

AMORE++

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

Generated by Doxygen 1.7.4

Wed Jul 13 2011 23:19:04

Contents

1	The AMORE++ package	1
1.1	Introduction	1
1.2	Motivation	1
1.3	Road Map	1
2	Class Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Class Documentation	9
5.1	AdaptBehavior Class Reference	9
5.1.1	Detailed Description	11
5.1.2	Member Function Documentation	11
5.1.2.1	adjustParameters	11
5.2	ADAPTgd Class Reference	12
5.2.1	Detailed Description	13
5.2.2	Member Function Documentation	13
5.2.2.1	adjustParameters	14
5.2.3	Member Data Documentation	14
5.2.3.1	outputDerivative	14
5.3	ADAPTgdwm Class Reference	14
5.3.1	Detailed Description	16

5.3.2	Member Function Documentation	16
5.3.2.1	adjustParameters	17
5.3.3	Member Data Documentation	17
5.3.3.1	outputDerivative	17
5.4	BatchBehavior Class Reference	17
5.4.1	Detailed Description	19
5.4.2	Member Function Documentation	19
5.4.2.1	adjustParameters	19
5.5	BATCHgd Class Reference	20
5.5.1	Detailed Description	21
5.5.2	Member Function Documentation	21
5.5.2.1	adjustParameters	22
5.5.3	Member Data Documentation	22
5.5.3.1	outputDerivative	22
5.6	BATCHgdwm Class Reference	22
5.6.1	Detailed Description	24
5.6.2	Member Function Documentation	24
5.6.2.1	adjustParameters	25
5.6.3	Member Data Documentation	25
5.6.3.1	outputDerivative	25
5.7	CompareId Struct Reference	25
5.7.1	Detailed Description	25
5.7.2	Member Function Documentation	25
5.7.2.1	operator()	25
5.7.2.2	operator()	25
5.7.2.3	operator()	26
5.7.2.4	operator()	26
5.8	Con Class Reference	26
5.8.1	Detailed Description	27
5.8.2	Constructor & Destructor Documentation	28
5.8.2.1	Con	28
5.8.2.2	Con	28
5.8.2.3	Con	28
5.8.2.4	Con	28

5.8.2.5	Con	28
5.8.2.6	Con	29
5.8.2.7	~Con	29
5.8.3	Member Function Documentation	29
5.8.3.1	getFrom	29
5.8.3.2	getId	30
5.8.3.3	getNeuron	30
5.8.3.4	getWeight	31
5.8.3.5	getWeight	32
5.8.3.6	Id	32
5.8.3.7	setFrom	33
5.8.3.8	setNeuron	34
5.8.3.9	setWeight	34
5.8.3.10	setWeight	34
5.8.3.11	show	34
5.8.3.12	show	35
5.8.3.13	validate	35
5.8.3.14	validate	36
5.8.4	Member Data Documentation	36
5.8.4.1	d_neuron	36
5.8.4.2	d_weight	36
5.8.4.3	from	36
5.8.4.4	weight	37
5.9	ConContainer Class Reference	37
5.9.1	Detailed Description	40
5.9.2	Member Typedef Documentation	40
5.9.2.1	const_iterator	40
5.9.2.2	const_reference	41
5.9.2.3	iterator	41
5.9.2.4	value_type	41
5.9.3	Constructor & Destructor Documentation	41
5.9.3.1	ConContainer	41
5.9.3.2	ConContainer	41
5.9.4	Member Function Documentation	41

5.9.4.1	erase	41
5.9.4.2	getId	43
5.9.4.3	numOfCons	45
5.9.4.4	select	46
5.9.4.5	setFrom	48
5.9.4.6	setWeight	50
5.9.4.7	setWeight	52
5.9.4.8	validate	53
5.10	Container< T > Class Template Reference	54
5.10.1	Detailed Description	58
5.10.2	Member Typedef Documentation	58
5.10.2.1	const_iterator	58
5.10.2.2	const_reference	58
5.10.2.3	iterator	58
5.10.2.4	value_type	58
5.10.3	Constructor & Destructor Documentation	59
5.10.3.1	~Container	59
5.10.3.2	Container	59
5.10.3.3	Container	59
5.10.3.4	Container	59
5.10.4	Member Function Documentation	59
5.10.4.1	append	59
5.10.4.2	begin	61
5.10.4.3	clear	61
5.10.4.4	clear	62
5.10.4.5	createIterator	62
5.10.4.6	empty	62
5.10.4.7	empty	63
5.10.4.8	end	63
5.10.4.9	load	63
5.10.4.10	operator[]	64
5.10.4.11	push_back	64
5.10.4.12	push_back	65
5.10.4.13	reserve	66

5.10.4.14	reserve	66
5.10.4.15	resize	67
5.10.4.16	show	67
5.10.4.17	show	67
5.10.4.18	size	68
5.10.4.19	size	68
5.10.4.20	store	69
5.10.4.21	validate	69
5.10.4.22	validate	69
5.10.5	Member Data Documentation	70
5.10.5.1	collection	70
5.11	Iterator< T > Class Template Reference	70
5.11.1	Detailed Description	71
5.11.2	Constructor & Destructor Documentation	72
5.11.2.1	~Iterator	72
5.11.2.2	Iterator	72
5.11.3	Member Function Documentation	72
5.11.3.1	currentItem	72
5.11.3.2	first	72
5.11.3.3	isDone	72
5.11.3.4	next	72
5.12	MLPbehavior Class Reference	72
5.12.1	Detailed Description	75
5.12.2	Member Function Documentation	75
5.12.2.1	predict	75
5.12.3	Member Data Documentation	75
5.12.3.1	d_accumulator	75
5.12.3.2	d_bias	75
5.12.3.3	d_nCons	75
5.12.3.4	d_output	76
5.13	MLPlayer Class Reference	76
5.13.1	Detailed Description	79
5.14	MLPlayerContainer Class Reference	79
5.14.1	Detailed Description	82

5.15 MLPneuralNet Class Reference	82
5.15.1 Detailed Description	84
5.15.2 Member Data Documentation	84
5.15.2.1 nLayers	84
5.16 MLPneuron Class Reference	84
5.16.1 Detailed Description	87
5.16.2 Member Data Documentation	87
5.16.2.1 bias	87
5.17 MLPneuronContainer Class Reference	87
5.17.1 Detailed Description	90
5.17.2 Member Function Documentation	90
5.17.2.1 buildAndAppend	90
5.17.2.2 getId	90
5.18 NeuralNet Class Reference	90
5.18.1 Detailed Description	91
5.18.2 Member Function Documentation	91
5.18.2.1 train	91
5.19 Neuron Class Reference	91
5.19.1 Detailed Description	94
5.19.2 Constructor & Destructor Documentation	95
5.19.2.1 Neuron	95
5.19.2.2 Neuron	95
5.19.2.3 Neuron	95
5.19.2.4 ~Neuron	95
5.19.3 Member Function Documentation	95
5.19.3.1 adjustParameters	95
5.19.3.2 getConId	95
5.19.3.3 getId	95
5.19.3.4 getId	95
5.19.3.5 getWeight	96
5.19.3.6 numOfCons	96
5.19.3.7 predict	96
5.19.3.8 setFrom	96
5.19.3.9 setId	96

5.19.3.10	setId	96
5.19.3.11	setPredictBehavior	96
5.19.3.12	setTrainingBehavior	96
5.19.3.13	setWeight	96
5.19.3.14	show	96
5.19.3.15	show	97
5.19.3.16	validate	97
5.19.3.17	validate	97
5.19.4	Member Data Documentation	97
5.19.4.1	con	98
5.19.4.2	d_predictBehavior	98
5.19.4.3	d_trainingBehavior	98
5.19.4.4	Id	98
5.19.4.5	outputValue	98
5.20	NeuronContainer Class Reference	98
5.20.1	Detailed Description	101
5.20.2	Member Typedef Documentation	101
5.20.2.1	const_iterator	101
5.20.2.2	const_reference	101
5.20.2.3	iterator	102
5.20.2.4	value_type	102
5.20.3	Constructor & Destructor Documentation	102
5.20.3.1	NeuronContainer	102
5.20.3.2	NeuronContainer	102
5.20.3.3	~NeuronContainer	102
5.20.4	Member Function Documentation	102
5.20.4.1	getConId	102
5.20.4.2	getFrom	103
5.20.4.3	getId	103
5.20.4.4	getWeight	103
5.20.4.5	numOfCons	103
5.20.4.6	numOfNeurons	104
5.20.4.7	setFrom	104
5.20.4.8	setId	104

5.20.4.9	setWeight	105
5.21	PredictBehavior Class Reference	105
5.21.1	Detailed Description	106
5.21.2	Member Function Documentation	106
5.21.2.1	predict	106
5.22	RBFbehavior Class Reference	106
5.22.1	Detailed Description	109
5.22.2	Member Function Documentation	109
5.22.2.1	predict	109
5.22.3	Member Data Documentation	109
5.22.3.1	d_accumulator	109
5.22.3.2	d_altitude	109
5.22.3.3	d_nCons	109
5.22.3.4	d_output	109
5.22.3.5	d_width	110
5.23	RBFneuralNet Class Reference	110
5.23.1	Detailed Description	111
5.24	SimpleContainer< T > Class Template Reference	111
5.24.1	Detailed Description	114
5.24.2	Constructor & Destructor Documentation	114
5.24.2.1	SimpleContainer	114
5.24.2.2	~SimpleContainer	114
5.24.3	Member Function Documentation	114
5.24.3.1	clear	115
5.24.3.2	createIterator	115
5.24.3.3	empty	115
5.24.3.4	push_back	115
5.24.3.5	reserve	116
5.24.3.6	show	116
5.24.3.7	size	117
5.24.3.8	validate	117
5.24.4	Friends And Related Function Documentation	117
5.24.4.1	SimpleContainerIterator< T >	117
5.24.5	Member Data Documentation	117

5.24.5.1	d_collection	118
5.25	SimpleContainerIterator< T > Class Template Reference	118
5.25.1	Detailed Description	120
5.25.2	Constructor & Destructor Documentation	120
5.25.2.1	SimpleContainerIterator	120
5.25.2.2	~SimpleContainerIterator	120
5.25.3	Member Function Documentation	120
5.25.3.1	currentItem	120
5.25.3.2	first	120
5.25.3.3	isDone	120
5.25.3.4	next	121
5.25.4	Friends And Related Function Documentation	121
5.25.4.1	SimpleContainer< T >	121
5.25.5	Member Data Documentation	121
5.25.5.1	d_container	121
5.25.5.2	d_iterator	121
5.26	SimpleNeuron Class Reference	121
5.26.1	Detailed Description	124
5.26.2	Constructor & Destructor Documentation	124
5.26.2.1	SimpleNeuron	124
5.26.3	Member Function Documentation	124
5.26.3.1	adjustParameters	124
5.26.3.2	getId	124
5.26.3.3	predict	124
5.26.3.4	setId	124
5.26.3.5	setPredictBehavior	125
5.26.3.6	setTrainingBehavior	125
5.26.3.7	show	125
5.26.3.8	validate	125
5.26.4	Member Data Documentation	125
5.26.4.1	d_Id	125
5.27	TrainingBehavior Class Reference	125
5.27.1	Detailed Description	126
5.27.2	Member Function Documentation	126

5.27.2.1	adjustParameters	126
6	File Documentation	127
6.1	pkg/AMORE/src/AMORE.h File Reference	127
6.1.1	Define Documentation	128
6.1.1.1	foreach	128
6.1.1.2	size_type	128
6.1.2	Typedef Documentation	128
6.1.2.1	ConContainer	128
6.1.2.2	ConIteratorInterfacePtr	128
6.1.2.3	ConPtr	128
6.1.2.4	Handler	128
6.1.2.5	NeuronContainer	129
6.1.2.6	NeuronIteratorInterfacePtr	129
6.1.2.7	NeuronPtr	129
6.1.2.8	NeuronRef	129
6.1.2.9	PredictBehaviorRef	129
6.1.2.10	TrainingBehaviorRef	129
6.2	pkg/AMORE/src/Con.cpp File Reference	129
6.3	pkg/AMORE/src/old/Con.cpp File Reference	130
6.4	pkg/AMORE/src/Container.cpp File Reference	131
6.5	pkg/AMORE/src/old/Container.cpp File Reference	131
6.6	pkg/AMORE/src/containerInterface.cpp File Reference	131
6.7	pkg/AMORE/src/ContainerIterator.cpp File Reference	132
6.8	pkg/AMORE/src/dia/AdaptBehavior.h File Reference	132
6.9	pkg/AMORE/src/dia/ADAPTgd.h File Reference	133
6.10	pkg/AMORE/src/dia/ADAPTgdwm.h File Reference	134
6.11	pkg/AMORE/src/dia/BatchBehavior.h File Reference	135
6.12	pkg/AMORE/src/dia/BATCHgd.h File Reference	136
6.13	pkg/AMORE/src/dia/BATCHgdwm.h File Reference	137
6.14	pkg/AMORE/src/dia/Con.h File Reference	138
6.15	pkg/AMORE/src/old/Con.h File Reference	138
6.16	pkg/AMORE/src/dia/Container.h File Reference	139
6.17	pkg/AMORE/src/old/Container.h File Reference	140

6.18	pkg/AMORE/src/dia/Iterator.h File Reference	140
6.19	pkg/AMORE/src/dia/MLPbehavior.h File Reference	141
6.20	pkg/AMORE/src/dia/Neuron.h File Reference	142
6.21	pkg/AMORE/src/old/Neuron.h File Reference	142
6.22	pkg/AMORE/src/dia/PredictBehavior.h File Reference	143
6.23	pkg/AMORE/src/dia/RBFbehavior.h File Reference	143
6.24	pkg/AMORE/src/dia/SimpleContainer.h File Reference	144
6.25	pkg/AMORE/src/dia/SimpleContainerIterator.h File Reference	145
6.26	pkg/AMORE/src/dia/SimpleNeuron.h File Reference	145
6.27	pkg/AMORE/src/dia/TrainingBehavior.h File Reference	146
6.28	pkg/AMORE/src/IteratorInterface.cpp File Reference	146
6.29	pkg/AMORE/src/Neuron.cpp File Reference	147
6.30	pkg/AMORE/src/old/Neuron.cpp File Reference	147
6.31	pkg/AMORE/src/old/ConContainer.cpp File Reference	147
6.32	pkg/AMORE/src/old/ConContainer.h File Reference	148
6.33	pkg/AMORE/src/old/MLPlayer.h File Reference	148
6.34	pkg/AMORE/src/old/MLPlayerContainer.h File Reference	148
6.35	pkg/AMORE/src/old/MLPneuralNet.h File Reference	148
6.36	pkg/AMORE/src/old/MLPneuralNetFactory.cpp File Reference	148
6.36.1	Function Documentation	148
6.36.1.1	CreateMLPneuralNet	148
6.37	pkg/AMORE/src/old/MLPneuron.h File Reference	149
6.38	pkg/AMORE/src/old/MLPneuronContainer.h File Reference	149
6.39	pkg/AMORE/src/old/NeuralNet.h File Reference	149
6.40	pkg/AMORE/src/old/NeuronContainer.cpp File Reference	150
6.41	pkg/AMORE/src/old/NeuronContainer.h File Reference	150
6.42	pkg/AMORE/src/old/RBFneuralNet.h File Reference	150

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	25
Con	26
Container< T >	54
SimpleContainer< T >	111
Container< Con >	54
ConContainer	37
Container< MLPlayer >	54
MLPlayerContainer	79
Container< Neuron >	54
NeuronContainer	98
Iterator< T >	70
SimpleContainerIterator< T >	118
NeuralNet	90
MLPneuralNet	82
RBFneuralNet	110
Neuron	91
MLPneuron	84
SimpleNeuron	121
NeuronContainer< MLP >	98
MLPneuronContainer	87
MLPlayer	76
PredictBehavior	105
MLPbehavior	72
RBFbehavior	106
TrainingBehavior	125
AdaptBehavior	9

ADAPTgd	12
ADAPTgdwm	14
BatchBehavior	17
BATCHgd	20
BATCHgdwm	22

Chapter 3

Class Index

3.1 Class List

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

AdaptBehavior (Class AdaptBehavior -)	9
ADAPTgd (Class ADAPTgd -)	12
ADAPTgdwm (Class ADAPTgdwm -)	14
BatchBehavior (Class BatchBehavior -)	17
BATCHgd (Class BATCHgd -)	20
BATCHgdwm (Class BATCHgdwm -)	22
CompareId	25
Con (Class Con -)	26
ConContainer (A vector of connections)	37
Container< T > (Class Container -)	54
Iterator< T > (Class Iterator -)	70
MLPbehavior (Class MLPbehavior -)	72
MLPlayer	76
MLPlayerContainer	79
MLPneuralNet	82
MLPneuron	84
MLPneuronContainer (A vector of connections)	87
NeuralNet	90
Neuron (Class Neuron -)	91
NeuronContainer (A vector of neurons)	98
PredictBehavior (Class PredictBehavior -)	105
RBFbehavior (Class RBFbehavior -)	106
RBFneuralNet	110
SimpleContainer< T > (Class SimpleContainer -)	111
SimpleContainerIterator< T > (Class SimpleContainerIterator -)	118
SimpleNeuron (Class SimpleNeuron -)	121
TrainingBehavior (Class TrainingBehavior -)	125

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

pkg/AMORE/src/AMORE.h	127
pkg/AMORE/src/Con.cpp	129
pkg/AMORE/src/Container.cpp	131
pkg/AMORE/src/containerInterface.cpp	131
pkg/AMORE/src/ContainerIterator.cpp	132
pkg/AMORE/src/IteratorInterface.cpp	146
pkg/AMORE/src/Neuron.cpp	147
pkg/AMORE/src/dia/AdaptBehavior.h	132
pkg/AMORE/src/dia/ADAPTgd.h	133
pkg/AMORE/src/dia/ADAPTgdwm.h	134
pkg/AMORE/src/dia/BatchBehavior.h	135
pkg/AMORE/src/dia/BATCHgd.h	136
pkg/AMORE/src/dia/BATCHgdwm.h	137
pkg/AMORE/src/dia/Con.h	138
pkg/AMORE/src/dia/Container.h	139
pkg/AMORE/src/dia/Iterator.h	140
pkg/AMORE/src/dia/MLPbehavior.h	141
pkg/AMORE/src/dia/Neuron.h	142
pkg/AMORE/src/dia/PredictBehavior.h	143
pkg/AMORE/src/dia/RBFbehavior.h	143
pkg/AMORE/src/dia/SimpleContainer.h	144
pkg/AMORE/src/dia/SimpleContainerIterator.h	145
pkg/AMORE/src/dia/SimpleNeuron.h	145
pkg/AMORE/src/dia/TrainingBehavior.h	146
pkg/AMORE/src/old/Con.cpp	130
pkg/AMORE/src/old/Con.h	138
pkg/AMORE/src/old/ConContainer.cpp	147
pkg/AMORE/src/old/ConContainer.h	148
pkg/AMORE/src/old/Container.cpp	131

pkg/AMORE/src/old/ Container.h	140
pkg/AMORE/src/old/ MLPlayer.h	148
pkg/AMORE/src/old/ MLPlayerContainer.h	148
pkg/AMORE/src/old/ MLPneuralNet.h	148
pkg/AMORE/src/old/ MLPneuralNetFactory.cpp	148
pkg/AMORE/src/old/ MLPneuron.h	149
pkg/AMORE/src/old/ MLPneuronContainer.h	149
pkg/AMORE/src/old/ NeuralNet.h	149
pkg/AMORE/src/old/ Neuron.cpp	147
pkg/AMORE/src/old/ Neuron.h	142
pkg/AMORE/src/old/ NeuronContainer.cpp	150
pkg/AMORE/src/old/ NeuronContainer.h	150
pkg/AMORE/src/old/ RBFneuralNet.h	150

Chapter 5

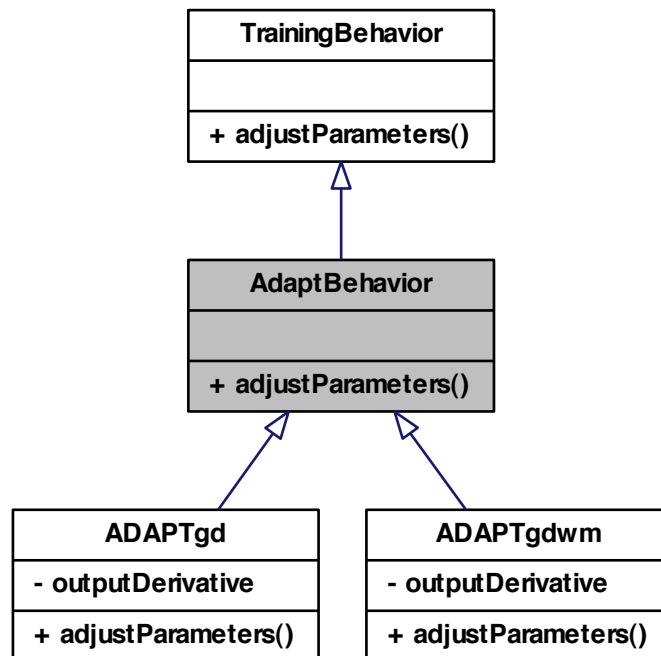
Class Documentation

5.1 `AdaptBehavior` Class Reference

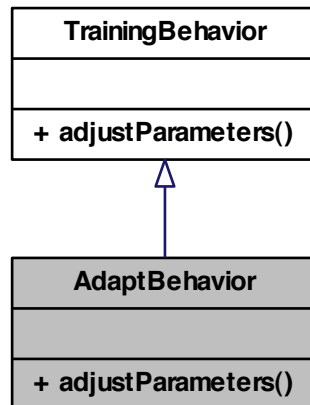
class `AdaptBehavior` -

```
#include <AdaptBehavior.h>
```

Inheritance diagram for AdaptBehavior:



Collaboration diagram for AdaptBehavior:



Public Member Functions

- virtual void [adjustParameters](#) ()=0

5.1.1 Detailed Description

class [AdaptBehavior](#) -

Definition at line 5 of file [AdaptBehavior.h](#).

5.1.2 Member Function Documentation

5.1.2.1 virtual void [AdaptBehavior::adjustParameters](#) () [pure virtual]

Reimplemented from [TrainingBehavior](#).

Implemented in [ADAPTgd](#), and [ADAPTgdwm](#).

