

## AMORE++

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

Generated by Doxygen 1.7.4

Thu Jul 14 2011 19:12:28



# Contents

<b>1</b>	<b>The AMORE++ package</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Motivation . . . . .	1
1.3	Road Map . . . . .	1
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class Hierarchy . . . . .	3
<b>3</b>	<b>Class Index</b>	<b>5</b>
3.1	Class List . . . . .	5
<b>4</b>	<b>File Index</b>	<b>7</b>
4.1	File List . . . . .	7
<b>5</b>	<b>Class Documentation</b>	<b>9</b>
5.1	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	29
5.8.2.6	~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	35
5.8.3.11	show	35
5.8.3.12	show	36
5.8.3.13	validate	36
5.8.3.14	validate	36
5.8.4	Member Data Documentation	36
5.8.4.1	d_neuron	37
5.8.4.2	d_weight	37
5.8.4.3	from	37
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	<a href="#">getId</a>	43
5.9.4.3	<a href="#">numOfCons</a>	45
5.9.4.4	<a href="#">select</a>	46
5.9.4.5	<a href="#">setFrom</a>	48
5.9.4.6	<a href="#">setWeight</a>	50
5.9.4.7	<a href="#">setWeight</a>	52
5.9.4.8	<a href="#">validate</a>	53
5.10	<a href="#">Container&lt; T &gt; Class Template Reference</a>	54
5.10.1	<a href="#">Detailed Description</a>	58
5.10.2	<a href="#">Member Typedef Documentation</a>	58
5.10.2.1	<a href="#">const_iterator</a>	58
5.10.2.2	<a href="#">const_reference</a>	58
5.10.2.3	<a href="#">iterator</a>	58
5.10.2.4	<a href="#">value_type</a>	58
5.10.3	<a href="#">Constructor &amp; Destructor Documentation</a>	59
5.10.3.1	<a href="#">~Container</a>	59
5.10.3.2	<a href="#">Container</a>	59
5.10.3.3	<a href="#">Container</a>	59
5.10.3.4	<a href="#">Container</a>	59
5.10.4	<a href="#">Member Function Documentation</a>	59
5.10.4.1	<a href="#">append</a>	59
5.10.4.2	<a href="#">begin</a>	61
5.10.4.3	<a href="#">clear</a>	61
5.10.4.4	<a href="#">clear</a>	62
5.10.4.5	<a href="#">createIterator</a>	62
5.10.4.6	<a href="#">empty</a>	62
5.10.4.7	<a href="#">empty</a>	63
5.10.4.8	<a href="#">end</a>	63
5.10.4.9	<a href="#">load</a>	63
5.10.4.10	<a href="#">operator[]</a>	64
5.10.4.11	<a href="#">push_back</a>	64
5.10.4.12	<a href="#">push_back</a>	65
5.10.4.13	<a href="#">reserve</a>	66
5.10.4.14	<a href="#">reserve</a>	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	MLPfactory Class Reference	76
5.13.1	Detailed Description	77
5.13.2	Constructor & Destructor Documentation	77
5.13.2.1	MLPfactory	77
5.13.3	Member Function Documentation	78

---

5.13.3.1	makeCon	78
5.13.3.2	makeNeuron	78
5.14	MLPlayer Class Reference	78
5.14.1	Detailed Description	81
5.15	MLPlayerContainer Class Reference	81
5.15.1	Detailed Description	84
5.16	MLPneuralNet Class Reference	84
5.16.1	Detailed Description	86
5.16.2	Member Data Documentation	86
5.16.2.1	nLayers	86
5.17	MLPneuron Class Reference	86
5.17.1	Detailed Description	89
5.17.2	Member Data Documentation	89
5.17.2.1	bias	89
5.18	MLPneuronContainer Class Reference	89
5.18.1	Detailed Description	92
5.18.2	Member Function Documentation	92
5.18.2.1	buildAndAppend	92
5.18.2.2	getId	92
5.19	NeuralCreator Class Reference	92
5.19.1	Detailed Description	93
5.19.2	Member Function Documentation	93
5.19.2.1	createCon	93
5.19.2.2	createNeuron	94
5.20	NeuralFactory Class Reference	94
5.20.1	Detailed Description	95
5.20.2	Member Function Documentation	95
5.20.2.1	makeCon	95
5.20.2.2	makeNeuron	95
5.21	NeuralNet Class Reference	96
5.21.1	Detailed Description	96
5.21.2	Member Function Documentation	96
5.21.2.1	train	96
5.22	Neuron Class Reference	97



