#include <algorithm>
#include <vector>
#include <Rcpp.h>
#include "Con.cpp"
#include "vecAMORE.cpp"
#include "vecCon.cpp"
#include "Neuron.cpp"
```

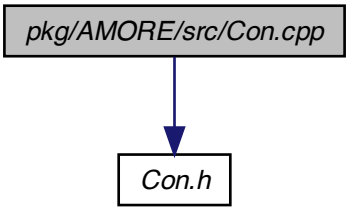
Include dependency graph for AMORE.h:



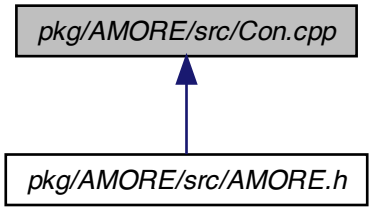
7.2 pkg/AMORE/src/Con.cpp File Reference

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

Include dependency graph for Con.cpp:

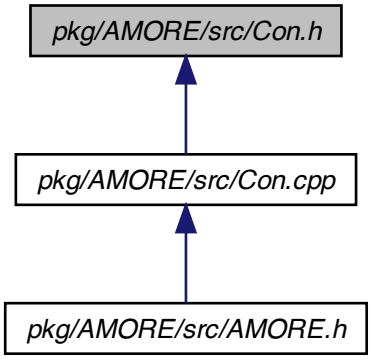


This graph shows which files directly or indirectly include this file:



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

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



Classes

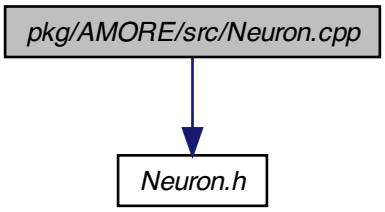
- class Con

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

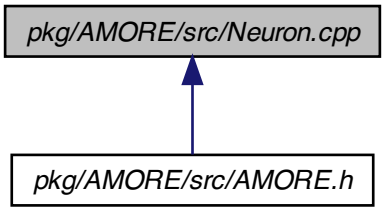
7.4 pkg/AMORE/src/Neuron.cpp File Reference

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

Include dependency graph for Neuron.cpp:

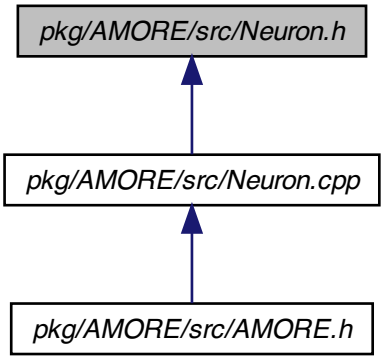


This graph shows which files directly or indirectly include this file:



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

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



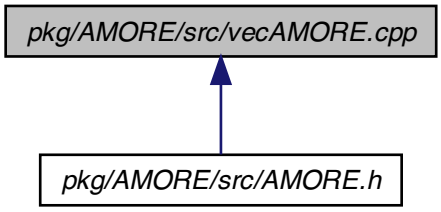
Classes

- 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:



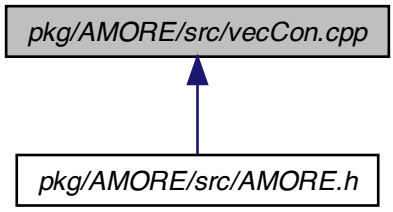
7.7 pkg/AMORE/src/vecAMORE.h File Reference

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

Classes

- class vecCon

  A vector of connections.

Index