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

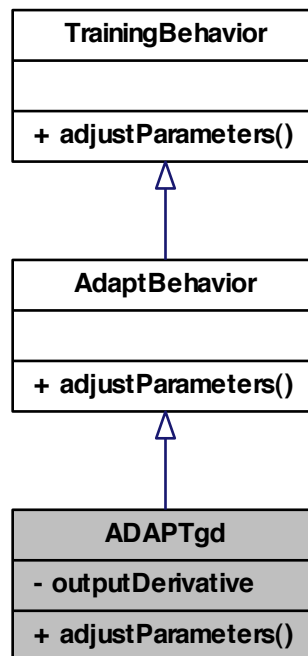
- [pkg/AMORE/src/dia/AdaptBehavior.h](#)

5.2 ADAPTgd Class Reference

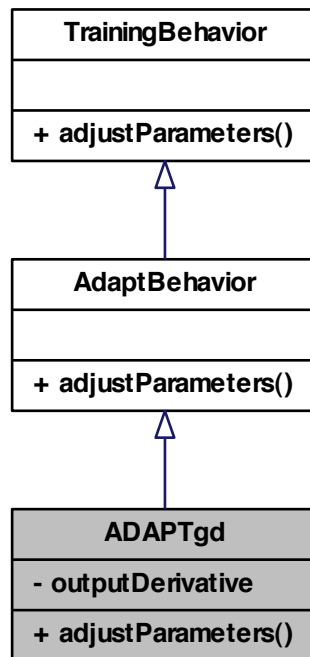
class [ADAPTgd](#) -

```
#include <ADAPTgd.h>
```

Inheritance diagram for ADAPTgd:



Collaboration diagram for ADAPTgd:



Public Member Functions

- void [adjustParameters](#) ()

Private Attributes

- double [outputDerivative](#)

5.2.1 Detailed Description

class [ADAPTgd](#) -

Definition at line 5 of file ADAPTgd.h.

5.2.2 Member Function Documentation

5.2.2.1 void ADAPTgd::adjustParameters () [virtual]

Implements [AdaptBehavior](#).

5.2.3 Member Data Documentation

5.2.3.1 double ADAPTgd::outputDerivative [private]

Definition at line 8 of file ADAPTgd.h.

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

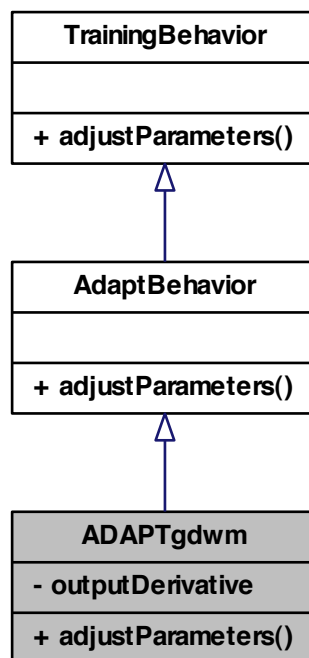
- pkg/AMORE/src/dia/[ADAPTgd.h](#)

5.3 ADAPTgdwm Class Reference

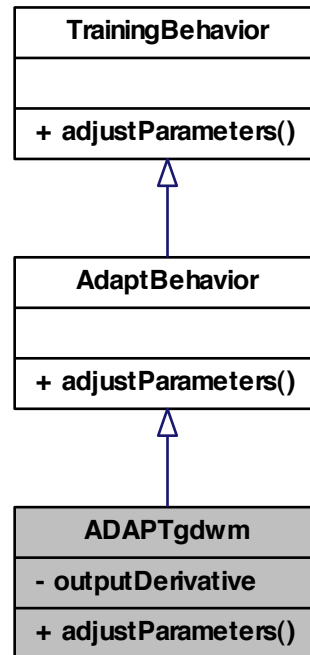
class [ADAPTgdwm](#) -

#include <ADAPTgdwm.h>

Inheritance diagram for ADAPTgdwm:



Collaboration diagram for ADAPTgdwm:



Public Member Functions

- void [adjustParameters](#) ()

Private Attributes

- double [outputDerivative](#)

5.3.1 Detailed Description

class [ADAPTgdwm](#) -

Definition at line 5 of file ADAPTgdwm.h.

5.3.2 Member Function Documentation

5.3.2.1 void ADAPTgdwm::adjustParameters () [virtual]

Implements [AdaptBehavior](#).

5.3.3 Member Data Documentation

5.3.3.1 double ADAPTgdwm::outputDerivative [private]

Definition at line 8 of file ADAPTgdwm.h.

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

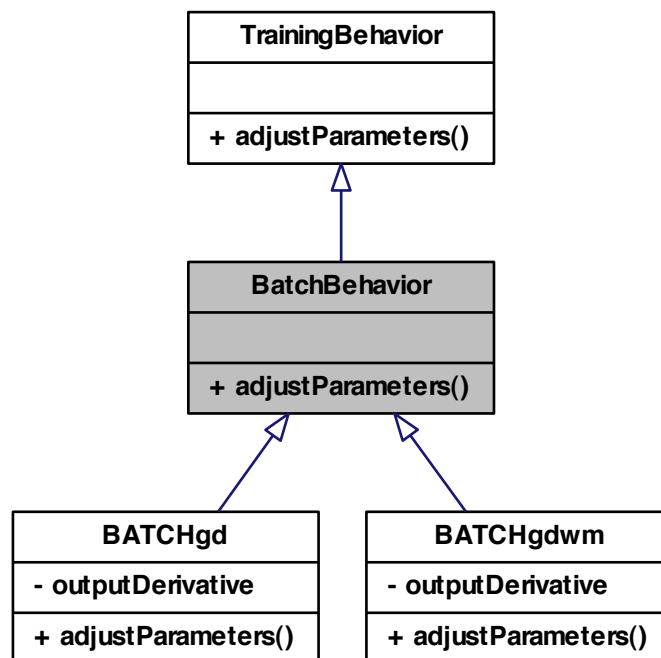
- pkg/AMORE/src/dia/[ADAPTgdwm.h](#)

5.4 BatchBehavior Class Reference

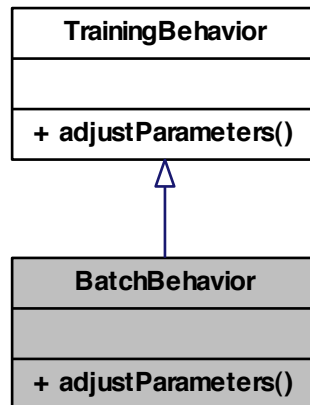
class [BatchBehavior](#) -

```
#include <BatchBehavior.h>
```

Inheritance diagram for BatchBehavior:



Collaboration diagram for BatchBehavior:



Public Member Functions

- virtual void [adjustParameters](#) ()=0

5.4.1 Detailed Description

class [BatchBehavior](#) -

Definition at line 5 of file [BatchBehavior.h](#).

5.4.2 Member Function Documentation

5.4.2.1 virtual void [BatchBehavior::adjustParameters](#) () `[pure virtual]`

Reimplemented from [TrainingBehavior](#).

Implemented in [BATCHgd](#), and [BATCHgdwm](#).

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

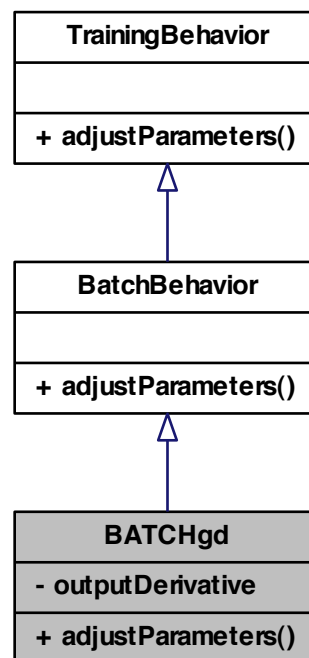
- [pkg/AMORE/src/dia/BatchBehavior.h](#)

5.5 BATCHgd Class Reference

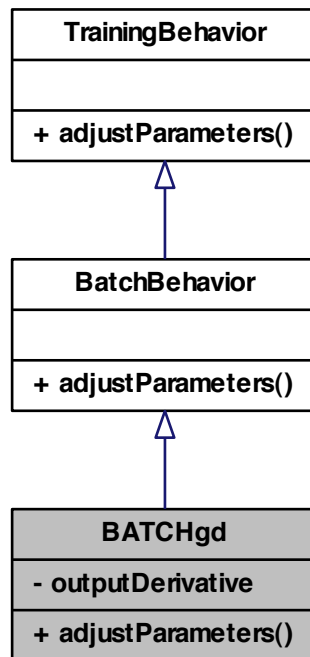
class [BATCHgd](#) -

```
#include <BATCHgd.h>
```

Inheritance diagram for BATCHgd:



Collaboration diagram for BATCHgd:



Public Member Functions

- void [adjustParameters](#) ()

Private Attributes

- double [outputDerivative](#)

5.5.1 Detailed Description

class [BATCHgd](#) -

Definition at line 5 of file BATCHgd.h.

5.5.2 Member Function Documentation

5.5.2.1 void BATCHgd::adjustParameters () [virtual]

Implements [BatchBehavior](#).

5.5.3 Member Data Documentation

5.5.3.1 double BATCHgd::outputDerivative [private]

Definition at line 8 of file BATCHgd.h.

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

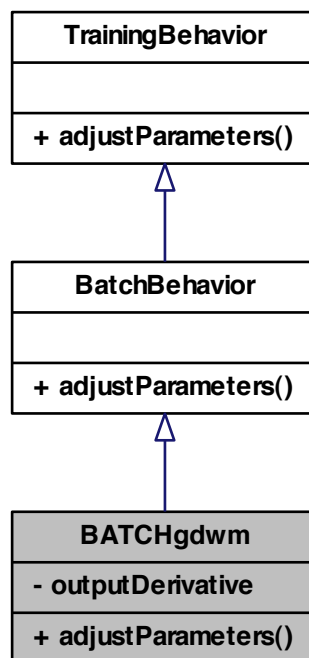
- pkg/AMORE/src/dia/[BATCHgd.h](#)

5.6 BATCHgdwm Class Reference

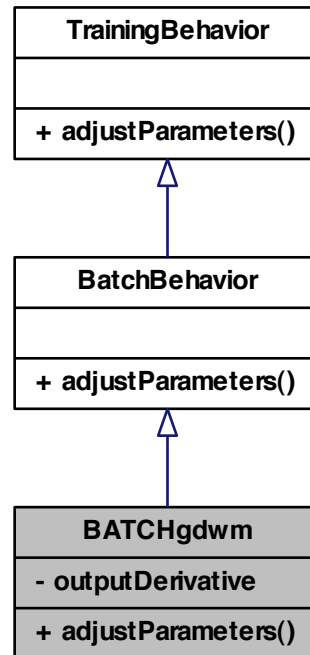
class [BATCHgdwm](#) -

```
#include <BATCHgdwm.h>
```

Inheritance diagram for BATCHgdwm:



Collaboration diagram for BATCHgdwm:



Public Member Functions

- void [adjustParameters](#) ()

Private Attributes

- double [outputDerivative](#)

5.6.1 Detailed Description

class [BATCHgdwm](#) -

Definition at line 5 of file BATCHgdwm.h.

5.6.2 Member Function Documentation

5.6.2.1 void BATCHgdwm::adjustParameters () [virtual]

Implements [BatchBehavior](#).

5.6.3 Member Data Documentation

5.6.3.1 double BATCHgdwm::outputDerivative [private]

Definition at line 8 of file BATCHgdwm.h.

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

- pkg/AMORE/src/dia/BATCHgdwm.h

5.7 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)

5.7.1 Detailed Description

Definition at line 352 of file ConContainer.cpp.

5.7.2 Member Function Documentation

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

Definition at line 356 of file ConContainer.cpp.

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

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

Definition at line 377 of file ConContainer.cpp.

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

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

Definition at line 370 of file ConContainer.cpp.

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

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

Definition at line 363 of file ConContainer.cpp.

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

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

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

5.8 Con Class Reference

class [Con](#) -

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

Public Member Functions

- [Con](#) ()
Default Constructor.
- [Con](#) ([Neuron](#) &neuron)
Constructor.
- [Con](#) ([Neuron](#) &neuron, double [weight](#))
Constructor.
- [Handler](#) [Id](#) ()
A getter of the Id of the [Neuron](#) pointed by the from field.
- [Neuron](#) & [getNeuron](#) ()
from field accessor.
- void [setNeuron](#) ([Neuron](#) &neuron)
- double [getWeight](#) ()
weight field accessor.
- void [setWeight](#) (double [weight](#))
weight field accessor.
- void [show](#) ()

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

- bool [validate](#) ()

Object validator.

- [Con](#) ()
- [Con](#) ([NeuronPtr](#) neuronPtr)

Constructor.

- [Con](#) ([NeuronPtr](#) neuronPtr, double value)

Constructor.

- [~Con](#) ()

Default Destructor.

- [NeuronPtr](#) [getFrom](#) ()

from field accessor.

- void [setFrom](#) ([NeuronPtr](#) neuronPtr)

from field accessor.

- int [getId](#) ()

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

- double [getWeight](#) ()
- void [setWeight](#) (double value)
- bool [show](#) ()
- bool [validate](#) ()

Private Attributes

- [NeuronRef](#) [d_neuron](#)
- double [d_weight](#)
- [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.

5.8.1 Detailed Description

class [Con](#) -

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

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

Definition at line 3 of file [Con.h](#).

5.8.2 Constructor & Destructor Documentation

5.8.2.1 Con::Con ()

Default Constructor.

Definition at line 17 of file Con.cpp.

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

5.8.2.2 Con::Con (Neuron & *neuron*)

Constructor.

Definition at line 17 of file Con.cpp.

```
        :  
        d_neuron( boost::ref(neuron) ), d_weight(0)  
    {  
    }
```

5.8.2.3 Con::Con (Neuron & *neuron*, double *weight*)

Constructor.

Definition at line 28 of file Con.cpp.

```
        :  
        d_neuron(boost::ref(neuron)), d_weight(weight)  
    {  
    }
```

5.8.2.4 Con::Con ()

5.8.2.5 Con::Con (NeuronPtr *neuronPtr*)

Constructor.

Definition at line 40 of file Con.cpp.

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

5.8.2.6 Con::Con (NeuronPtr neuronPtr, double value)

Constructor.

Definition at line 29 of file Con.cpp.

```

        from(neuronPtr), weight(value)
    {
    }

```

5.8.2.7 Con::~Con ()

Default Destructor.

Definition at line 46 of file Con.cpp.

```

{
}

```

5.8.3 Member Function Documentation

5.8.3.1 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 71 of file Con.cpp.

References from.

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

5.8.3.2 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 123 of file `Con.cpp`.

References from.

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

5.8.3.3 Neuron & Con::getNeuron ()

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 54 of file `Con.cpp`.

References `d_neuron`.

```
{
    return d_neuron;
}
```

5.8.3.4 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 114 of file `Con.cpp`.

References `d_weight`.

```
{
    return d_weight;
}
```

5.8.3.5 double Con::getWeight ()**5.8.3.6 int Con::Id ()**

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

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

Returns

The value of the Id (an integer).

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

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

See also

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

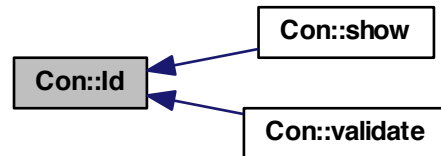
Definition at line 86 of file `Con.cpp`.

References `d_neuron`.

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

```
{
    return d_neuron.get().Id();
}
```

Here is the caller graph for this function:



5.8.3.7 void Con::setFrom (NeuronPtr neuronPtr)

from field accessor.

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

Parameters

<i>f</i>	A pointer to the neuron that is to be inserted in the from field.
----------	---

```

//=====
//Usage example:
//=====
// Data set up
NeuronPtr ptShNeuron ( new Neuron(1) );           // Neuron
Id is set to 1
ConPtr ptShCon( new Con() );
ptShCon->setFrom( ptShNeuron );

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

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

See also

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

Definition at line 98 of file `Con.cpp`.

References from.

```

{
    from = neuronPtr;
}
  
```

5.8.3.8 void Con::setNeuron (Neuron & neuron)

Definition at line 61 of file Con.cpp.

References `d_neuron`.

```
{
    d_neuron=boost::ref(neuron);
}
```

5.8.3.9 void Con::setWeight (double value)

weight field accessor.

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

Parameters

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

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

std::vector<double> result;
NeuronPtr ptShNeuron ( new Neuron(16) );           /
/ Neuron Id is set to 16
ConPtr ptShCon( new Con(ptShNeuron, 12.4) ); // from poi
nts to ptShNeuron and weight is set to 12.4
result.push_back(ptShCon->getWeight());

// Test
ptShCon->setWeight(2.2);
result.push_back(ptShCon->getWeight());

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

See also

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

Definition at line 121 of file Con.cpp.

References `d_weight`, and `weight`.

```
{
    d_weight=weight;
}
```

5.8.3.10 void Con::setWeight (double value)

5.8.3.11 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 133 of file `Con.cpp`.

References `d_weight`, and `Id()`.

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

Here is the call graph for this function:



5.8.3.12 `bool Con::show ()`

5.8.3.13 `bool Con::validate ()`

Object validator.

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

Returns

true in case the checks are Ok.

Exceptions

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

Definition at line 153 of file Con.cpp.

References `Id()`, and `weight`.

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

Here is the call graph for this function:



5.8.3.14 bool Con::validate ()

5.8.4 Member Data Documentation

5.8.4.1 NeuronRef Con::d_neuron [private]

Definition at line 6 of file Con.h.

Referenced by `getNeuron()`, `Id()`, and `setNeuron()`.

5.8.4.2 double Con::d_weight [private]

Definition at line 7 of file Con.h.

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

5.8.4.3 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 22 of file Con.h.

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

5.8.4.4 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 27 of file Con.h.

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

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

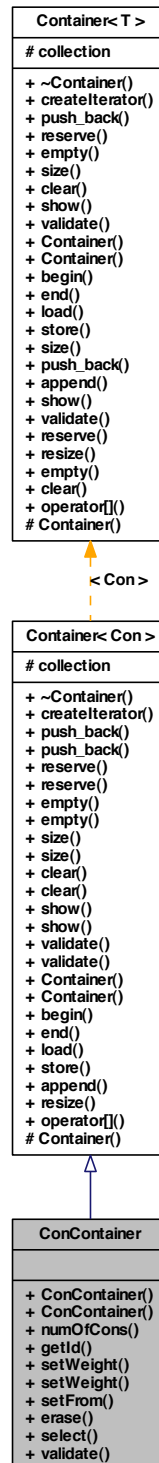
- `pkg/AMORE/src/dia/Con.h`
- `pkg/AMORE/src/old/Con.h`
- `pkg/AMORE/src/Con.cpp`
- `pkg/AMORE/src/old/Con.cpp`

5.9 ConContainer Class Reference

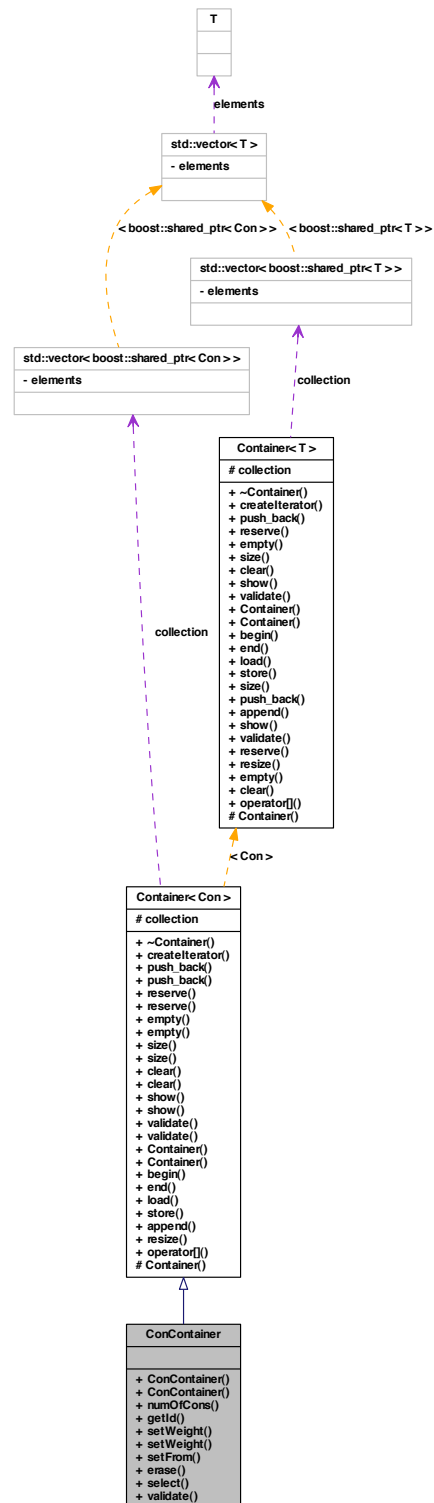
A vector of connections.

```
#include <ConContainer.h>
```

Inheritance diagram for ConContainer:



Collaboration diagram for ConContainer:



Public Types

- typedef std::vector< boost::shared_ptr< [Con](#) > >::iterator [iterator](#)
- typedef std::vector< boost::shared_ptr< [Con](#) > >::const_iterator [const_iterator](#)
- typedef boost::shared_ptr< [Con](#) > [value_type](#)
- typedef [value_type](#) const & [const_reference](#)

Public Member Functions

- [ConContainer](#) ()
- [ConContainer](#) (std::vector< [ConPtr](#) > [collection](#))
- int [numOfCons](#) ()
Size of the [ConContainer](#) object.
- std::vector< int > [getId](#) ()
Getter of the Id values of the vector of Cons.
- bool [setWeight](#) (std::vector< double > [nWeights](#))
Setter of the weight field of the [Con](#) objects related to [ConContainer](#).
- bool [setWeight](#) (std::vector< double > [nWeights](#), std::vector< int > [nIds](#))
Setter of the weights of the specified elements from the [ConContainer](#) object.
- bool [setFrom](#) ([NeuronContainer](#) [neuronContainer](#))
Setter of the from fields of the [Con](#) objects related to [ConContainer](#).
- void [erase](#) (std::vector< int > [nIds](#))
Erase the specified elements from the [vecCom](#) object.
- [ConContainerPtr](#) [select](#) (std::vector< int > [nIds](#))
Selects the specified elements from the [vecCom](#) object.
- bool [validate](#) ()
Object validator.

5.9.1 Detailed Description

A vector of connections.

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

Definition at line 16 of file [ConContainer.h](#).

5.9.2 Member Typedef Documentation

5.9.2.1 typedef std::vector<boost::shared_ptr<Con> >::const_iterator [ConContainer::const_iterator](#)

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

Definition at line 23 of file [ConContainer.h](#).

5.9.2.2 typedef value_type const& ConContainer::const_reference

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

Definition at line 27 of file ConContainer.h.

5.9.2.3 typedef std::vector<boost::shared_ptr<Con> >::iterator ConContainer::iterator

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

Definition at line 21 of file ConContainer.h.

5.9.2.4 typedef boost::shared_ptr<Con> ConContainer::value_type

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

Definition at line 25 of file ConContainer.h.

5.9.3 Constructor & Destructor Documentation**5.9.3.1 ConContainer::ConContainer ()**

Definition at line 8 of file ConContainer.cpp.

```
{
}
```

5.9.3.2 ConContainer::ConContainer (std::vector< ConPtr > collection)

Definition at line 12 of file ConContainer.cpp.

```

        :
        Container<Con> (collection) // Call to Base constructor
    {
    }

```

5.9.4 Member Function Documentation**5.9.4.1 void ConContainer::erase (std::vector< int > nlds)**

Erase the specified elements from the vecCom object.

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

Parameters

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

```

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

// Data set up
std::vector<int> result;
std::vector<NeuronPtr> neuronContainer;
ConContainerPtr conContainerPtr( new ConContainer() );
ConContainerPtr vErased;
ConPtr ptC;
NeuronPtr ptN;
int ids[]= {11, 10, 9, 3, 4, 5, 6, 7, 8, 2, 1};
std::vector<double> nWeights;
nWeights.push_back(11.32);
nWeights.push_back(1.26);
nWeights.push_back(2.14);
nWeights.push_back(3.16);
nWeights.push_back(4.14);
nWeights.push_back(5.19);
nWeights.push_back(6.18);
nWeights.push_back(7.16);
nWeights.push_back(8.14);
nWeights.push_back(9.12);
nWeights.push_back(10.31);

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

// Test

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

conContainerPtr->erase(toRemove);
conContainerPtr->show();
result=conContainerPtr->getId();

// The output at the R terminal would display :
//
// From:      2      Weight=      9.120000
// From:      4      Weight=      4.140000
// From:      6      Weight=      6.180000
// From:      8      Weight=      8.140000
// From:      9      Weight=      2.140000
// From:     10      Weight=      1.260000
// From:     11      Weight=     11.320000

```

See also

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

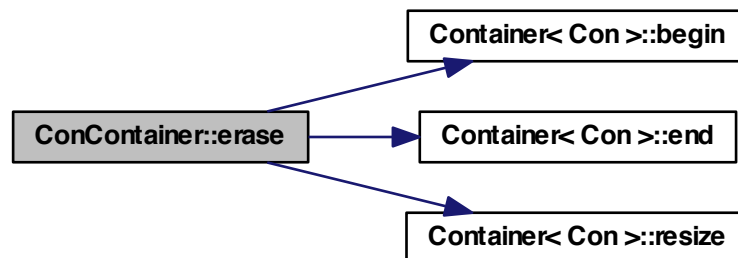
Definition at line 450 of file `ConContainer.cpp`.

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

Con >::resize().

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

Here is the call graph for this function:



5.9.4.2 std::vector<int> ConContainer::getId ()

Getter of the Id values of the vector of Cons.

This function returns the Id's of the neurons referred to by the vector of Cons.

Returns

An `std::vector<int>` that contains the Ids

```
//=====
//Usage example:
//=====
// Data set up
Neuron N1, N2, N3;
ConContainer conContainer;
std::vector<int> result;

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

ConPtr ptCon( new Con(&N1, 1.13) );    // Create new Con
```

```

    and initialize ptCon
        conContainer.push_back(ptCon);
/ push_back
        ptCon.reset( new Con(&N2, 2.22) );
new Con and assign to ptCon
        conContainer.push_back(ptCon);
/ push_back
        ptCon.reset( new Con(&N3, 3.33) );
new Con and assign to ptCon
        conContainer.push_back(ptCon);
/ push_back

// Test
        conContainer.show() ;
        conContainer.validate();
        result=conContainer.getId();

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

```

See also

getWeight and the unit test files, e.g. runit.Cpp.ConContainer.R, for further examples.

Definition at line 93 of file ConContainer.cpp.

References numOfCons().

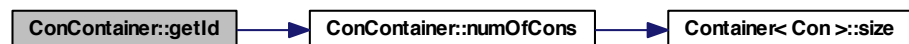
Referenced by validate().

```

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

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.4.3 int ConContainer::numOfCons ()

Size of the [ConContainer](#) object.

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

Returns

The size of the vector

```

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

Container<Neuron>( ) );
ConContainer( ) );

std::vector<int> result;
std::vector<ConPtr> vcA, vcB;
ContainerNeuronPtr neuronContainerPtr( new
ConContainerPtr conContainerPtr( new
ConPtr ptC;
NeuronPtr ptN;
int ids[] = {10, 20, 30};
double weights[] = {1.13, 2.22, 3.33 };
for (int i=0; i<=2 ; i++) {
    ptN.reset( new Neuron( ids[i] ) );
    neuronContainerPtr->push_back(ptN);
}

// Test
for (int i=0; i<=2 ; i++) {
    result.push_back(conContainerPtr->numOfCons());
    // Append numOfCons to result, create new Con and push_back into
    conContainer
    ptC.reset( new Con( neuronContainerPtr->load().at(i), weights[i]) );
    conContainerPtr->push_back(ptC);
}

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

See also

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

Definition at line 52 of file ConContainer.cpp.

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

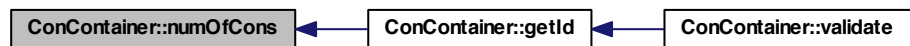
Referenced by `getId()`.

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

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.4.4 ConContainerPtr ConContainer::select (std::vector< int > nlds)

Selects the specified elements from the vecCom object.

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

Parameters

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

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

```

//=====

// Data set up
std::vector<int> result;
std::vector<NeuronPtr> neuronContainer;
ConContainerPtr conContainerPtr( new ConContainer() );
ConPtr ptC;
NeuronPtr ptN;
int ids[]= {11, 10, 9, 3, 4, 5, 6, 7, 8, 2, 1};
double weights[]={11.32, 1.26, 2.14, 3.16, 4.14, 5.19, 6.18, 7.16
, 8.14, 9.12, 10.31};
std::vector<double> nWeights;
for (int i=0; i<11; i++) {
    nWeights.push_back(weights[i]);
}
for (int i=0; i<nWeights.size() ; i++) {
/ Let's create a vector with three neurons
    ptN.reset( new Neuron( ids[i] ) );
    neuronContainer.push_back(ptN);
}
conContainerPtr->buildAndAppend(neuronContainer, nWeights);
// Test
std::vector<int> toSelect;
toSelect.push_back(1);
toSelect.push_back(3);
toSelect.push_back(5);
toSelect.push_back(7);

ConContainerPtr vSelect ( conContainerPtr->select(toSelect) );

result=vSelect->getId();

// Now, result is a numeric vector with the values 1, 3, 5 and 7.

```

See also

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

Definition at line 505 of file `ConContainer.cpp`.

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

Referenced by `setWeight()`.

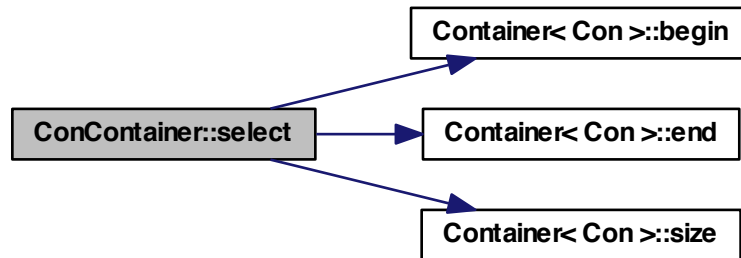
```

{
    ConContainerPtr result(new ConContainer);
    result->reserve(size());
    sort(begin(), end(), CompareId());
    sort(nIds.begin(), nIds.end());
    set_intersection(begin(), end(), nIds.begin(), nIds.end(),
        std::back_inserter(*result), CompareId());

    return result;
}

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.4.5 `bool ConContainer::setFrom (NeuronContainer neuronContainer)`

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

This function provides a convenient way of getting the values of the weight field of those [Con](#) object pointed to by the smart pointer stored in the [ConContainer](#) object.

Parameters

<i>vFrom</i>	An <code>std::vector<NeuronPtr></code> with the pointers to be set in the from fields of the ConContainer object.
--------------	---

Returns

true if not exception is thrown

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

```

//=====

// Data set up
std::vector<int> result;
ContainerNeuronPtr neuronContainerPtr( new
Container<Neuron>() );
ConContainerPtr conContainerPtr( new ConContainer() );
ConPtr ptC;
NeuronPtr ptN;

int ids[] = {10, 20, 30};
double weights[] = {1.13, 2.22, 3.33 };

for (int i=0; i<=2 ; i++) { // Let's
create a vector with three neurons
    ptN.reset( new Neuron( ids[i] ) );
    neuronContainerPtr->push_back( ptN );
}
for (int i=0; i<=2 ; i++) { // and a
vector with three connections
    ptC.reset( new Con() );
    conContainerPtr->push_back( ptC );
}
// Test
conContainerPtr->setFrom(neuronContainerPtr->load() );
conContainerPtr->show();
result=conContainerPtr->getId();

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

```

See also

getFrom and the unit test files, e.g. runit.Cpp.ConContainer.R, for further examples.

Definition at line 333 of file ConContainer.cpp.

References Container< T >::begin(), Container< T >::empty(), Container< Con >::size(), and Container< T >::size().

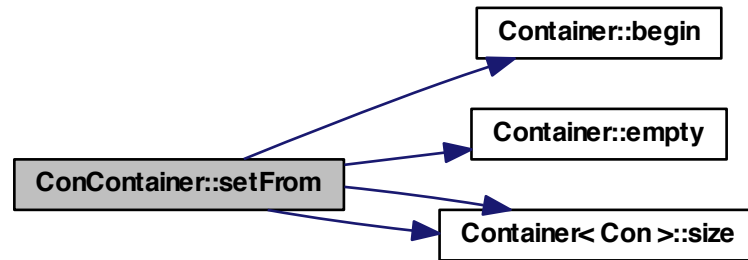
```

{
BEGIN_RCPP
if (neuronContainer.empty())
{ throw std::range_error("[ C++ ConContainer::setFrom]: Error, w is empty");}

if (neuronContainer.size() != size())
{
throw std::range_error(
"[C++ ConContainer::setFrom]: Error, neuronContainer.size() != collecti
on.size()");
}
std::vector<NeuronPtr>::iterator itrNeuron = neuronContainer.begin();
foreach(ConPtr itr , *this)
{
itr->setFrom( *itrNeuron );
itrNeuron++;
}
return true;
END_RCPP}

```

Here is the call graph for this function:



5.9.4.6 `bool ConContainer::setWeight (std::vector< double > nWeights, std::vector< int > nIds)`

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

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

Parameters

<i>nWeights</i>	A numeric (double) vector with the weights to be set in the Con objects contained in the ConContainer object.
<i>vFrom</i>	An <code>std::vector<int></code> with the lds of the connections to select

Returns

true in case no exception is thrown

```

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

// Data set up
std::vector<double> result;
std::vector<NeuronPtr> neuronContainer;
ConContainerPtr conContainerPtr( new ConContainer() );
ConPtr ptC;
NeuronPtr ptN;
int ids[] = {11, 10, 9, 3, 4, 5, 6, 7, 8, 2, 1};
double weights[] = {11.32, 1.26, 2.14, 3.16, 4.14, 5.19, 6.18, 7.16, 8.14, 9.12, 10.31};
std::vector<double> nWeights;
for (int i=0; i<11; i++) {
    nWeights.push_back(weights[i]);
}
  
```



```

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

);

    std::vector<int> toSelect;
    std::vector<double> vNewWeights;
    toSelect.push_back(1);
    toSelect.push_back(3);
    toSelect.push_back(5);
    toSelect.push_back(7);
    vNewWeights.push_back(1000.1);
    vNewWeights.push_back(3000.3);
    vNewWeights.push_back(5000.5);
    vNewWeights.push_back(7000.7);
    conContainerPtr->setWeight(vNewWeights, toSelect);

// Test

    result = conContainerPtr->getWeight();
    return wrap(result);

// Now, result is a numeric vector with the values 1000.10, 9.12, 3000.3
0, 4.14, 5000.50, 6.18, 7000.70, 8.14, 2.14, 1.26 and 11.32 .

```

See also

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

Definition at line 627 of file `ConContainer.cpp`.

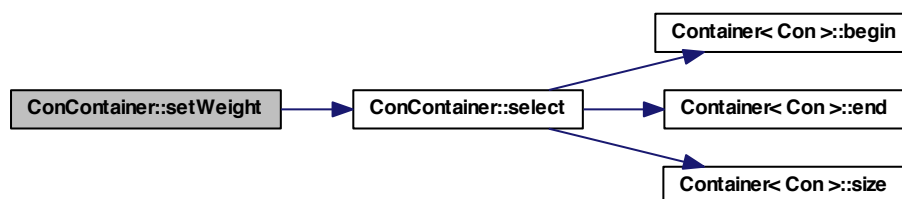
References `select()`.

```

{
BEGIN_RCPP return select(nIds)->setWeight(nWeights);
END_RCPP
}

```

Here is the call graph for this function:



5.9.4.7 bool ConContainer::setWeight (std::vector< double > nWeights)

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

This function provides a convenient way of setting the values of the weight field of those [Con](#) objects pointed to by the smart pointer stored in the [ConContainer](#) object.

Parameters

<i>nWeights</i>	A numeric (double) vector with the weights to be set in the Con objects contained in the ConContainer object.
-----------------	---

Returns

true in case no exception is thrown

```
//=====
//Usage example:
//=====
// Data set up
std::vector<double> result;
int ids[] = {1, 2, 3};
double weights[] = {12.3, 1.2, 2.1 };
ConContainer conContainer;
std::vector<NeuronPtr> neuronContainer;
std::vector<double> nWeights;
NeuronPtr ptNeuron;

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

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

// Test
conContainer.setWeight(nWeights);
/ weights are set to 12.3, 1.2 and 2.1
result=conContainer.getWeight();

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

See also

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

Definition at line 270 of file `ConContainer.cpp`.

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

```
{
    BEGIN_RCPP
```

```

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

```

Here is the call graph for this function:



5.9.4.8 bool ConContainer::validate () [virtual]

Object validator.

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

Returns

true in case the checks are Ok.

Exceptions

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

See also

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

Implements [Container< Con >](#).

Definition at line 645 of file `ConContainer.cpp`.

References getId().

```
{
    BEGIN_RCPP

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

Here is the call graph for this function:



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

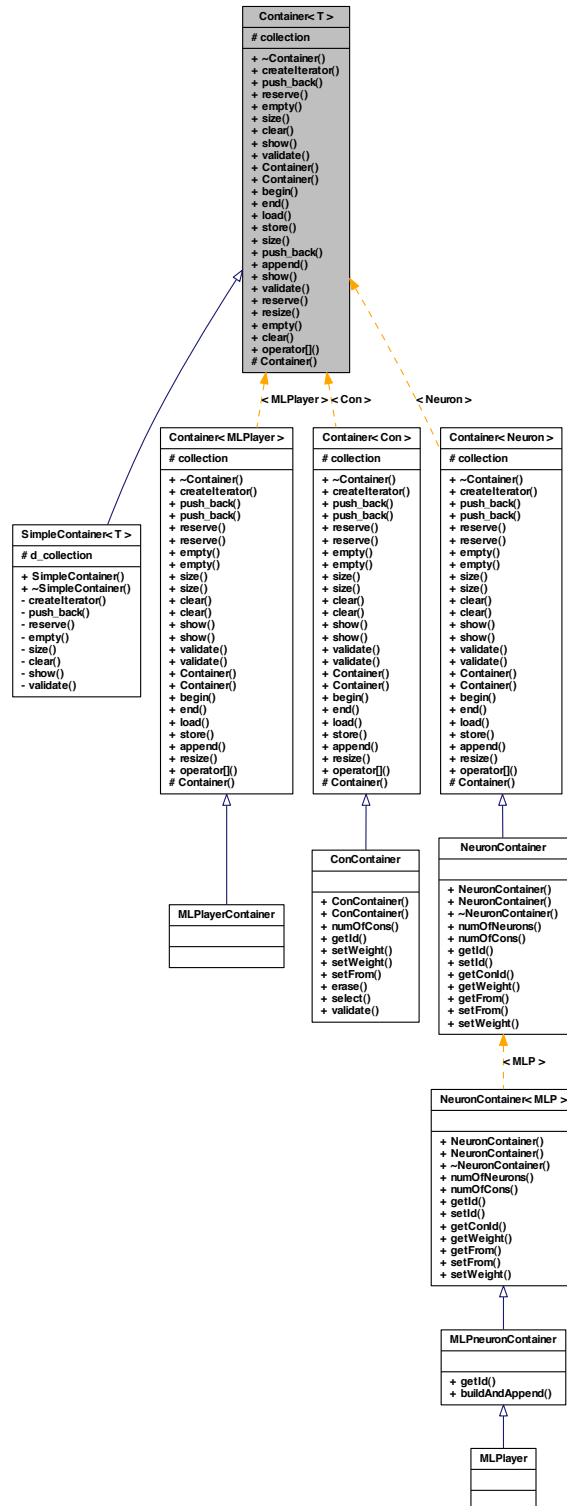
- pkg/AMORE/src/old/[ConContainer.h](#)
- pkg/AMORE/src/old/[ConContainer.cpp](#)

5.10 Container< T > Class Template Reference

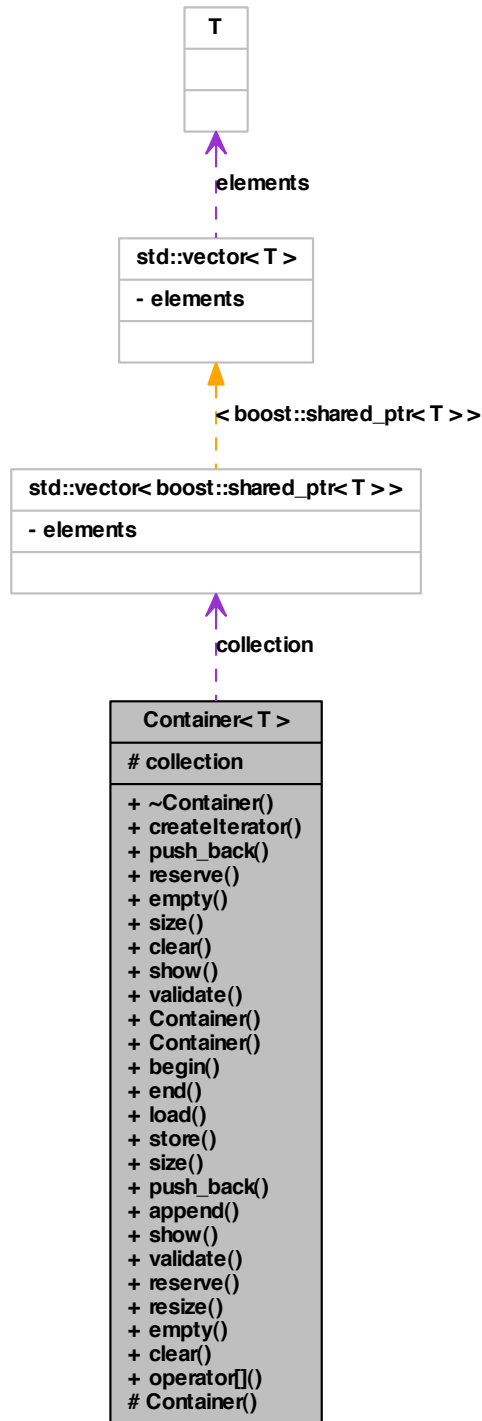
class [Container](#) -

```
#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
- typedef boost::shared_ptr< T > value_type
- typedef value_type const & const_reference

Public Member Functions

- virtual ~Container ()
- virtual boost::shared_ptr< iterator< T > > createIterator ()=0
- virtual void push_back (T const &const_reference)=0
 - Append a shared_ptr at the end of collection.*
- virtual void reserve (int n)=0
- virtual bool empty ()=0
- virtual size_type size ()=0
 - Returns the size or length of the vector.*
- virtual void clear ()=0
- virtual void show ()=0
 - Pretty print of the Container<T>*
- virtual bool validate ()=0
 - Object validator.*
- Container ()
- Container (std::vector< value > first, std::vector< value > last)
- iterator begin ()
- iterator end ()
- std::vector< boost::shared_ptr< T > > load ()
 - collection field accessor function*
- void store (typename std::vector< boost::shared_ptr< T > > collectionT)
 - collection field accessor function*
- size_type size ()
- void push_back (boost::shared_ptr< T > const &const_reference)
 - Append a shared_ptr at the end of collection.*
- void append (Container< T > containerT)
 - Appends a Container<T> object.*
- bool show ()
- bool validate ()
- void reserve (int n)
- void resize (int n)
- bool empty ()
- void clear ()
- boost::shared_ptr< T > & operator[] (size_type offset)

Protected Member Functions

- [Container](#) ()

Protected Attributes

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

5.10.1 Detailed Description

`template<typename T>class Container< T >`

class [Container](#) -

Definition at line 5 of file Container.h.

5.10.2 Member Typedef Documentation

5.10.2.1 `template<typename T> typedef std::vector<boost::shared_ptr<T>
>::const_iterator Container< T >::const_iterator`

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

Definition at line 22 of file Container.h.

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

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

Definition at line 26 of file Container.h.

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

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

Definition at line 19 of file Container.h.

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

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

Definition at line 24 of file Container.h.

5.10.3 Constructor & Destructor Documentation

5.10.3.1 `template<typename T> Container< T >::~~Container ()` [virtual]

Definition at line 17 of file Container.cpp.

```
{
}
```

5.10.3.2 `template<typename T> Container< T >::Container ()` [protected]

Definition at line 11 of file Container.cpp.

```
{
}
```

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

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

5.10.4 Member Function Documentation

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

Appends a Container<T> object.

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

Parameters

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

See also

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

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

Container<Neuron>() );
Container<Con>() );
Container<Con>() );

std::vector<int> result;
std::vector<ConPtr> vcA, vcB;
ContainerNeuronPtr neuronContainerPtr( new
ContainerConPtr conContainerPtrA( new
ContainerConPtr conContainerPtrB( new
```

```

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

for (int i=0; i<=5 ; i++) {
/ Let's create a vector with six neurons
    ptN.reset( new Neuron( ids[i] ) );
    neuronContainerPtr->push_back(ptN);
}
for (int i=0; i<=2 ; i++) {
/ A vector with three connections
    ptC.reset( new Con( neuronContainerPtr->load().at(i), weights[i] ) );
    conContainerPtrA->push_back(ptC);
}
for (int i=3; i<=5 ; i++) {
/ Another vector with three connections
    ptC.reset( new Con( neuronContainerPtr->load().at(i), weights[i] ) );
    conContainerPtrB->push_back(ptC);
}

// Test
conContainerPtrA->append(*conContainerPtrB);
conContainerPtrA->validate();
conContainerPtrA->show();

// After execution of the code above, the output at the R terminal would
display:
//
// From:      1      Weight=      1.130000
//      From:      2      Weight=      2.220000
//      From:      3      Weight=      3.330000
//      From:      4      Weight=      5.600000
//      From:      5      Weight=      4.200000
//      From:      6      Weight=      3.600000

```

See also

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

Definition at line 207 of file `Container.cpp`.

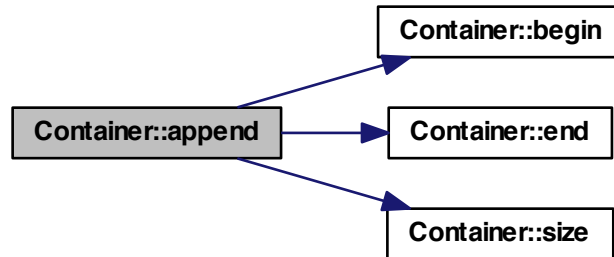
References `Container< T >::begin()`, `Container< T >::end()`, and `Container< T >::size()`.

```

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

```

Here is the call graph for this function:



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

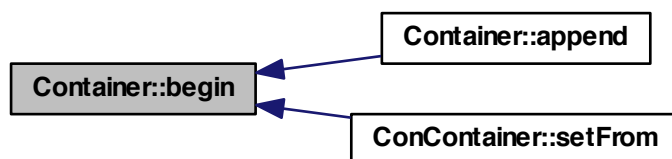
Definition at line 22 of file `Container.cpp`.

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

```

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

Here is the caller graph for this function:



5.10.4.3 `template<typename T> void Container< T >::clear () [pure virtual]`

Implemented in [SimpleContainer< T >](#).

Definition at line 177 of file Container.cpp.

```
{
    collection.clear();
}
```

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

Reimplemented in [SimpleContainer< T >](#).

5.10.4.5 `template<typename T> boost::shared_ptr< IteratorInterface< T > > Container< T >::createIterator ()` [pure virtual]

Implemented in [SimpleContainer< T >](#).

Definition at line 23 of file Container.cpp.

```
{
    boost::shared_ptr< ContainerIterator<T> > containerIteratorPtr( new Container
        Iterator<T> ());
    containerIteratorPtr->d_container = this;
    containerIteratorPtr->d_iterator = collection.begin();
    return containerIteratorPtr;
}
```

5.10.4.6 `template<typename T> bool Container< T >::empty ()` [pure virtual]

Implemented in [SimpleContainer< T >](#).

Definition at line 163 of file Container.cpp.

Referenced by `ConContainer::setFrom()`.

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

Here is the caller graph for this function:



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

Reimplemented in [SimpleContainer< T >](#).

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

Definition at line 29 of file Container.cpp.

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

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

Here is the caller graph for this function:



5.10.4.9 `template<typename T> std::vector< boost::shared_ptr< T > > Container< T
>::load ()`

collection field accessor function

This method allows access to the data stored in the [collection](#) field.

Returns

The collection vector.

```
//=====
//Usage example:
//=====
// Data set up
std::vector<int> result;
std::vector<ConPtr> vcA, vcB;
ContainerNeuronPtr neuronContainerPtr( new
Container<Neuron>() );
ContainerConPtr conContainerPtr( new
Container<Con>() );
ConPtr ptC;
NeuronPtr ptN;
int ids[]= {10, 20, 30};
```

```

double weights[] = {1.13, 2.22, 3.33 };
for (int i=0; i<=2 ; i++) {
/ Let's create a vector with three neurons
    ptN.reset( new Neuron( ids[i] ) );
    neuronContainerPtr->push_back(ptN);
}
for (int i=0; i<=2 ; i++) {
/ and a vector with three connections
    ptC.reset( new Con( neuronContainerPtr->load().at(i), weights[i] ) );
    vcA.push_back(ptC);
}
// Test
conContainerPtr->store(vcA);
vcB = conContainerPtr->load();
for (int i=0; i<=2 ; i++) {
/ get Ids. Container does not have getId defined
    result.push_back( vcB.at(i)->getId());
}

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

```

See also

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

Definition at line 254 of file `Container.cpp`.

```

{
    return collection;
}

```

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

Definition at line 317 of file `Container.cpp`.

```

{
    return collection[offset];
}

```

5.10.4.11 `template<typename T> void Container< T>::push_back (T const & reference) [pure virtual]`

Append a `shared_ptr` at the end of collection.

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

Parameters

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

//=====

```

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

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

    vc = conContainer.load();

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

// After execution of this code, result contains a numeric vector with va
lues 10, 20 and 30.

```

See also

C++ documentation for `std::vector::push_back` and the unit test files, e.g., `runit.Cpp.Container.R`, for usage examples.

Implemented in [SimpleContainer< T >](#).

Definition at line 68 of file `Container.cpp`.

```

{
    collection.push_back(reference);
}

```

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

Append a `shared_ptr` at the end of collection.

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

Parameters

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

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

```

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

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

    vc = conContainer.load();

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

// After execution of this code, result contains a numeric vector with va
lues 10, 20 and 30.

```

See also

C++ documentation for `std::vector::push_back` and the unit test files, e.g., `runit.Cpp.Container.R`, for usage examples.

Definition at line 71 of file `Container.cpp`.

```

{
    collection.push_back(const_reference);
}

```

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

Reimplemented in [SimpleContainer< T >](#).

5.10.4.14 `template<typename T > void Container< T >::reserve (int n)` [pure virtual]

Implemented in [SimpleContainer< T >](#).

Definition at line 170 of file `Container.cpp`.

```

{

```



```

    collection.reserve(n);
}

```

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

Definition at line 289 of file Container.cpp.

```

{
    collection.resize(n);
}

```

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

Reimplemented in [SimpleContainer< T >](#).

5.10.4.17 `template<typename T> bool Container< T >::show () [pure virtual]`

Pretty print of the Container<T>

This method outputs in the R terminal the contents of [Container::collection](#).

Returns

true in case everything works without throwing an exception

*

```

//=====
//Usage example:
//=====
// Data set up
ContainerNeuronPtr      neuronContainerPtr( new
Container<Neuron>() );
ContainerConPtr conContainerPtr( new Container<Con>() );
ConPtr ptC;
NeuronPtr ptN;
int ids[]= {10, 20, 30};
double weights[] = {1.13, 2.22, 3.33 };

    for (int i=0; i<=2 ; i++) {
/ Let's create a vector with three neurons
        ptN.reset( new Neuron( ids[i] ) );
        neuronContainerPtr->push_back(ptN);
    }

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

// Test

```

```

conContainerPtr->show() ;

// The output at the R terminal would display:
//
//      # From:  10      Weight=      1.130000
//      # From:  20      Weight=      2.220000
//      # From:  30      Weight=      3.330000
//

```

See also

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

Implemented in [SimpleContainer< T >](#).

Definition at line 118 of file `Container.cpp`.

```

{
    for (typename std::vector<T>::iterator itr(collection.begin()); itr
        != collection.end(); ++itr)
    {
        itr->show();
    }
}

```

5.10.4.18 template<typename T> size_type Container< T >::size ()

Reimplemented in [SimpleContainer< T >](#).

5.10.4.19 template<typename T > size_type Container< T >::size () [pure virtual]

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.

Implemented in [SimpleContainer< T >](#).

Definition at line 155 of file `Container.cpp`.

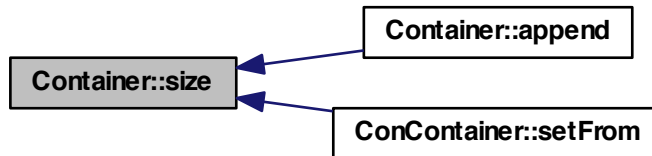
Referenced by `Container< T >::append()`, and `ConContainer::setFrom()`.

```

{
    return collection.size();
}

```

Here is the caller graph for this function:



5.10.4.20 `template<typename T> void Container< T >::store (typename std::vector< boost::shared_ptr< T > > collectionT)`

collection field accessor function

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

Parameters

<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 268 of file `Container.cpp`.

```

{
    collection = collectionT;
}

```

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

Reimplemented in [SimpleContainer< T >](#), and [ConContainer](#).

5.10.4.22 `template<typename T> bool Container< T >::validate () [pure virtual]`

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.

Implemented in [SimpleContainer< T >](#), and [ConContainer](#).

Definition at line 136 of file `Container.cpp`.

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

5.10.5 Member Data Documentation**5.10.5.1 `template<typename T> std::vector<boost::shared_ptr<T> > Container< T >::collection` [protected]**

Definition at line 15 of file `Container.h`.

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

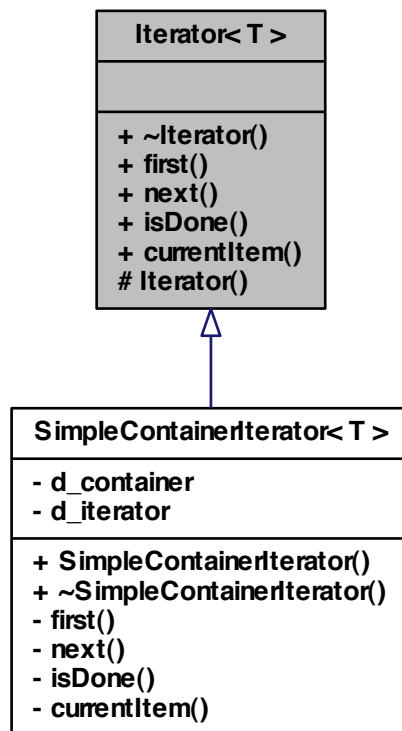
- `pkg/AMORE/src/dia/Container.h`
- `pkg/AMORE/src/old/Container.h`
- `pkg/AMORE/src/Container.cpp`
- `pkg/AMORE/src/old/Container.cpp`

5.11 `Iterator< T >` Class Template Reference

class [Iterator](#) -

```
#include <Iterator.h>
```

Inheritance diagram for Iterator< T >:



Public Member Functions

- virtual `~Iterator()`
- virtual void `first()`=0
- virtual void `next()`=0
- virtual bool `isDone()`=0
- virtual T `currentItem()`=0

Protected Member Functions

- `Iterator()`

5.11.1 Detailed Description

```
template<typename T>class Iterator< T >
```

class [Iterator](#) -

Definition at line 5 of file Iterator.h.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 `template<typename T > virtual Iterator< T >::~~Iterator () [virtual]`

5.11.2.2 `template<typename T > Iterator< T >::Iterator () [protected]`

5.11.3 Member Function Documentation

5.11.3.1 `template<typename T > virtual T Iterator< T >::currentItem () [pure virtual]`

Implemented in [SimpleContainerIterator< T >](#).

5.11.3.2 `template<typename T > virtual void Iterator< T >::first () [pure virtual]`

Implemented in [SimpleContainerIterator< T >](#).

5.11.3.3 `template<typename T > virtual bool Iterator< T >::isDone () [pure virtual]`

Implemented in [SimpleContainerIterator< T >](#).

5.11.3.4 `template<typename T > virtual void Iterator< T >::next () [pure virtual]`

Implemented in [SimpleContainerIterator< T >](#).

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

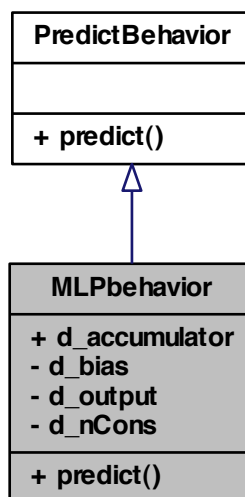
- pkg/AMORE/src/dia/[Iterator.h](#)

5.12 MLPbehavior Class Reference

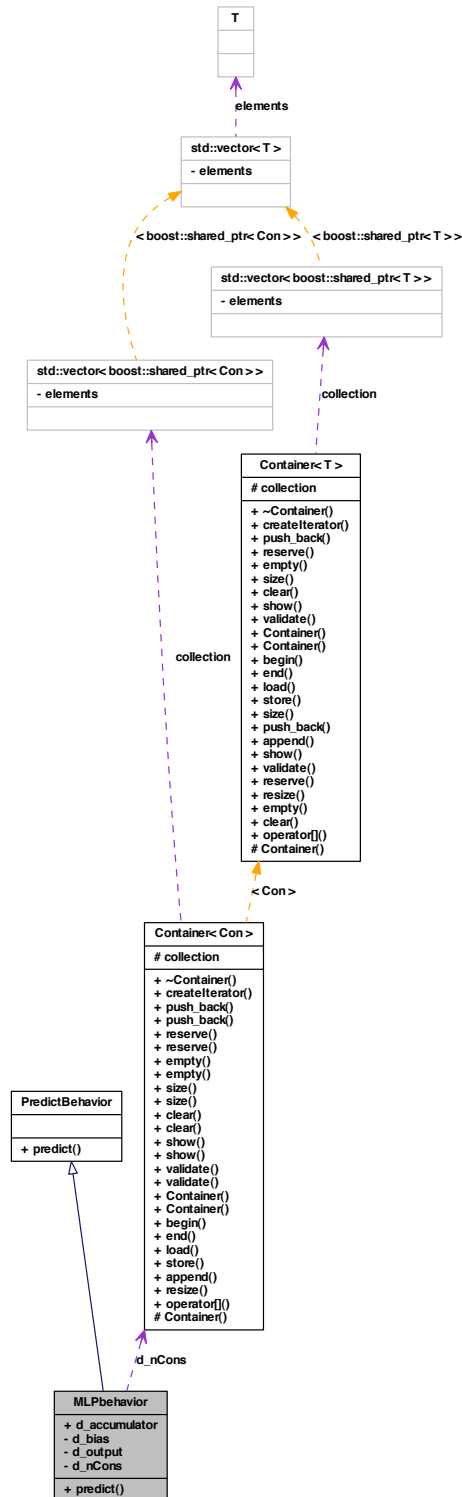
class [MLPbehavior](#) -

```
#include <MLPbehavior.h>
```

Inheritance diagram for MLPbehavior:



Collaboration diagram for MLPbehavior:



Public Member Functions

- void [predict](#) ()

Public Attributes

- double [d_accumulator](#)

Private Attributes

- double [d_bias](#)
- double [d_output](#)
- [Container](#)< [Con](#) > [d_nCons](#)

5.12.1 Detailed Description

class [MLPbehavior](#) -

Definition at line 5 of file MLPbehavior.h.

5.12.2 Member Function Documentation

5.12.2.1 void [MLPbehavior::predict](#) ()

Reimplemented from [PredictBehavior](#).

5.12.3 Member Data Documentation

5.12.3.1 double [MLPbehavior::d_accumulator](#)

Definition at line 8 of file MLPbehavior.h.

5.12.3.2 double [MLPbehavior::d_bias](#) [private]

Definition at line 10 of file MLPbehavior.h.

5.12.3.3 [Container](#)<[Con](#)> [MLPbehavior::d_nCons](#) [private]

Definition at line 12 of file MLPbehavior.h.

5.12.3.4 double MLPbehavior::d_output [private]

Definition at line 11 of file MLPbehavior.h.

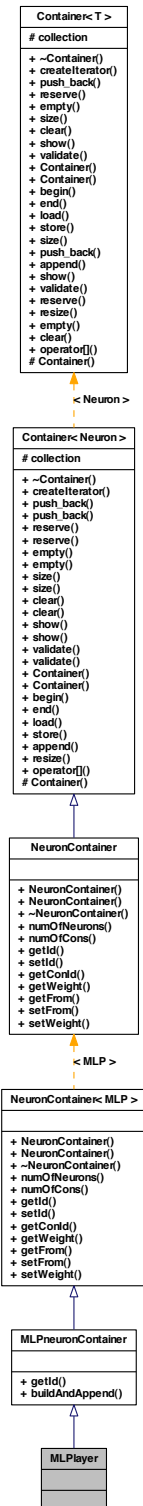
The documentation for this class was generated from the following file:

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

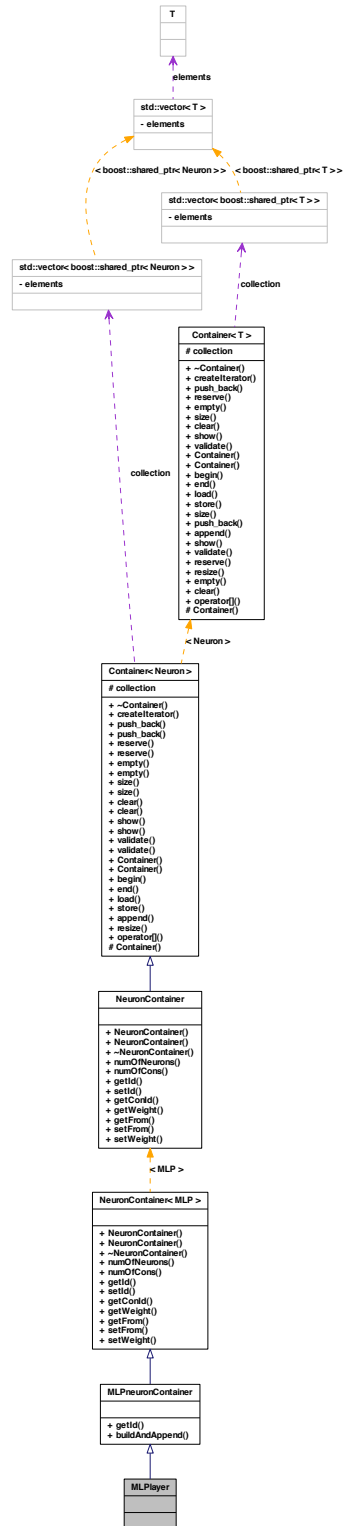
5.13 MLPlayer Class Reference

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

Inheritance diagram for MLPlayer:



Collaboration diagram for MLPlayer:



5.13.1 Detailed Description

Definition at line 1 of file MLPlayer.h.

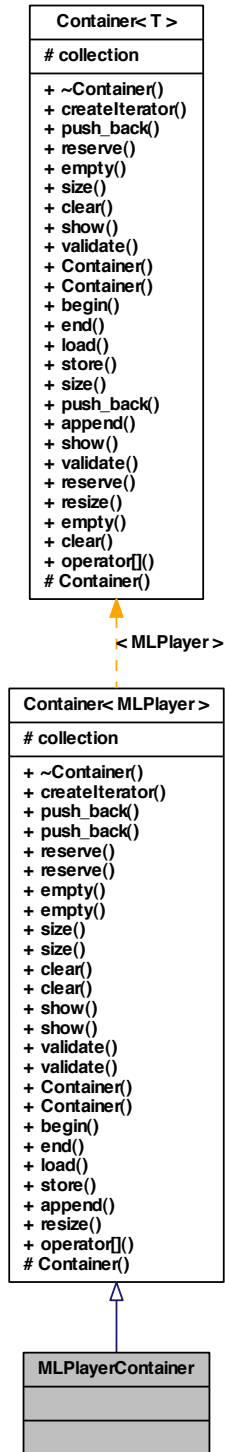
The documentation for this class was generated from the following file:

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

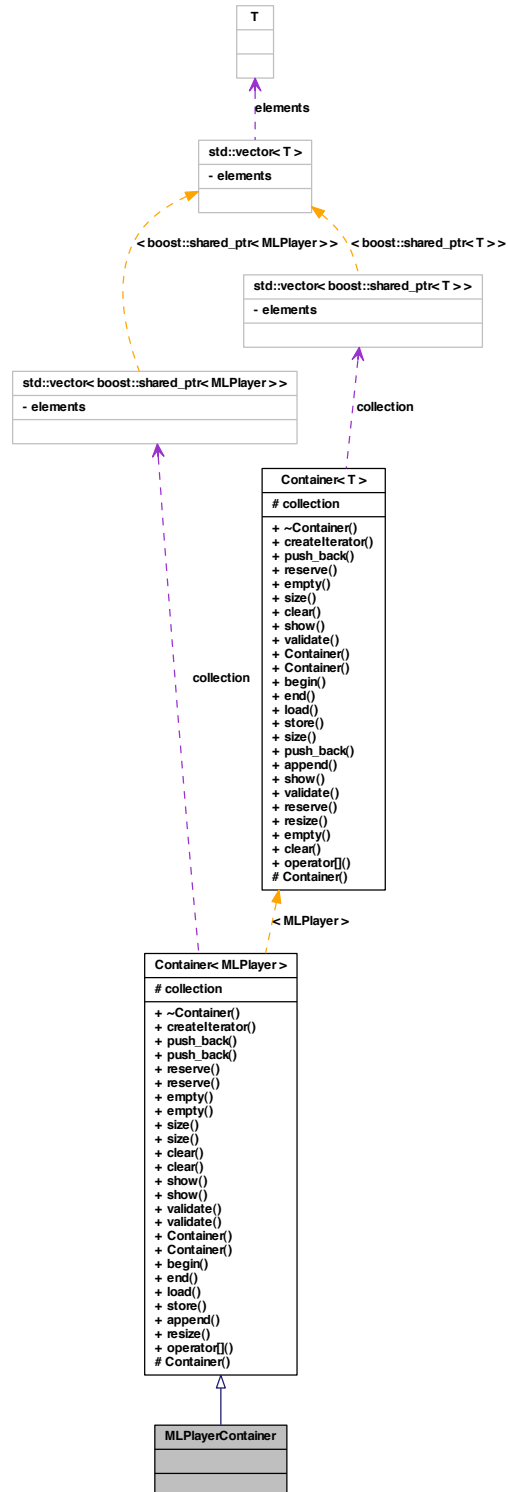
5.14 MLPlayerContainer Class Reference

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

Inheritance diagram for MLPlayerContainer:



Collaboration diagram for MLPlayerContainer:



5.14.1 Detailed Description

Definition at line 1 of file `MLPlayerContainer.h`.

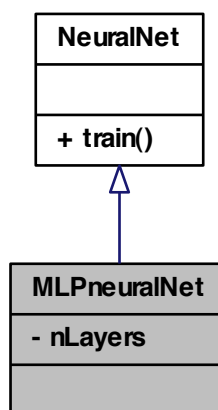
The documentation for this class was generated from the following file:

- `pkg/AMORE/src/old/MLPlayerContainer.h`

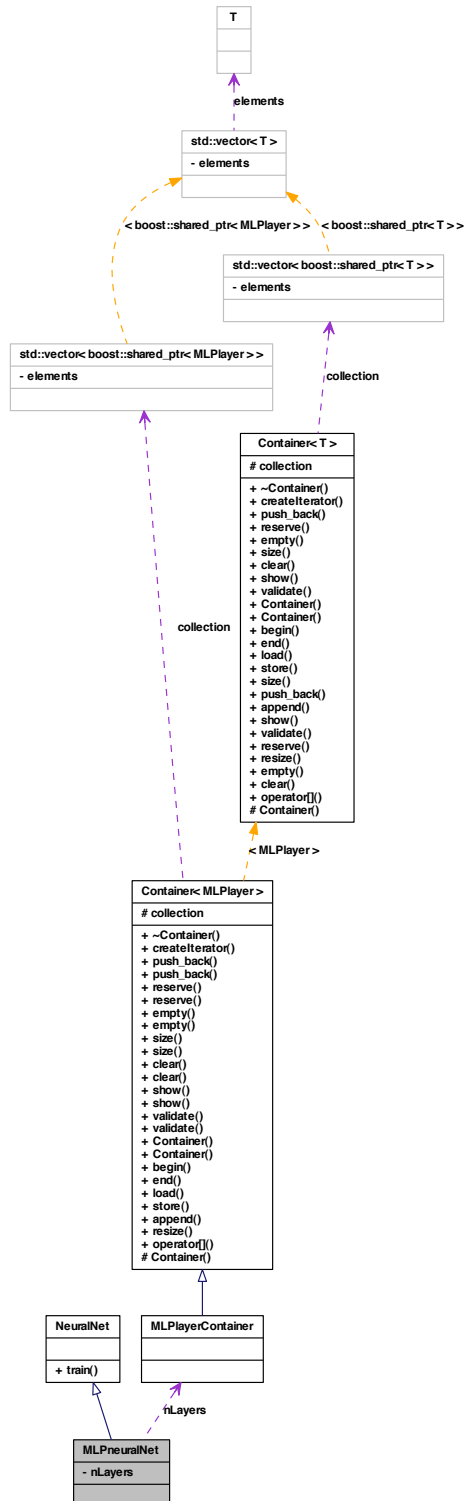
5.15 MLPneuralNet Class Reference

```
#include <MLPneuralNet.h>
```

Inheritance diagram for MLPneuralNet:



Collaboration diagram for MLPneuralNet:



Private Attributes

- [MLPlayerContainer nLayers](#)

5.15.1 Detailed Description

Definition at line 1 of file MLPneuralNet.h.

5.15.2 Member Data Documentation

5.15.2.1 **MLPlayerContainer MLPneuralNet::nLayers** `[private]`

Definition at line 2 of file MLPneuralNet.h.

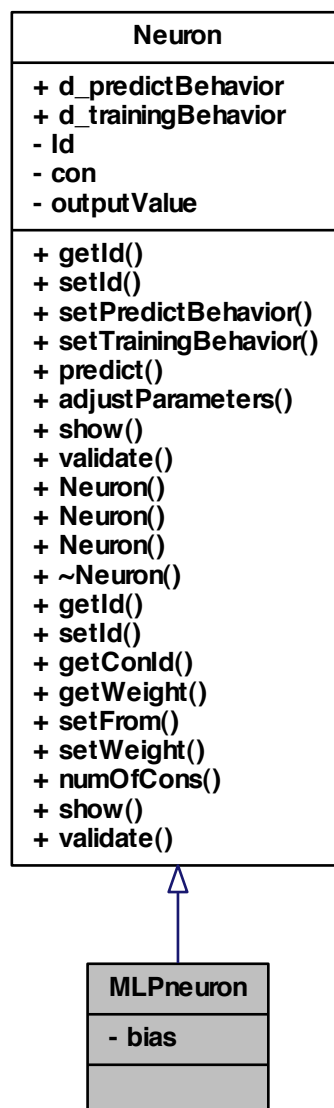
The documentation for this class was generated from the following file:

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

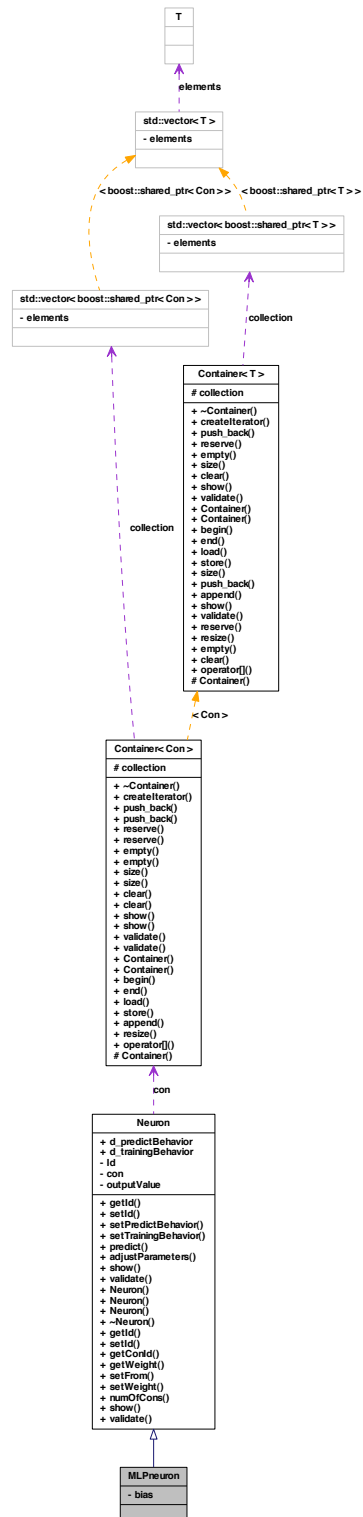
5.16 MLPneuron Class Reference

```
#include <MLPneuron.h>
```

Inheritance diagram for MLPneuron:



Collaboration diagram for MLPNeuron:



Private Attributes

- int [bias](#)

5.16.1 Detailed Description

Definition at line 1 of file MLPneuron.h.

5.16.2 Member Data Documentation

5.16.2.1 int MLPneuron::bias [private]

Definition at line 2 of file MLPneuron.h.

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

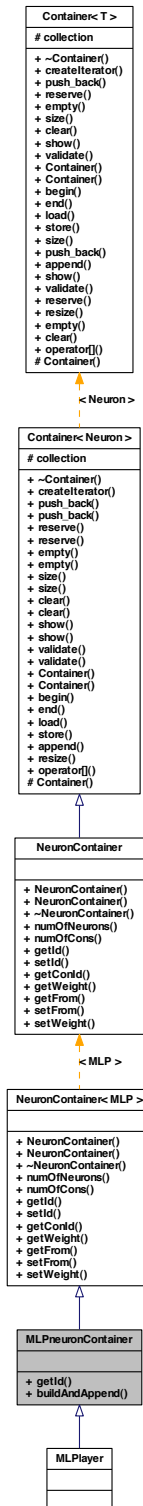
- pkg/AMORE/src/old/[MLPneuron.h](#)

5.17 MLPneuronContainer Class Reference

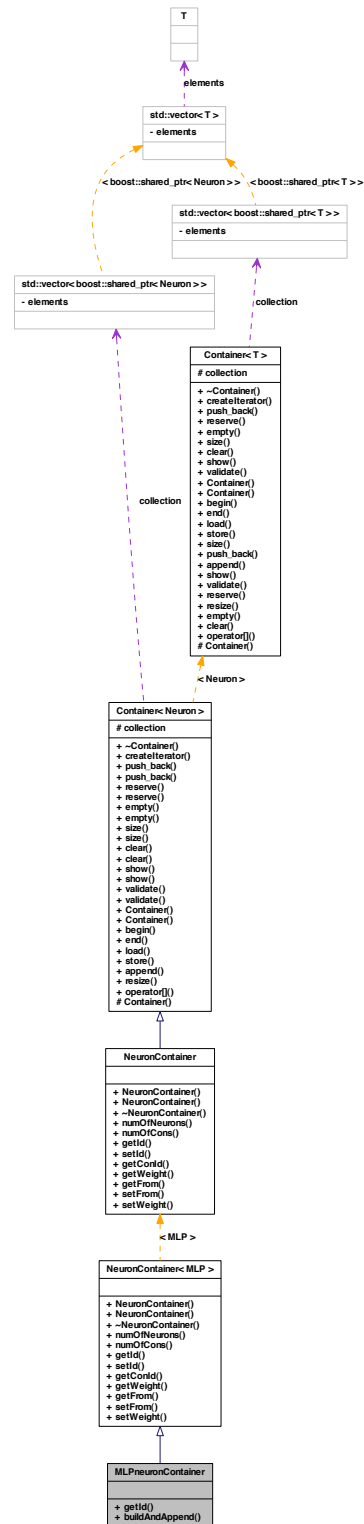
A vector of connections.

```
#include <MLPneuronContainer.h>
```

Inheritance diagram for MLPneuronContainer:



Collaboration diagram for MLPNeuronContainer:



Public Member Functions

- `std::vector< int > getId ()`
- `bool buildAndAppend (std::vector< int > IDS, std::vector< int > BIAS, ConContainer VC)`

5.17.1 Detailed Description

A vector of connections.

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

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

5.17.2 Member Function Documentation

5.17.2.1 `bool MLPneuronContainer::buildAndAppend (std::vector< int > IDS, std::vector< int > BIAS, ConContainer VC)`

5.17.2.2 `std::vector<int> MLPneuronContainer::getId ()`

Reimplemented from [NeuronContainer< MLP >](#).

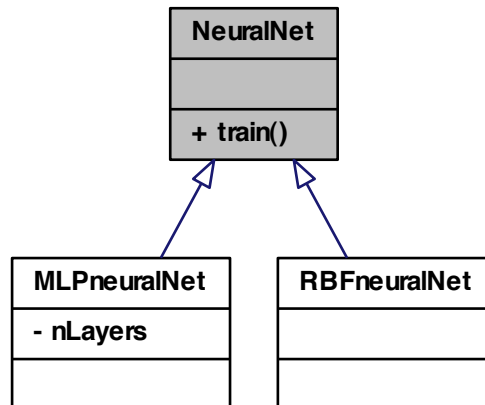
The documentation for this class was generated from the following file:

- `pkg/AMORE/src/old/MLPneuronContainer.h`

5.18 NeuralNet Class Reference

```
#include <NeuralNet.h>
```


Inheritance diagram for NeuralNet:



Public Member Functions

- virtual void [train](#) ()=0

5.18.1 Detailed Description

Definition at line 1 of file NeuralNet.h.

5.18.2 Member Function Documentation

5.18.2.1 virtual void NeuralNet::train () [pure virtual]

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

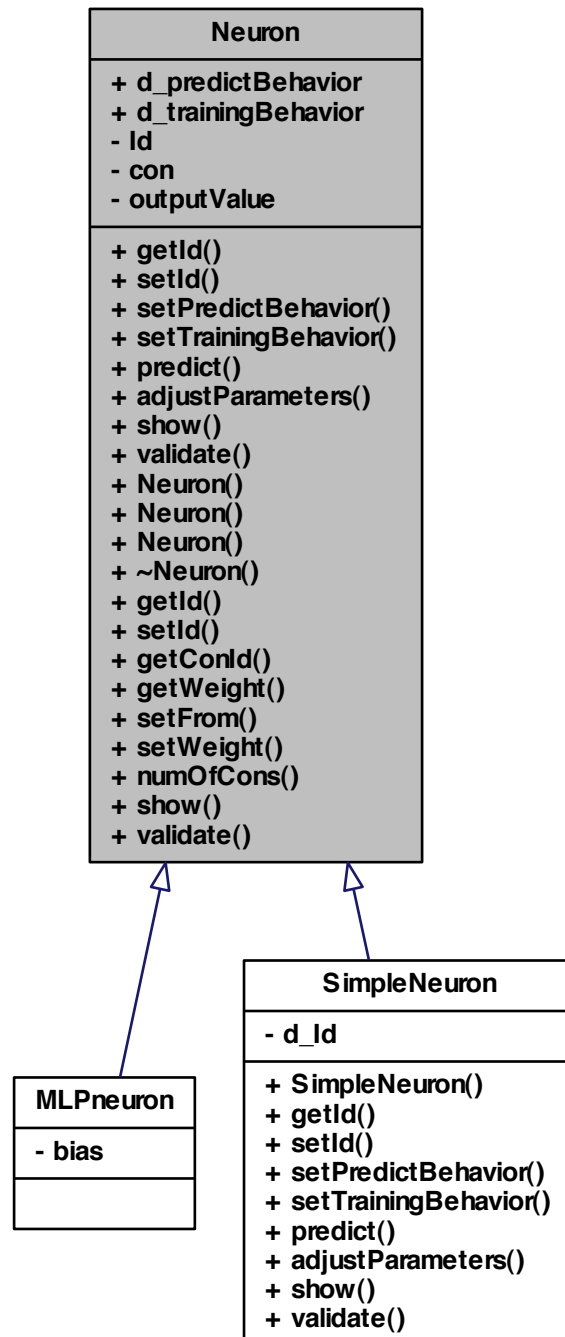
- pkg/AMORE/src/old/[NeuralNet.h](#)

5.19 Neuron Class Reference

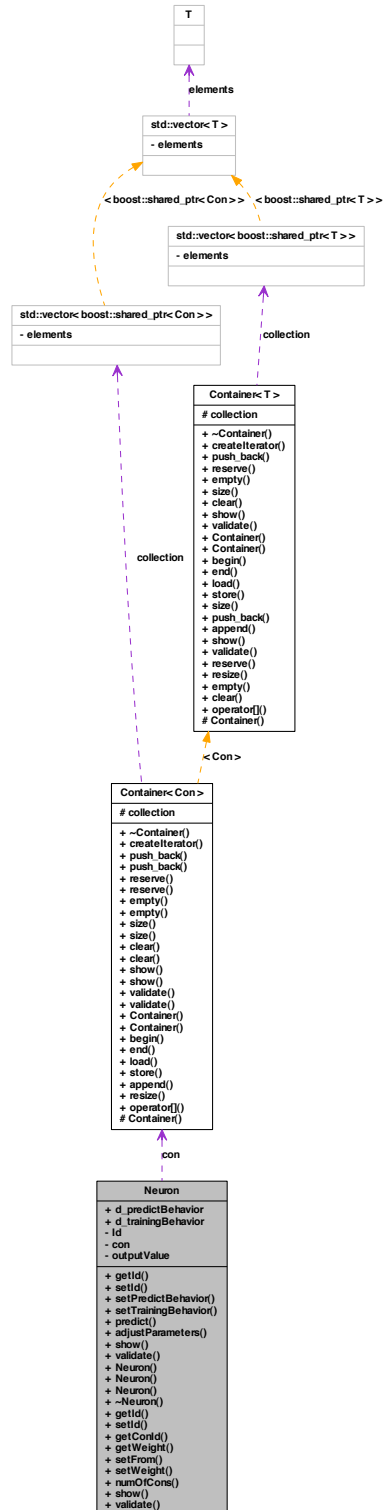
class [Neuron](#) -

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

Inheritance diagram for Neuron:



Collaboration diagram for Neuron:



Public Member Functions

- virtual [Handler](#) [getId](#) ()=0
- virtual void [setId](#) ([Handler Id](#))=0
- virtual void [setPredictBehavior](#) ([PredictBehavior](#) &predictBehavior)=0
- virtual void [setTrainingBehavior](#) ([TrainingBehavior](#) &trainingBehavior)=0
- virtual void [predict](#) ()=0
- virtual void [adjustParameters](#) ()=0
- virtual void [show](#) ()()=0
- virtual bool [validate](#) ()=0
- [Neuron](#) ()
- [Neuron](#) (int [Id](#))
- [Neuron](#) (int [Id](#), [ConContainer](#) con)
- \sim [Neuron](#) ()
- int [getId](#) ()
- void [setId](#) (int value)
- std::vector< int > [getConId](#) ()
- std::vector< double > [getWeight](#) ()
- bool [setFrom](#) ([NeuronContainer](#) neuronContainer)
- bool [setWeight](#) (std::vector< double > nWeights)
- int [numOfCons](#) ()
- bool [show](#) ()
- bool [validate](#) ()

Public Attributes

- [PredictBehaviorRef](#) [d_predictBehavior](#)
- [TrainingBehaviorRef](#) [d_trainingBehavior](#)

Private Attributes

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

5.19.1 Detailed Description

class [Neuron](#) -

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

A general class for neurons. The [MLPneuron](#) and [RBFneuron](#) classes will specialize this general class

Definition at line 3 of file [Neuron.h](#).

5.19.2 Constructor & Destructor Documentation

5.19.2.1 Neuron::Neuron ()

Definition at line 10 of file Neuron.cpp.

```
        :  
    d_Id(NA_INTEGER) //, nCons()  
{  
}
```

5.19.2.2 Neuron::Neuron (int *Id*)

Definition at line 15 of file Neuron.cpp.

```
        :  
    d_Id(Id) //, nCons()  
{  
}
```

5.19.2.3 Neuron::Neuron (int *Id*, ConContainer *con*)

5.19.2.4 Neuron::~Neuron ()

5.19.3 Member Function Documentation

5.19.3.1 virtual void Neuron::adjustParameters () [pure virtual]

Implemented in [SimpleNeuron](#).

5.19.3.2 std::vector<int> Neuron::getConId ()

5.19.3.3 int Neuron::getId ()

Reimplemented in [SimpleNeuron](#).

5.19.3.4 int Neuron::getId () [pure virtual]

Implemented in [SimpleNeuron](#).

Definition at line 26 of file Neuron.cpp.

References *Id*.

```
{  
    return Id;  
}
```

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

5.19.3.6 `int Neuron::numOfCons ()`

5.19.3.7 `virtual void Neuron::predict ()` [pure virtual]

Implemented in [SimpleNeuron](#).

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

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

Reimplemented in [SimpleNeuron](#).

5.19.3.10 `void Neuron::setId (Handler Id)` [pure virtual]

Implemented in [SimpleNeuron](#).

Definition at line 32 of file Neuron.cpp.

References `Id`.

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

5.19.3.11 `virtual void Neuron::setPredictBehavior (PredictBehavior & predictBehavior)`
[pure virtual]

Implemented in [SimpleNeuron](#).

5.19.3.12 `virtual void Neuron::setTrainingBehavior (TrainingBehavior & trainingBehavior)`
[pure virtual]

Implemented in [SimpleNeuron](#).

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

5.19.3.14 `bool Neuron::show ()` [pure virtual]

Implemented in [SimpleNeuron](#).

Definition at line 54 of file Neuron.cpp.

References `Id`.

```

{
    int id = Id();
    Rprintf("\n-----\n");
    if (id == NA_INTEGER)
    {
        Rprintf("\n Id: NA, Invalid neuron Id");
    }
    else
    {
        Rprintf("\n Id: %d", id);
    }
    Rprintf("\n-----\n");
#ifdef 0

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

```

5.19.3.15 bool Neuron::show ()

Reimplemented in [SimpleNeuron](#).

5.19.3.16 bool Neuron::validate () [pure virtual]

Implemented in [SimpleNeuron](#).

Definition at line 82 of file Neuron.cpp.

References [Id](#).

```

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

```

5.19.3.17 bool Neuron::validate ()

Reimplemented in [SimpleNeuron](#).

5.19.4 Member Data Documentation

5.19.4.1 **ConContainer Neuron::con** [private]

A vector of input connections.

Definition at line 29 of file Neuron.h.

5.19.4.2 **PredictBehaviorRef Neuron::d_predictBehavior**

Definition at line 6 of file Neuron.h.

5.19.4.3 **TrainingBehaviorRef Neuron::d_trainingBehavior**

Definition at line 7 of file Neuron.h.

5.19.4.4 **int Neuron::ld** [private]

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

The [Neuron](#) ld 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 22 of file Neuron.h.

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

5.19.4.5 **double Neuron::outputValue** [private]

Definition at line 30 of file Neuron.h.

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

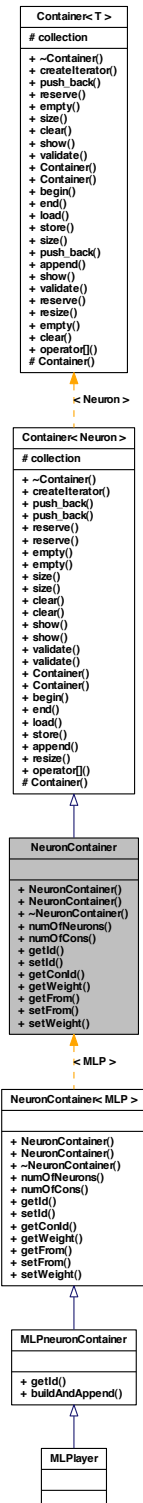
- `pkg/AMORE/src/dia/Neuron.h`
- `pkg/AMORE/src/old/Neuron.h`
- `pkg/AMORE/src/Neuron.cpp`
- `pkg/AMORE/src/old/Neuron.cpp`

5.20 NeuronContainer Class Reference

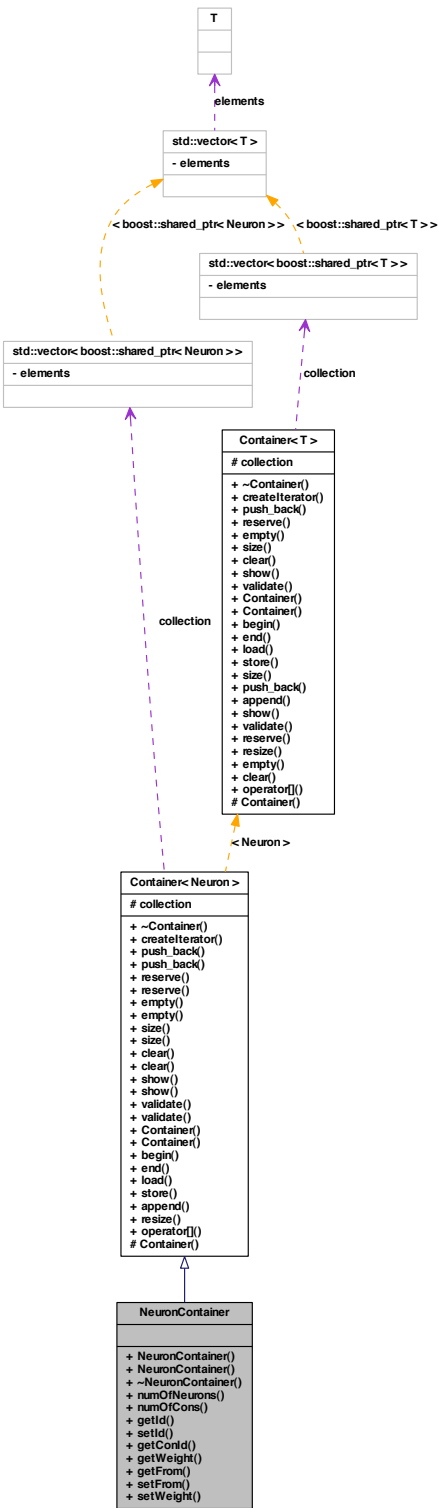
A vector of neurons.

```
#include <NeuronContainer.h>
```


Inheritance diagram for NeuronContainer:



Collaboration diagram for NeuronContainer:



Public Types

- typedef NeuronContainer_iterator [iterator](#)
- typedef NeuronContainer_const_iterator [const_iterator](#)
- typedef boost::shared_ptr< [Neuron](#) > [value_type](#)
- typedef [value_type](#) const & [const_reference](#)

Public Member Functions

- [NeuronContainer](#) ()
- [NeuronContainer](#) (std::vector< [NeuronPtr](#) > neuronContainer)
- [~NeuronContainer](#) ()
- int [numOfNeurons](#) ()
- std::vector< int > [numOfCons](#) ()
- std::vector< int > [getId](#) ()
- void [setId](#) (std::vector< int > nlds)
- std::vector< std::vector< int > > [getConId](#) ()
- std::vector< std::vector< double > > [getWeight](#) ()
- std::vector< [NeuronContainer](#) > [getFrom](#) ()
- void [setFrom](#) (std::vector< [NeuronContainer](#) > neuronArray)
- void [setWeight](#) (std::vector< std::vector< double > > value)

5.20.1 Detailed Description

A vector of neurons.

The vecNeuron class provides a simple class for a vector of neurons. It's named after the R equivalent Reference Class.

Definition at line 17 of file NeuronContainer.h.

5.20.2 Member Typedef Documentation

5.20.2.1 typedef NeuronContainer_const_iterator NeuronContainer::const_iterator

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

Definition at line 23 of file NeuronContainer.h.

5.20.2.2 typedef value_type const& NeuronContainer::const_reference

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

Definition at line 27 of file NeuronContainer.h.

5.20.2.3 `typedef NeuronContainer_iterator NeuronContainer::iterator`

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

Definition at line 21 of file NeuronContainer.h.

5.20.2.4 `typedef boost::shared_ptr<Neuron> NeuronContainer::value_type`

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

Definition at line 25 of file NeuronContainer.h.

5.20.3 Constructor & Destructor Documentation

5.20.3.1 `NeuronContainer::NeuronContainer ()`

Definition at line 8 of file NeuronContainer.cpp.

```
{
}
```

5.20.3.2 `NeuronContainer::NeuronContainer (std::vector< NeuronPtr > neuronContainer)`

Definition at line 12 of file NeuronContainer.cpp.

```
Container<Neuron> (collection)
{
}
:
```

5.20.3.3 `NeuronContainer::~~NeuronContainer ()`

Definition at line 17 of file NeuronContainer.cpp.

```
{
}
```

5.20.4 Member Function Documentation

5.20.4.1 `std::vector< std::vector< int > > NeuronContainer::getConld ()`

Definition at line 60 of file NeuronContainer.cpp.

```
{
    std::vector < std::vector<int> > result;
    foreach(NeuronPtr itrNeuron, *this)
```

```
{
    result.push_back( itrNeuron->getConId() );
}
return result;
}
```

5.20.4.2 `std::vector<NeuronContainer> NeuronContainer::getFrom ()`

5.20.4.3 `std::vector<int> NeuronContainer::getId ()`

Reimplemented in [MLPNeuronContainer](#).

Definition at line 39 of file NeuronContainer.cpp.

```
{
    std::vector<int> nIds;
    foreach(NeuronPtr itrNeuron, *this)
    {
        nIds.push_back( itrNeuron->getId() );
    }
    return nIds;
}
```

5.20.4.4 `std::vector< std::vector<double> > NeuronContainer::getWeight ()`

Definition at line 71 of file NeuronContainer.cpp.

```
{
    std::vector < std::vector<double> > result;
    foreach(NeuronPtr itrNeuron, *this)
    {
        result.push_back( itrNeuron->getWeight() );
    }
    return result;
}
```

5.20.4.5 `std::vector<int> NeuronContainer::numOfCons ()`

Definition at line 28 of file NeuronContainer.cpp.

```
{
    std::vector<int> nIds;
    foreach(NeuronPtr itrNeuron, *this)
    {
        nIds.push_back( itrNeuron->numOfCons() );
    }
    return nIds;
}
```

5.20.4.6 int NeuronContainer::numOfNeurons ()

Definition at line 22 of file NeuronContainer.cpp.

References Container< Neuron >::size().

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

Here is the call graph for this function:



5.20.4.7 void NeuronContainer::setFrom (std::vector< NeuronContainer > neuronArray)

Definition at line 83 of file NeuronContainer.cpp.

```
{
    std::vector<NeuronContainer>::iterator itrArray(neuronArray.begin());
    foreach(NeuronPtr itrNeuron, *this)
    {
        itrNeuron->setFrom(*itrArray);
        itrArray++;
    }
}
```

5.20.4.8 void NeuronContainer::setId (std::vector< int > nIds)

Definition at line 50 of file NeuronContainer.cpp.

```
{
    std::vector<int>::iterator itrId(nIds.begin());
    foreach(NeuronPtr itrNeuron, *this)
    {
        itrNeuron->setId(*itrId);
    }
}
```

5.20.4.9 void NeuronContainer::setWeight (std::vector< std::vector< double > > value)

Definition at line 94 of file NeuronContainer.cpp.

```
{
    std::vector<std::vector<double> >::iterator itrValue(value.begin());
    foreach(NeuronPtr itrNeuron, *this)
    {
        itrNeuron->setWeight(*itrValue);
        itrValue++;
    }
}
```

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

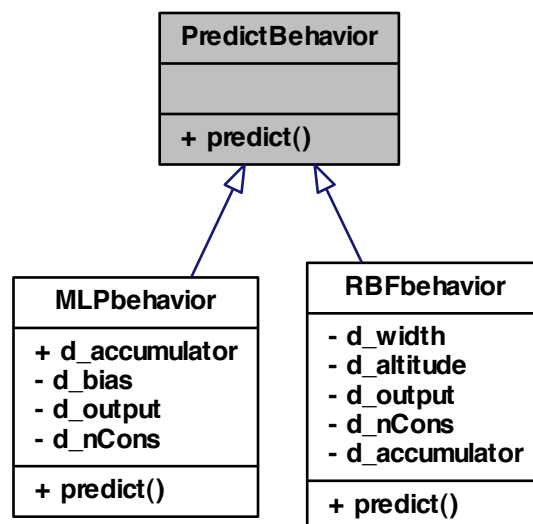
- pkg/AMORE/src/old/[NeuronContainer.h](#)
- pkg/AMORE/src/old/[NeuronContainer.cpp](#)

5.21 PredictBehavior Class Reference

class [PredictBehavior](#) -

```
#include <PredictBehavior.h>
```

Inheritance diagram for PredictBehavior:



Public Member Functions

- void [predict](#) ()

5.21.1 Detailed Description

class [PredictBehavior](#) -

Definition at line 4 of file PredictBehavior.h.

5.21.2 Member Function Documentation

5.21.2.1 void PredictBehavior::predict ()

Reimplemented in [MLPbehavior](#), and [RBFbehavior](#).

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

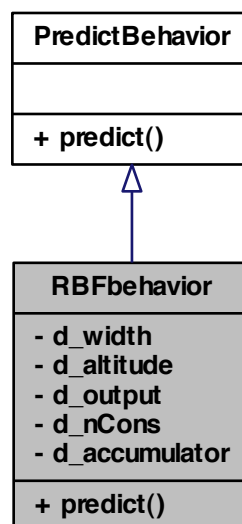
- pkg/AMORE/src/dia/[PredictBehavior.h](#)

5.22 RBFbehavior Class Reference

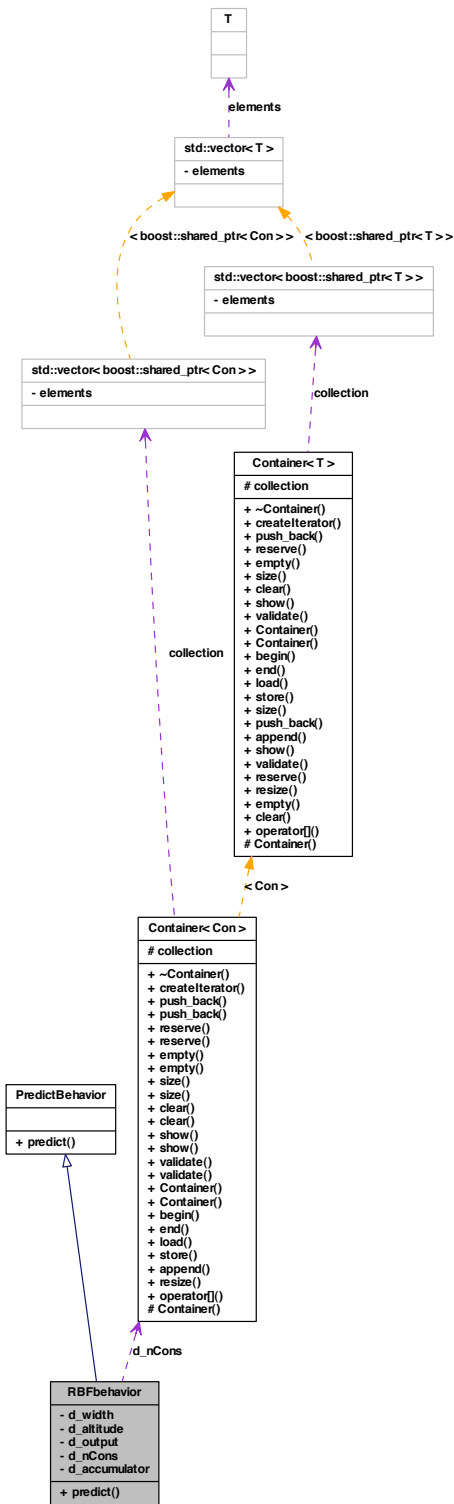
class [RBFbehavior](#) -

```
#include <RBFbehavior.h>
```


Inheritance diagram for RBFbehavior:



Collaboration diagram for RBFbehavior:



Public Member Functions

- void [predict](#) ()

Private Attributes

- double [d_width](#)
- double [d_altitude](#)
- double [d_output](#)
- [Container](#)< [Con](#) > [d_nCons](#)
- double [d_accumulator](#)

5.22.1 Detailed Description

class [RBFbehavior](#) -

Definition at line 5 of file [RBFbehavior.h](#).

5.22.2 Member Function Documentation

5.22.2.1 void [RBFbehavior::predict](#) ()

Reimplemented from [PredictBehavior](#).

5.22.3 Member Data Documentation

5.22.3.1 double [RBFbehavior::d_accumulator](#) [private]

Definition at line 12 of file [RBFbehavior.h](#).

5.22.3.2 double [RBFbehavior::d_altitude](#) [private]

Definition at line 9 of file [RBFbehavior.h](#).

5.22.3.3 [Container](#)<[Con](#)> [RBFbehavior::d_nCons](#) [private]

Definition at line 11 of file [RBFbehavior.h](#).

5.22.3.4 double [RBFbehavior::d_output](#) [private]

Definition at line 10 of file [RBFbehavior.h](#).

5.22.3.5 double RBFbehavior::d_width [private]

Definition at line 8 of file RBFbehavior.h.

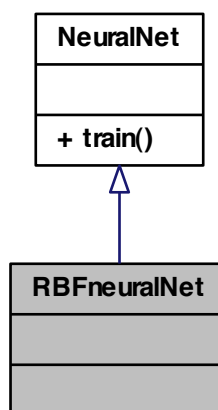
The documentation for this class was generated from the following file:

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

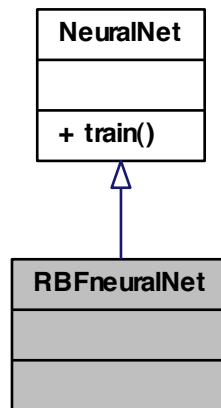
5.23 RBFneuralNet Class Reference

```
#include <RBFneuralNet.h>
```

Inheritance diagram for RBFneuralNet:



Collaboration diagram for RBFneuralNet:



5.23.1 Detailed Description

Definition at line 1 of file `RBFneuralNet.h`.

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

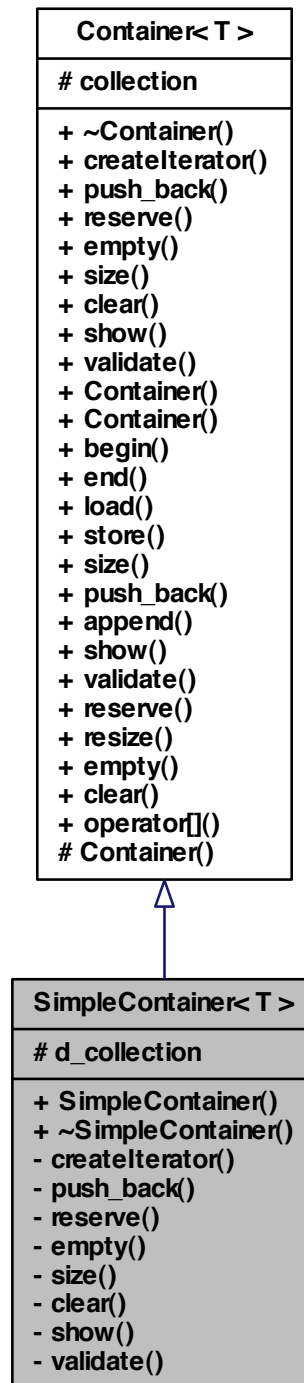
- `pkg/AMORE/src/old/RBFneuralNet.h`

5.24 SimpleContainer< T > Class Template Reference

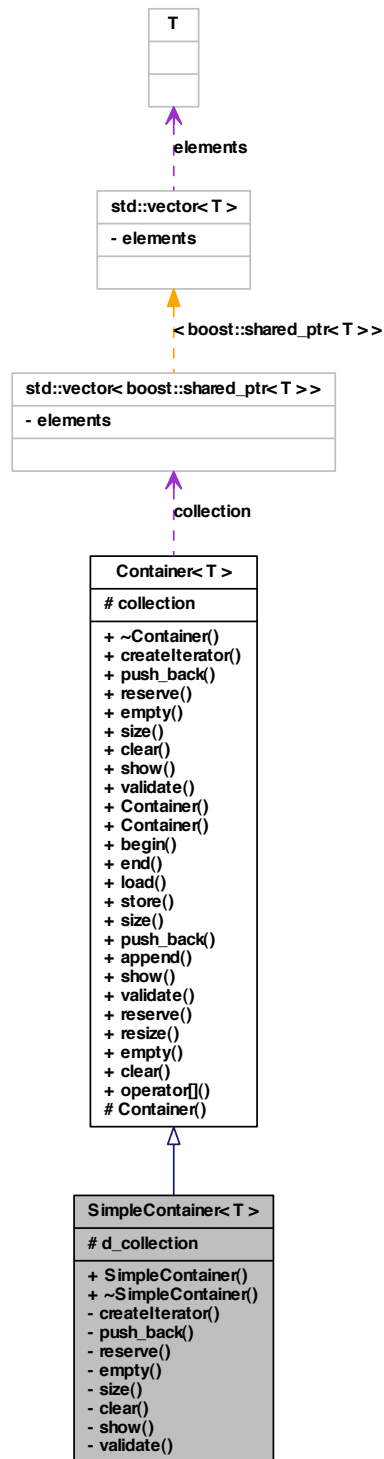
class `SimpleContainer` -

```
#include <SimpleContainer.h>
```

Inheritance diagram for SimpleContainer< T >:



Collaboration diagram for SimpleContainer< T >:



Public Member Functions

- [SimpleContainer](#) ()
- [~SimpleContainer](#) ()

Protected Attributes

- `std::vector< T >` [d_collection](#)

Private Member Functions

- `boost::shared_ptr< Iterator< T > >` [createIterator](#) ()
- `void` [push_back](#) (T const &[const_reference](#))
Append a shared_ptr at the end of collection.
- `void` [reserve](#) (int n)
- `bool` [empty](#) ()
- `size_type` [size](#) ()
Returns the size or length of the vector.
- `void` [clear](#) ()
- `void` [show](#) ()
Pretty print of the Container<T>
- `bool` [validate](#) ()
Object validator.

Friends

- `class` [SimpleContainerIterator](#)< T >

5.24.1 Detailed Description

`template<typename T>class SimpleContainer< T >`

class [SimpleContainer](#) -

Definition at line 6 of file SimpleContainer.h.

5.24.2 Constructor & Destructor Documentation

5.24.2.1 `template<typename T > SimpleContainer< T >::SimpleContainer ()`

5.24.2.2 `template<typename T > SimpleContainer< T >::~~SimpleContainer ()`

5.24.3 Member Function Documentation

5.24.3.1 `template<typename T> void SimpleContainer< T >::clear ()` [private, virtual]

Implements [Container< T >](#).

5.24.3.2 `template<typename T> boost::shared_ptr< Iterator<T> > SimpleContainer< T >::createIterator ()` [private, virtual]

Implements [Container< T >](#).

5.24.3.3 `template<typename T> bool SimpleContainer< T >::empty ()` [private, virtual]

Implements [Container< T >](#).

5.24.3.4 `template<typename T> void SimpleContainer< T >::push_back (T const & reference)` [private, virtual]

Append a `shared_ptr` at the end of collection.

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

Parameters

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

```

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

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

    vc = conContainer.load();

    result.push_back(vc.at(0)->getId());

```

```

        result.push_back(vc.at(1)->getId());
        result.push_back(vc.at(2)->getId());
    // After execution of this code, result contains a numeric vector with va
    lues 10, 20 and 30.

```

See also

C++ documentation for `std::vector::push_back` and the unit test files, e.g., `runit.Cpp.Container.R`, for usage examples.

Implements [Container< T >](#).

5.24.3.5 `template<typename T> void SimpleContainer< T >::reserve (int n)`
`[private, virtual]`

Implements [Container< T >](#).

5.24.3.6 `template<typename T> void SimpleContainer< T >::show ()` `[private, virtual]`

Pretty print of the `Container<T>`

This method outputs in the R terminal the contents of [Container::collection](#).

Returns

true in case everything works without throwing an exception

*

```

//=====
//Usage example:
//=====
// Data set up
ContainerNeuronPtr      neuronContainerPtr( new
Container<Neuron>() );
ContainerConPtr conContainerPtr( new Container<Con>() );
ConPtr ptC;
NeuronPtr ptN;
int ids[] = {10, 20, 30};
double weights[] = {1.13, 2.22, 3.33 };

    for (int i=0; i<=2 ; i++) {                                /
/ Let's create a vector with three neurons
        ptN.reset( new Neuron( ids[i] ) );
        neuronContainerPtr->push_back(ptN);
    }

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

// Test

```

```

conContainerPtr->show() ;

// The output at the R terminal would display:
//
//      # From:  10      Weight=      1.130000
//      # From:  20      Weight=      2.220000
//      # From:  30      Weight=      3.330000
//

```

See also

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

Implements [Container< T >](#).

5.24.3.7 `template<typename T> size_type SimpleContainer< T >::size ()`
`[private, virtual]`

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.

Implements [Container< T >](#).

5.24.3.8 `template<typename T> bool SimpleContainer< T >::validate ()`
`[private, virtual]`

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.

Implements [Container< T >](#).

5.24.4 Friends And Related Function Documentation

5.24.4.1 `template<typename T> friend class SimpleContainerIterator< T >`
`[friend]`

Definition at line 12 of file `SimpleContainer.h`.

5.24.5 Member Data Documentation

5.24.5.1 `template<typename T> std::vector< T> SimpleContainer< T>::d_collection` [protected]

Definition at line 9 of file SimpleContainer.h.

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

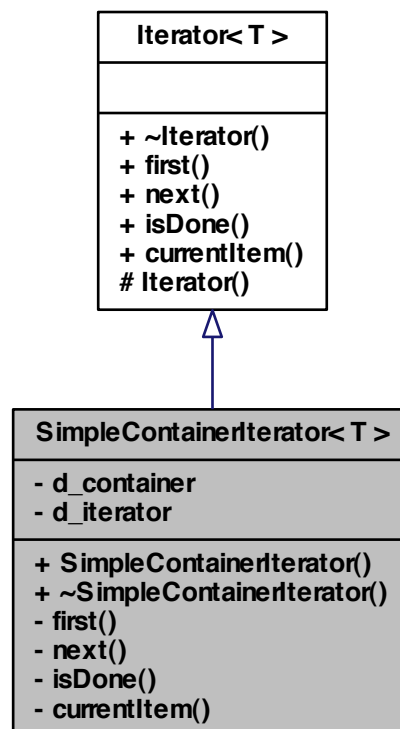
- pkg/AMORE/src/dia/[SimpleContainer.h](#)

5.25 SimpleContainerIterator< T> Class Template Reference

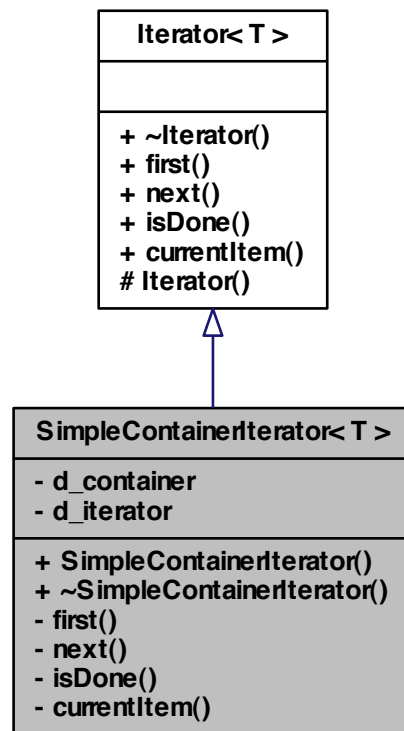
class [SimpleContainerIterator](#) -

```
#include <SimpleContainerIterator.h>
```

Inheritance diagram for SimpleContainerIterator< T>:



Collaboration diagram for SimpleContainerIterator< T >:



Public Member Functions

- [SimpleContainerIterator \(\)](#)
- [~SimpleContainerIterator \(\)](#)

Private Member Functions

- void [first \(\)](#)
- void [next \(\)](#)
- bool [isDone \(\)](#)
- T [currentItem \(\)](#)

Private Attributes

- [Container< T > * d_container](#)
- [std::vector< T >::iterator d_iterator](#)

Friends

- class [SimpleContainer< T >](#)

5.25.1 Detailed Description

`template<typename T>class SimpleContainerIterator< T >`

class [SimpleContainerIterator](#) -

Definition at line 6 of file SimpleContainerIterator.h.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 `template<typename T > SimpleContainerIterator< T >::SimpleContainerIterator ()`

5.25.2.2 `template<typename T > SimpleContainerIterator< T >::~~SimpleContainerIterator ()`

5.25.3 Member Function Documentation

5.25.3.1 `template<typename T > T SimpleContainerIterator< T >::currentItem ()`
[private, virtual]

Implements [Iterator< T >](#).

5.25.3.2 `template<typename T > void SimpleContainerIterator< T >::first ()`
[private, virtual]

Implements [Iterator< T >](#).

5.25.3.3 `template<typename T > bool SimpleContainerIterator< T >::isDone ()`
[private, virtual]

Implements [Iterator< T >](#).

5.25.3.4 `template<typename T> void SimpleContainerIterator< T >::next ()`
`[private, virtual]`

Implements [Iterator< T >](#).

5.25.4 Friends And Related Function Documentation

5.25.4.1 `template<typename T> friend class SimpleContainer< T >` `[friend]`

Definition at line 13 of file SimpleContainerIterator.h.

5.25.5 Member Data Documentation

5.25.5.1 `template<typename T> Container<T>* SimpleContainerIterator< T`
`>::d_container` `[private]`

Definition at line 9 of file SimpleContainerIterator.h.

5.25.5.2 `template<typename T> std::vector<T>::iterator SimpleContainerIterator< T`
`>::d_iterator` `[private]`

Definition at line 10 of file SimpleContainerIterator.h.

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

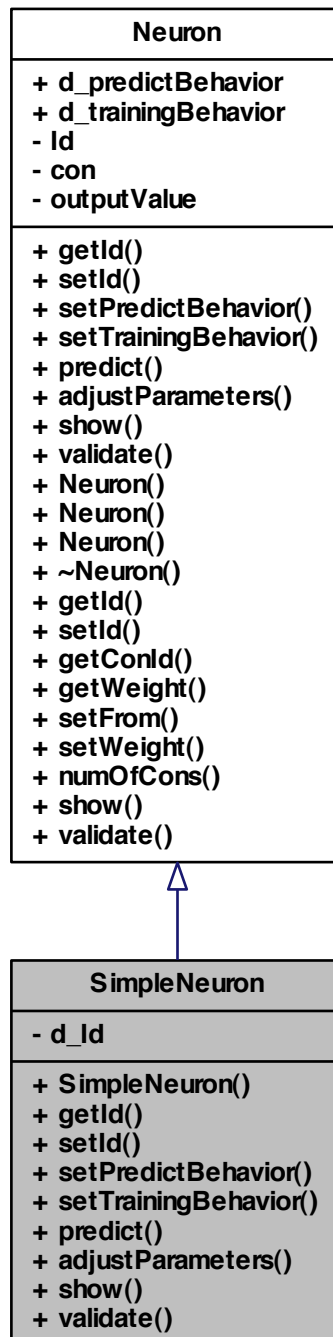
- `pkg/AMORE/src/dia/SimpleContainerIterator.h`

5.26 SimpleNeuron Class Reference

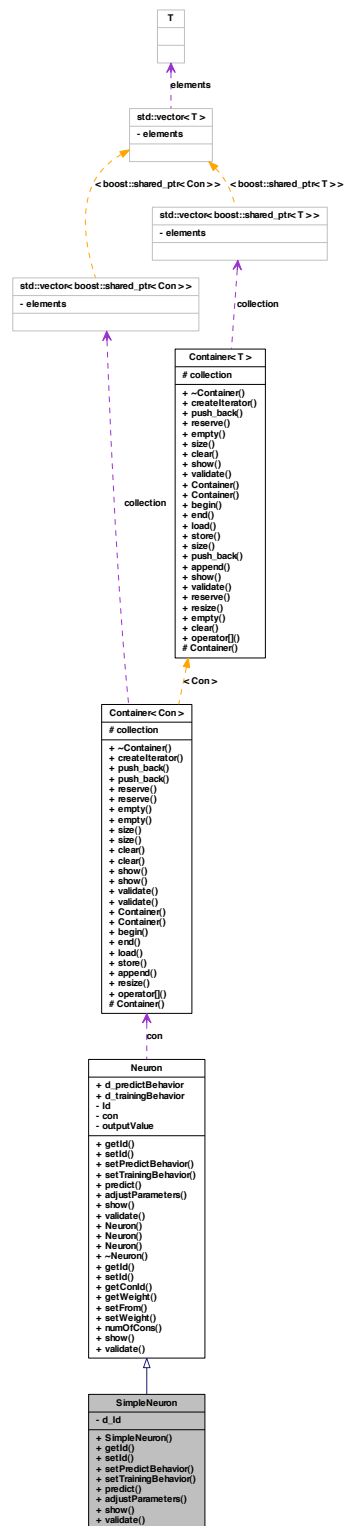
class [SimpleNeuron](#) -

`#include <SimpleNeuron.h>`

Inheritance diagram for SimpleNeuron:



Collaboration diagram for SimpleNeuron:



Public Member Functions

- [SimpleNeuron](#) ()
- [Handler](#) [getId](#) ()
- void [setId](#) ([Handler](#) *Id*)
- void [setPredictBehavior](#) ([PredictBehavior](#) &predictBehavior)
- void [setTrainingBehavior](#) ([TrainingBehavior](#) &trainingBehavior)
- void [predict](#) ()
- void [adjustParameters](#) ()
- void [show](#) ()
- bool [validate](#) ()

Private Attributes

- int [d_Id](#)

5.26.1 Detailed Description

class [SimpleNeuron](#) -

Definition at line 5 of file SimpleNeuron.h.

5.26.2 Constructor & Destructor Documentation

5.26.2.1 [SimpleNeuron::SimpleNeuron](#) ()

5.26.3 Member Function Documentation

5.26.3.1 void [SimpleNeuron::adjustParameters](#) () [virtual]

Implements [Neuron](#).

5.26.3.2 [Handler](#) [SimpleNeuron::getId](#) () [virtual]

Implements [Neuron](#).

5.26.3.3 void [SimpleNeuron::predict](#) () [virtual]

Implements [Neuron](#).

5.26.3.4 void [SimpleNeuron::setId](#) ([Handler](#) *Id*) [virtual]

Implements [Neuron](#).

5.26.3.5 `void SimpleNeuron::setPredictBehavior (PredictBehavior & predictBehavior)`
[virtual]

Implements [Neuron](#).

5.26.3.6 `void SimpleNeuron::setTrainingBehavior (TrainingBehavior & trainingBehavior)`
[virtual]

Implements [Neuron](#).

5.26.3.7 `void SimpleNeuron::show ()` [virtual]

Implements [Neuron](#).

5.26.3.8 `bool SimpleNeuron::validate ()` [virtual]

Implements [Neuron](#).

5.26.4 Member Data Documentation

5.26.4.1 `int SimpleNeuron::d_Id` [private]

Definition at line 8 of file SimpleNeuron.h.

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

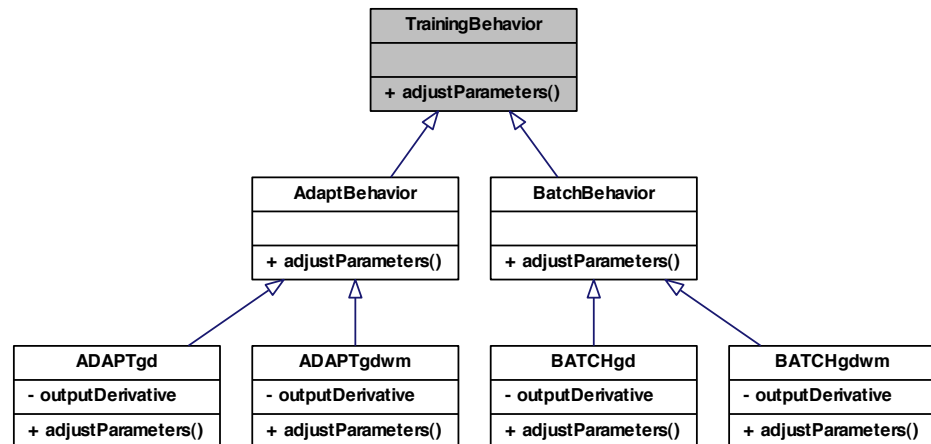
- `pkg/AMORE/src/dia/SimpleNeuron.h`

5.27 TrainingBehavior Class Reference

class [TrainingBehavior](#) -

```
#include <TrainingBehavior.h>
```

Inheritance diagram for TrainingBehavior:



Public Member Functions

- void [adjustParameters](#) ()

5.27.1 Detailed Description

class [TrainingBehavior](#) -

Definition at line 4 of file TrainingBehavior.h.

5.27.2 Member Function Documentation

5.27.2.1 void TrainingBehavior::adjustParameters ()

Reimplemented in [AdaptBehavior](#), [ADAPTgd](#), [ADAPTgdwm](#), [BatchBehavior](#), [BATCHgd](#), and [BATCHgdwm](#).

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

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

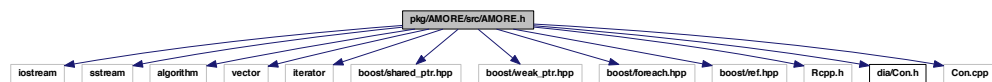
Chapter 6

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 <boost/ref.hpp>
#include <Rcpp.h>
#include "dia/Con.h"
#include "Con.cpp"
```

Include dependency graph for AMORE.h:



Defines

- #define `foreach` BOOST_FOREACH

- #define `size_type` unsigned int

Typedefs

- typedef int `Handler`
- typedef boost::reference_wrapper< `PredictBehavior` > `PredictBehaviorRef`
- typedef boost::reference_wrapper< `TrainingBehavior` > `TrainingBehaviorRef`
- typedef boost::reference_wrapper< `Neuron` > `NeuronRef`
- typedef boost::shared_ptr< `Neuron` > `NeuronPtr`
- typedef boost::shared_ptr< `Con` > `ConPtr`
- typedef boost::shared_ptr< IteratorInterface< `Neuron` > > `NeuronIteratorInterfacePtr`
- typedef boost::shared_ptr< IteratorInterface< `Con` > > `ConIteratorInterfacePtr`
- typedef `Container`< `Con` > `ConContainer`
- typedef `Container`< `Neuron` > `NeuronContainer`

6.1.1 Define Documentation

6.1.1.1 #define foreach BOOST_FOREACH

Definition at line 53 of file AMORE.h.

6.1.1.2 #define size_type unsigned int

Definition at line 56 of file AMORE.h.

6.1.2 Typedef Documentation

6.1.2.1 typedef Container<Con> ConContainer

Definition at line 79 of file AMORE.h.

6.1.2.2 typedef boost::shared_ptr< IteratorInterface<Con> > ConIteratorInterfacePtr

Definition at line 66 of file AMORE.h.

6.1.2.3 typedef boost::shared_ptr<Con> ConPtr

Definition at line 64 of file AMORE.h.

6.1.2.4 typedef int Handler

Definition at line 59 of file AMORE.h.

6.1.2.5 typedef Container<Neuron> NeuronContainer

Definition at line 80 of file AMORE.h.

6.1.2.6 typedef boost::shared_ptr< IteratorInterface<Neuron> > NeuronIteratorInterfacePtr

Definition at line 65 of file AMORE.h.

6.1.2.7 typedef boost::shared_ptr<Neuron> NeuronPtr

Definition at line 63 of file AMORE.h.

6.1.2.8 typedef boost::reference_wrapper<Neuron> NeuronRef

Definition at line 62 of file AMORE.h.

6.1.2.9 typedef boost::reference_wrapper<PredictBehavior> PredictBehaviorRef

Definition at line 60 of file AMORE.h.

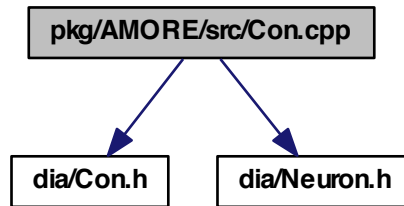
6.1.2.10 typedef boost::reference_wrapper<TrainingBehavior> TrainingBehaviorRef

Definition at line 61 of file AMORE.h.

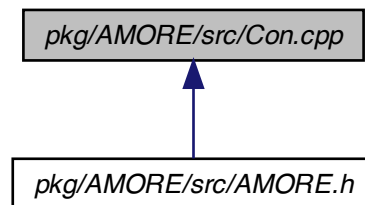
6.2 pkg/AMORE/src/Con.cpp File Reference

```
#include "dia/Con.h"  
#include "dia/Neuron.h"
```

Include dependency graph for Con.cpp:



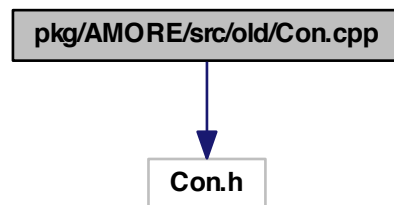
This graph shows which files directly or indirectly include this file:



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

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

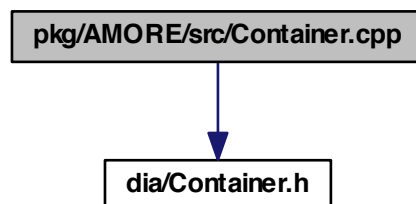

Include dependency graph for Con.cpp:



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

```
#include "dia/Container.h"
```

Include dependency graph for Container.cpp:

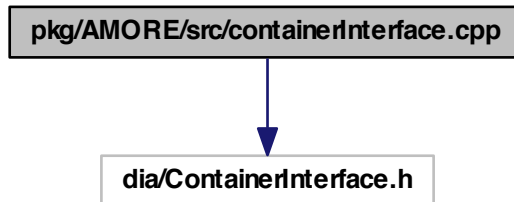


6.5 pkg/AMORE/src/old/Container.cpp File Reference

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

```
#include "dia/ContainerInterface.h"
```

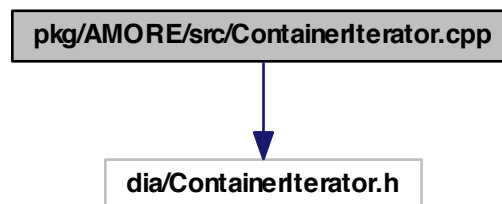
Include dependency graph for containerInterface.cpp:



6.7 pkg/AMORE/src/ContainerIterator.cpp File Reference

```
#include "dia/ContainerIterator.h"
```

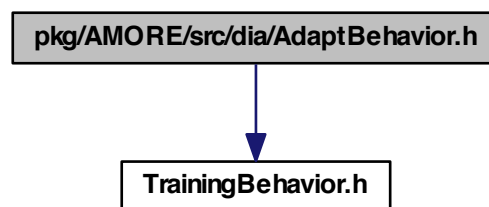
Include dependency graph for ContainerIterator.cpp:



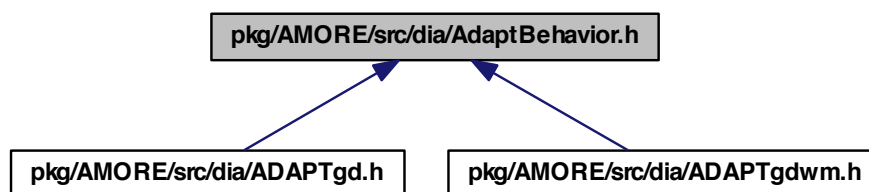
6.8 pkg/AMORE/src/dia/AdaptBehavior.h File Reference

```
#include "TrainingBehavior.h"
```

Include dependency graph for AdaptBehavior.h:



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



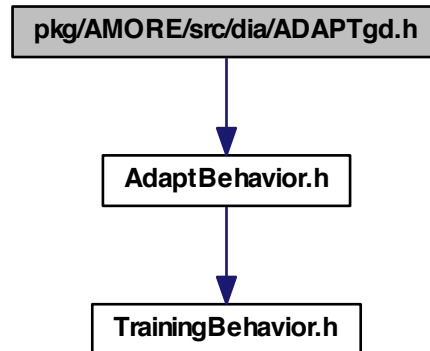
Classes

- class [AdaptBehavior](#)
class [AdaptBehavior](#) -

6.9 pkg/AMORE/src/dia/ADAPTgd.h File Reference

```
#include "AdaptBehavior.h"
```

Include dependency graph for ADAPTgd.h:



Classes

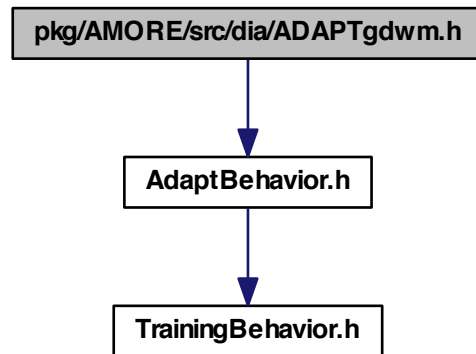
- class [ADAPTgd](#)

class [ADAPTgd](#) -

6.10 pkg/AMORE/src/dia/ADAPTgdwm.h File Reference

```
#include "AdaptBehavior.h"
```

Include dependency graph for ADAPTgdwm.h:



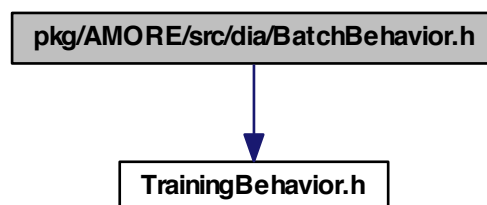
Classes

- class [ADAPTgdwm](#)
class [ADAPTgdwm](#) -

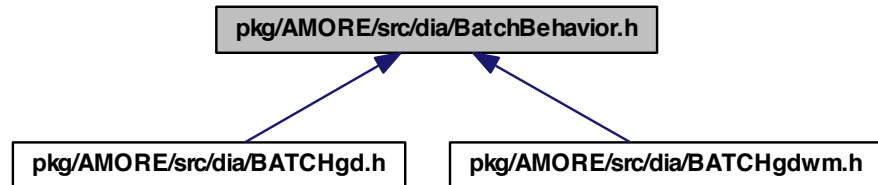
6.11 pkg/AMORE/src/dia/BatchBehavior.h File Reference

```
#include "TrainingBehavior.h"
```

Include dependency graph for BatchBehavior.h:



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



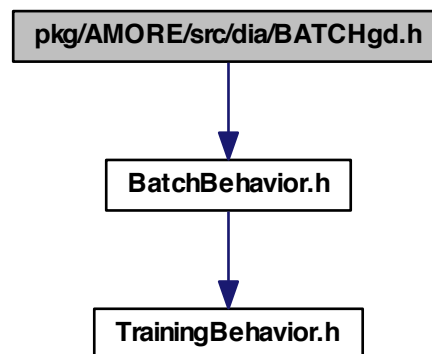
Classes

- class [BatchBehavior](#)
class [BatchBehavior](#) -

6.12 pkg/AMORE/src/dia/BATCHgd.h File Reference

```
#include "BatchBehavior.h"
```

Include dependency graph for BATCHgd.h:



Classes

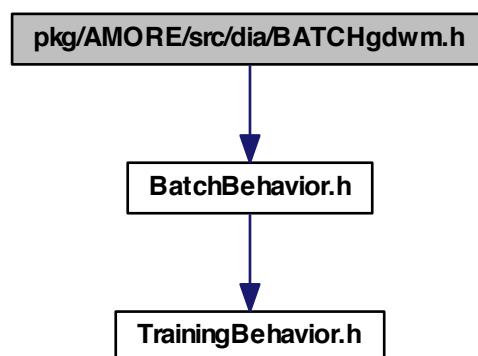
- class [BATCHgd](#)

class [BATCHgd](#) -

6.13 pkg/AMORE/src/dia/BATCHgdwm.h File Reference

```
#include "BatchBehavior.h"
```

Include dependency graph for BATCHgdwm.h:



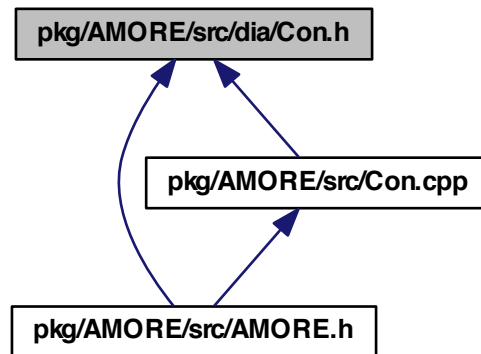
Classes

- class [BATCHgdwm](#)

class [BATCHgdwm](#) -

6.14 pkg/AMORE/src/dia/Con.h File Reference

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

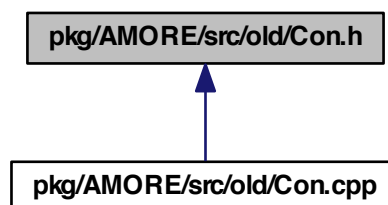


Classes

- class `Con`
class `Con` -

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

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



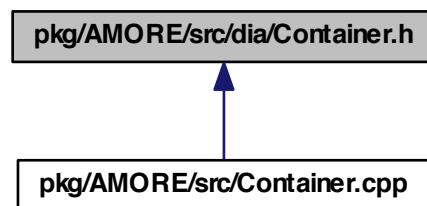
Classes

- class [Con](#)

class [Con](#) -

6.16 pkg/AMORE/src/dia/Container.h File Reference

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



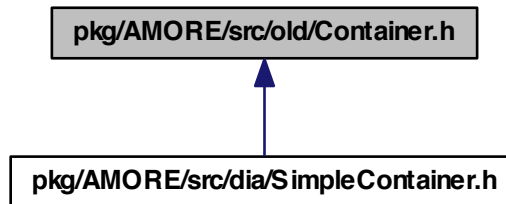
Classes

- class [Container< T >](#)

class [Container](#) -

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

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

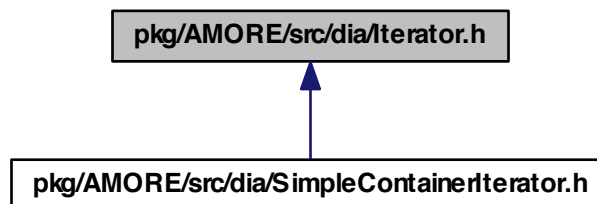


Classes

- class [Container< T >](#)
class [Container](#) -

6.18 pkg/AMORE/src/dia/Iterator.h File Reference

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



Classes

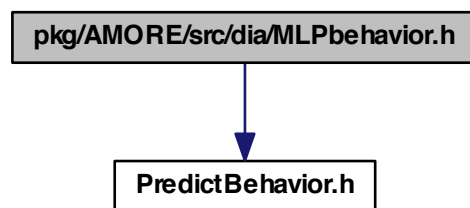
- class [Iterator< T >](#)

class [Iterator](#) -

6.19 pkg/AMORE/src/dia/MLPbehavior.h File Reference

```
#include "PredictBehavior.h"
```

Include dependency graph for MLPbehavior.h:



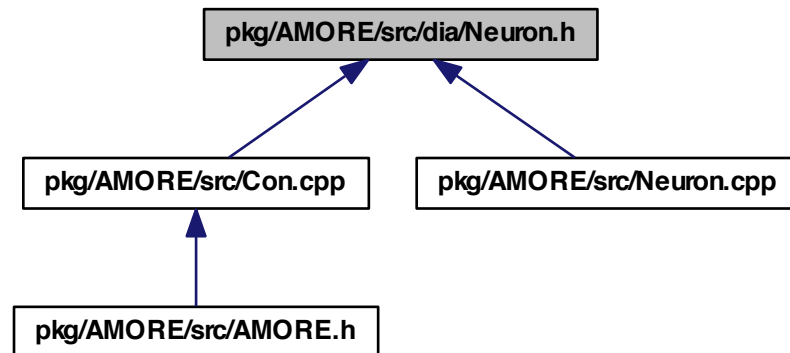
Classes

- class [MLPbehavior](#)

class [MLPbehavior](#) -

6.20 pkg/AMORE/src/dia/Neuron.h File Reference

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

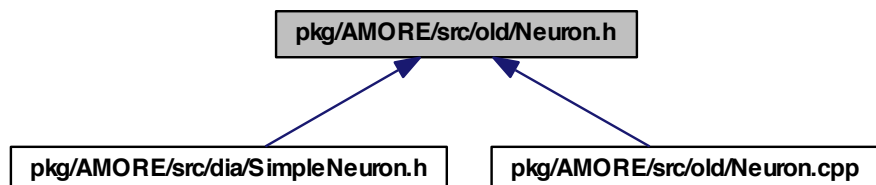


Classes

- class `Neuron`
class `Neuron` -

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

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



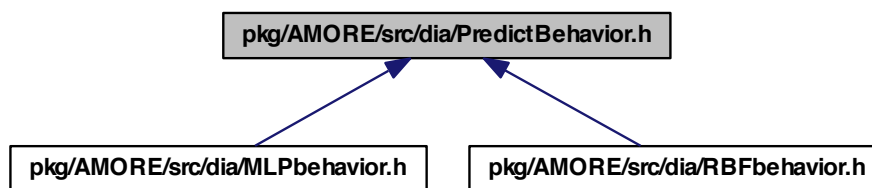
Classes

- class [Neuron](#)

class [Neuron](#) -

6.22 pkg/AMORE/src/dia/PredictBehavior.h File Reference

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



Classes

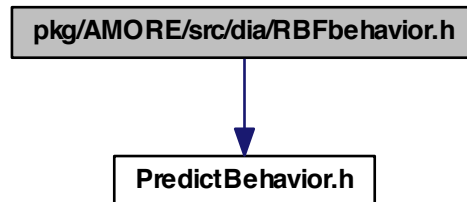
- class [PredictBehavior](#)

class [PredictBehavior](#) -

6.23 pkg/AMORE/src/dia/RBFbehavior.h File Reference

```
#include "PredictBehavior.h"
```

Include dependency graph for RBFbehavior.h:



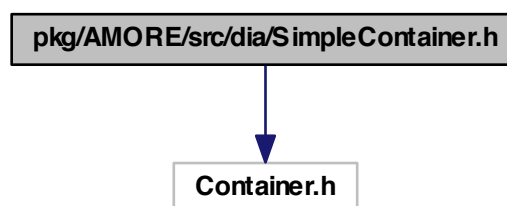
Classes

- class [RBFbehavior](#)
class [RBFbehavior](#) -

6.24 pkg/AMORE/src/dia/SimpleContainer.h File Reference

```
#include "Container.h"
```

Include dependency graph for SimpleContainer.h:



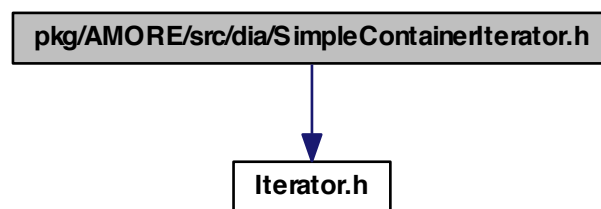
Classes

- class [SimpleContainer< T >](#)
class [SimpleContainer](#) -

6.25 pkg/AMORE/src/dia/SimpleContainerIterator.h File Reference

```
#include "Iterator.h"
```

Include dependency graph for SimpleContainerIterator.h:



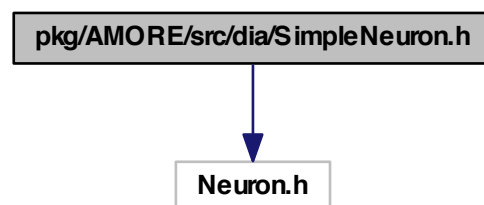
Classes

- class `SimpleContainerIterator< T >`
class `SimpleContainerIterator` -

6.26 pkg/AMORE/src/dia/SimpleNeuron.h File Reference

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

Include dependency graph for SimpleNeuron.h:

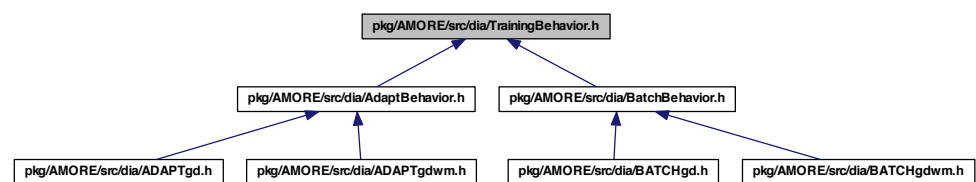


Classes

- class [SimpleNeuron](#)
class [SimpleNeuron](#) -

6.27 pkg/AMORE/src/dia/TrainingBehavior.h File Reference

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



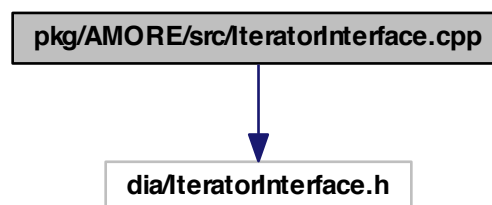
Classes

- class [TrainingBehavior](#)
class [TrainingBehavior](#) -

6.28 pkg/AMORE/src/IteratorInterface.cpp File Reference

```
#include "dia/IteratorInterface.h"
```

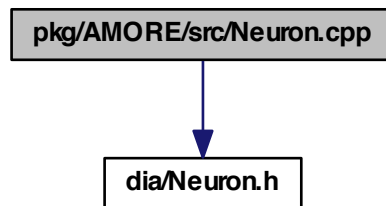
Include dependency graph for `IteratorInterface.cpp`:



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

```
#include "dia/Neuron.h"
```

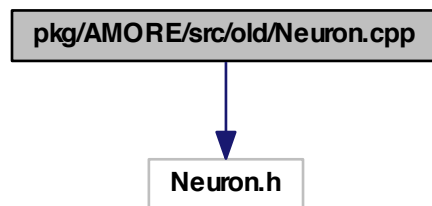
Include dependency graph for Neuron.cpp:



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

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

Include dependency graph for Neuron.cpp:



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

Classes

- struct [CompareId](#)

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

Classes

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

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

Classes

- class [MLPlayer](#)

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

Classes

- class [MLPlayerContainer](#)

6.35 pkg/AMORE/src/old/MLPneuralNet.h File Reference

Classes

- class [MLPneuralNet](#)

6.36 pkg/AMORE/src/old/MLPneuralNetFactory.cpp File Reference

Functions

- [MLPneuralNet CreateMLPneuralNet](#) (std::vector< int > numberOfNeuronsPerLayer)

6.36.1 Function Documentation

6.36.1.1 MLPneuralNet CreateMLPneuralNet (std::vector< int > *numberOfNeuronsPerLayer*)

Definition at line 2 of file MLPneuralNetFactory.cpp.

```
{  
  
    net = new MLPNeuralNet();  
  
    MLPPlayerPtr mlpLayerPtr;  
  
    std::vector<int> idx;  
  
    foreach (int n, numberOfNeuronsPerLayer)  
    {  
        for (int i=1; i<=n; ++i)  
        {  
            idx.push_back(i);  
        }  
        mlpLayerPtr.reset(new MLPPlayer( idx ) );  
        net.nLayers.push_back(mlpLayerPtr);  
    }  
  
  
    for (int i=1; i<=; ++i)  
    {  
  
        mlpPtr->buildAndAppend();  
  
    }  
}
```

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

Classes

- class [MLPneuron](#)

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

Classes

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

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

Classes

- class [NeuralNet](#)

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

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

Classes

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

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

Classes

- class [RBFneuralNet](#)

Index

- ~Con
 - Con, [29](#)
- ~Container
 - Container, [59](#)
- ~Iterator
 - Iterator, [72](#)
- ~Neuron
 - Neuron, [95](#)
- ~NeuronContainer
 - NeuronContainer, [102](#)
- ~SimpleContainer
 - SimpleContainer, [114](#)
- ~SimpleContainerIterator
 - SimpleContainerIterator, [120](#)
- AdaptBehavior, [9](#)
 - adjustParameters, [11](#)
- ADAPTgd, [12](#)
 - adjustParameters, [13](#)
 - outputDerivative, [14](#)
- ADAPTgdwm, [14](#)
 - adjustParameters, [16](#)
 - outputDerivative, [17](#)
- adjustParameters
 - AdaptBehavior, [11](#)
 - ADAPTgd, [13](#)
 - ADAPTgdwm, [16](#)
 - BatchBehavior, [19](#)
 - BATCHgd, [21](#)
 - BATCHgdwm, [24](#)
 - Neuron, [95](#)
 - SimpleNeuron, [124](#)
 - TrainingBehavior, [126](#)
- AMORE.h
 - ConContainer, [128](#)
 - ConIteratorInterfacePtr, [128](#)
 - ConPtr, [128](#)
 - foreach, [128](#)
 - Handler, [128](#)
 - NeuronContainer, [128](#)
 - NeuronIteratorInterfacePtr, [129](#)
 - NeuronPtr, [129](#)
 - NeuronRef, [129](#)
 - PredictBehaviorRef, [129](#)
 - size_type, [128](#)
 - TrainingBehaviorRef, [129](#)
- append
 - Container, [59](#)
- BatchBehavior, [17](#)
 - adjustParameters, [19](#)
- BATCHgd, [20](#)
 - adjustParameters, [21](#)
 - outputDerivative, [22](#)
- BATCHgdwm, [22](#)
 - adjustParameters, [24](#)
 - outputDerivative, [25](#)
- begin
 - Container, [61](#)
- bias
 - MLPneuron, [87](#)
- buildAndAppend
 - MLPneuronContainer, [90](#)
- clear
 - Container, [61](#), [62](#)
 - SimpleContainer, [114](#)
- collection
 - Container, [70](#)
- CompareId, [25](#)
 - operator(), [25](#), [26](#)
- Con, [26](#)
 - ~Con, [29](#)
 - Con, [28](#)
 - d_neuron, [36](#)
 - d_weight, [36](#)
 - from, [36](#)
 - getFrom, [29](#)
 - getId, [30](#)
 - getNeuron, [30](#)
 - getWeight, [31](#), [32](#)
 - Id, [32](#)

- setFrom, 33
 - setNeuron, 33
 - setWeight, 34
 - show, 34, 35
 - validate, 35, 36
 - weight, 37
- con
 - Neuron, 97
- ConContainer, 37
 - AMORE.h, 128
 - ConContainer, 41
 - const_iterator, 40
 - const_reference, 40
 - erase, 41
 - getId, 43
 - iterator, 41
 - numOfCons, 45
 - select, 46
 - setFrom, 48
 - setWeight, 50, 51
 - validate, 53
 - value_type, 41
- ConIteratorInterfacePtr
 - AMORE.h, 128
- ConPtr
 - AMORE.h, 128
- const_iterator
 - ConContainer, 40
 - Container, 58
 - NeuronContainer, 101
- const_reference
 - ConContainer, 40
 - Container, 58
 - NeuronContainer, 101
- Container, 54
 - ~Container, 59
 - append, 59
 - begin, 61
 - clear, 61, 62
 - collection, 70
 - const_iterator, 58
 - const_reference, 58
 - Container, 59
 - createIterator, 62
 - empty, 62
 - end, 63
 - iterator, 58
 - load, 63
 - push_back, 64, 65
 - reserve, 66
 - resize, 67
 - show, 67
 - size, 68
 - store, 69
 - validate, 69
 - value_type, 58
- createIterator
 - Container, 62
 - SimpleContainer, 115
- CreateMLPNeuralNet
 - MLPNeuralNetFactory.cpp, 148
- currentItem
 - Iterator, 72
 - SimpleContainerIterator, 120
- d_accumulator
 - MLPbehavior, 75
 - RBFbehavior, 109
- d_altitude
 - RBFbehavior, 109
- d_bias
 - MLPbehavior, 75
- d_collection
 - SimpleContainer, 117
- d_container
 - SimpleContainerIterator, 121
- d_id
 - SimpleNeuron, 125
- d_iterator
 - SimpleContainerIterator, 121
- d_nCons
 - MLPbehavior, 75
 - RBFbehavior, 109
- d_neuron
 - Con, 36
- d_output
 - MLPbehavior, 75
 - RBFbehavior, 109
- d_predictBehavior
 - Neuron, 98
- d_trainingBehavior
 - Neuron, 98
- d_weight
 - Con, 36
- d_width
 - RBFbehavior, 109
- empty
 - Container, 62
 - SimpleContainer, 115

- end
 - Container, [63](#)
- erase
 - ConContainer, [41](#)
- first
 - Iterator, [72](#)
 - SimpleContainerIterator, [120](#)
- foreach
 - AMORE.h, [128](#)
- from
 - Con, [36](#)
- getConId
 - Neuron, [95](#)
 - NeuronContainer, [102](#)
- getFrom
 - Con, [29](#)
 - NeuronContainer, [103](#)
- getId
 - Con, [30](#)
 - ConContainer, [43](#)
 - MLPNeuronContainer, [90](#)
 - Neuron, [95](#)
 - NeuronContainer, [103](#)
 - SimpleNeuron, [124](#)
- getNeuron
 - Con, [30](#)
- getWeight
 - Con, [31](#), [32](#)
 - Neuron, [95](#)
 - NeuronContainer, [103](#)
- Handler
 - AMORE.h, [128](#)
- Id
 - Con, [32](#)
 - Neuron, [98](#)
- isDone
 - Iterator, [72](#)
 - SimpleContainerIterator, [120](#)
- Iterator, [70](#)
 - ~Iterator, [72](#)
 - currentItem, [72](#)
 - first, [72](#)
 - isDone, [72](#)
 - Iterator, [72](#)
 - next, [72](#)
- iterator
 - ConContainer, [41](#)
 - Container, [58](#)
 - NeuronContainer, [101](#)
- load
 - Container, [63](#)
- MLPbehavior, [72](#)
 - d_accumulator, [75](#)
 - d_bias, [75](#)
 - d_nCons, [75](#)
 - d_output, [75](#)
 - predict, [75](#)
- MLPlayer, [76](#)
- MLPlayerContainer, [79](#)
- MLPneuralNet, [82](#)
 - nLayers, [84](#)
- MLPneuralNetFactory.cpp
 - CreateMLPneuralNet, [148](#)
- MLPneuron, [84](#)
 - bias, [87](#)
- MLPneuronContainer, [87](#)
 - buildAndAppend, [90](#)
 - getId, [90](#)
- NeuralNet, [90](#)
 - train, [91](#)
- Neuron, [91](#)
 - ~Neuron, [95](#)
 - adjustParameters, [95](#)
 - con, [97](#)
 - d_predictBehavior, [98](#)
 - d_trainingBehavior, [98](#)
 - getConId, [95](#)
 - getId, [95](#)
 - getWeight, [95](#)
 - Id, [98](#)
 - Neuron, [95](#)
 - numOfCons, [96](#)
 - outputValue, [98](#)
 - predict, [96](#)
 - setFrom, [96](#)
 - setId, [96](#)
 - setPredictBehavior, [96](#)
 - setTrainingBehavior, [96](#)
 - setWeight, [96](#)
 - show, [96](#), [97](#)
 - validate, [97](#)
- NeuronContainer, [98](#)
 - ~NeuronContainer, [102](#)

- AMORE.h, 128
- const_iterator, 101
- const_reference, 101
- getConId, 102
- getFrom, 103
- getId, 103
- getWeight, 103
- iterator, 101
- NeuronContainer, 102
- numOfCons, 103
- numOfNeurons, 103
- setFrom, 104
- setId, 104
- setWeight, 104
- value_type, 102
- NeuronIteratorInterfacePtr
 - AMORE.h, 129
- NeuronPtr
 - AMORE.h, 129
- NeuronRef
 - AMORE.h, 129
- next
 - Iterator, 72
 - SimpleContainerIterator, 120
- nLayers
 - MLPNeuralNet, 84
- numOfCons
 - ConContainer, 45
 - Neuron, 96
 - NeuronContainer, 103
- numOfNeurons
 - NeuronContainer, 103
- operator()
 - CompareId, 25, 26
- outputDerivative
 - ADAPTgd, 14
 - ADAPTgdwm, 17
 - BATCHgd, 22
 - BATCHgdwm, 25
- outputValue
 - Neuron, 98
- pkg/AMORE/src/AMORE.h, 127
- pkg/AMORE/src/Con.cpp, 129
- pkg/AMORE/src/Container.cpp, 131
- pkg/AMORE/src/containerInterface.cpp, 131
- pkg/AMORE/src/ContainerIterator.cpp, 132
- pkg/AMORE/src/dia/AdaptBehavior.h, 132
- pkg/AMORE/src/dia/ADAPTgd.h, 133
- pkg/AMORE/src/dia/ADAPTgdwm.h, 134
- pkg/AMORE/src/dia/BatchBehavior.h, 135
- pkg/AMORE/src/dia/BATCHgd.h, 136
- pkg/AMORE/src/dia/BATCHgdwm.h, 137
- pkg/AMORE/src/dia/Con.h, 138
- pkg/AMORE/src/dia/Container.h, 139
- pkg/AMORE/src/dia/Iterator.h, 140
- pkg/AMORE/src/dia/MLPbehavior.h, 141
- pkg/AMORE/src/dia/Neuron.h, 142
- pkg/AMORE/src/dia/PredictBehavior.h, 143
- pkg/AMORE/src/dia/RBFbehavior.h, 143
- pkg/AMORE/src/dia/SimpleContainer.h, 144
- pkg/AMORE/src/dia/SimpleContainerIterator.h, 145
- pkg/AMORE/src/dia/SimpleNeuron.h, 145
- pkg/AMORE/src/dia/TrainingBehavior.h, 146
- pkg/AMORE/src/IteratorInterface.cpp, 146
- pkg/AMORE/src/Neuron.cpp, 147
- pkg/AMORE/src/old/Con.cpp, 130
- pkg/AMORE/src/old/Con.h, 138
- pkg/AMORE/src/old/ConContainer.cpp, 147
- pkg/AMORE/src/old/ConContainer.h, 148
- pkg/AMORE/src/old/Container.cpp, 131
- pkg/AMORE/src/old/Container.h, 140
- pkg/AMORE/src/old/MLPlayer.h, 148
- pkg/AMORE/src/old/MLPlayerContainer.h, 148
- pkg/AMORE/src/old/MLPNeuralNet.h, 148
- pkg/AMORE/src/old/MLPNeuralNetFactory.cpp, 148
- pkg/AMORE/src/old/MLPNeuron.h, 149
- pkg/AMORE/src/old/MLPNeuronContainer.h, 149
- pkg/AMORE/src/old/NeuralNet.h, 149
- pkg/AMORE/src/old/Neuron.cpp, 147
- pkg/AMORE/src/old/Neuron.h, 142
- pkg/AMORE/src/old/NeuronContainer.cpp, 150
- pkg/AMORE/src/old/NeuronContainer.h, 150
- pkg/AMORE/src/old/RBFneuralNet.h, 150
- predict
 - MLPbehavior, 75
 - Neuron, 96
 - PredictBehavior, 106
 - RBFbehavior, 109
 - SimpleNeuron, 124
- PredictBehavior, 105
- predict, 106
- PredictBehaviorRef
 - AMORE.h, 129

- push_back
 - Container, [64, 65](#)
 - SimpleContainer, [115](#)
- RBFbehavior, [106](#)
 - d_accumulator, [109](#)
 - d_altitude, [109](#)
 - d_nCons, [109](#)
 - d_output, [109](#)
 - d_width, [109](#)
 - predict, [109](#)
- RBFneuralNet, [110](#)
- reserve
 - Container, [66](#)
 - SimpleContainer, [116](#)
- resize
 - Container, [67](#)
- select
 - ConContainer, [46](#)
- setFrom
 - Con, [33](#)
 - ConContainer, [48](#)
 - Neuron, [96](#)
 - NeuronContainer, [104](#)
- setId
 - Neuron, [96](#)
 - NeuronContainer, [104](#)
 - SimpleNeuron, [124](#)
- setNeuron
 - Con, [33](#)
- setPredictBehavior
 - Neuron, [96](#)
 - SimpleNeuron, [124](#)
- setTrainingBehavior
 - Neuron, [96](#)
 - SimpleNeuron, [125](#)
- setWeight
 - Con, [34](#)
 - ConContainer, [50, 51](#)
 - Neuron, [96](#)
 - NeuronContainer, [104](#)
- show
 - Con, [34, 35](#)
 - Container, [67](#)
 - Neuron, [96, 97](#)
 - SimpleContainer, [116](#)
 - SimpleNeuron, [125](#)
- SimpleContainer, [111](#)
 - ~SimpleContainer, [114](#)
- clear, [114](#)
- createIterator, [115](#)
- d_collection, [117](#)
- empty, [115](#)
- push_back, [115](#)
- reserve, [116](#)
- show, [116](#)
- SimpleContainer, [114](#)
- SimpleContainerIterator< T >, [117](#)
- size, [117](#)
- validate, [117](#)
- SimpleContainer< T >
 - SimpleContainerIterator, [121](#)
- SimpleContainerIterator, [118](#)
 - ~SimpleContainerIterator, [120](#)
- currentItem, [120](#)
- d_container, [121](#)
- d_iterator, [121](#)
- first, [120](#)
- isDone, [120](#)
- next, [120](#)
- SimpleContainer< T >, [121](#)
- SimpleContainerIterator, [120](#)
- SimpleContainerIterator< T >
 - SimpleContainer, [117](#)
- SimpleNeuron, [121](#)
 - adjustParameters, [124](#)
 - d_Id, [125](#)
 - getId, [124](#)
 - predict, [124](#)
 - setId, [124](#)
 - setPredictBehavior, [124](#)
 - setTrainingBehavior, [125](#)
 - show, [125](#)
 - SimpleNeuron, [124](#)
 - validate, [125](#)
- size
 - Container, [68](#)
 - SimpleContainer, [117](#)
- size_type
 - AMORE.h, [128](#)
- store
 - Container, [69](#)
- train
 - NeuralNet, [91](#)
- TrainingBehavior, [125](#)
 - adjustParameters, [126](#)
- TrainingBehaviorRef
 - AMORE.h, [129](#)

validate

Con, [35](#), [36](#)ConContainer, [53](#)Container, [69](#)Neuron, [97](#)SimpleContainer, [117](#)SimpleNeuron, [125](#)

value_type

ConContainer, [41](#)Container, [58](#)NeuronContainer, [102](#)

weight

Con, [37](#)