5.22.1	Detailed Description	99
5.22.2	Constructor & Destructor Documentation	99
5.22.2.1	Neuron	99
5.22.2.2	Neuron	100
5.22.2.3	Neuron	100
5.22.2.4	~Neuron	100
5.22.3	Member Function Documentation	100
5.22.3.1	getConId	100
5.22.3.2	getId	100
5.22.3.3	getId	100
5.22.3.4	getWeight	101
5.22.3.5	numOfCons	101
5.22.3.6	setFrom	101
5.22.3.7	setId	101
5.22.3.8	setId	101
5.22.3.9	setWeight	101
5.22.3.10	show	101
5.22.3.11	show	102
5.22.3.12	validate	102
5.22.3.13	validate	103
5.22.4	Member Data Documentation	103
5.22.4.1	con	103
5.22.4.2	id	103
5.22.4.3	outputValue	103
5.23	NeuronContainer Class Reference	103
5.23.1	Detailed Description	106
5.23.2	Member Typedef Documentation	106
5.23.2.1	const_iterator	106
5.23.2.2	const_reference	106
5.23.2.3	iterator	107
5.23.2.4	value_type	107
5.23.3	Constructor & Destructor Documentation	107
5.23.3.1	NeuronContainer	107
5.23.3.2	NeuronContainer	107

5.23.3.3	<a href="#">~NeuronContainer</a>	107
5.23.4	<a href="#">Member Function Documentation</a>	107
5.23.4.1	<a href="#">getConId</a>	107
5.23.4.2	<a href="#">getFrom</a>	108
5.23.4.3	<a href="#">getId</a>	108
5.23.4.4	<a href="#">getWeight</a>	108
5.23.4.5	<a href="#">numOfCons</a>	108
5.23.4.6	<a href="#">numOfNeurons</a>	109
5.23.4.7	<a href="#">setFrom</a>	109
5.23.4.8	<a href="#">setId</a>	109
5.23.4.9	<a href="#">setWeight</a>	110
5.24	<a href="#">PredictBehavior Class Reference</a>	110
5.24.1	<a href="#">Detailed Description</a>	111
5.24.2	<a href="#">Member Function Documentation</a>	111
5.24.2.1	<a href="#">predict</a>	111
5.25	<a href="#">RBFbehavior Class Reference</a>	111
5.25.1	<a href="#">Detailed Description</a>	114
5.25.2	<a href="#">Member Function Documentation</a>	114
5.25.2.1	<a href="#">predict</a>	114
5.25.3	<a href="#">Member Data Documentation</a>	114
5.25.3.1	<a href="#">d_accumulator</a>	114
5.25.3.2	<a href="#">d_altitude</a>	114
5.25.3.3	<a href="#">d_nCons</a>	114
5.25.3.4	<a href="#">d_output</a>	114
5.25.3.5	<a href="#">d_width</a>	115
5.26	<a href="#">RBFfactory Class Reference</a>	115
5.26.1	<a href="#">Detailed Description</a>	116
5.26.2	<a href="#">Constructor &amp; Destructor Documentation</a>	116
5.26.2.1	<a href="#">RBFfactory</a>	116
5.26.3	<a href="#">Member Function Documentation</a>	116
5.26.3.1	<a href="#">makeCon</a>	116
5.26.3.2	<a href="#">makeNeuron</a>	117
5.27	<a href="#">RBFneuralNet Class Reference</a>	117
5.27.1	<a href="#">Detailed Description</a>	118

5.28 SimpleContainer< T > Class Template Reference . . . . .	118
5.28.1 Detailed Description . . . . .	121
5.28.2 Constructor & Destructor Documentation . . . . .	121
5.28.2.1 SimpleContainer . . . . .	121
5.28.2.2 ~SimpleContainer . . . . .	121
5.28.3 Member Function Documentation . . . . .	121
5.28.3.1 clear . . . . .	122
5.28.3.2 createIterator . . . . .	122
5.28.3.3 empty . . . . .	122
5.28.3.4 push_back . . . . .	122
5.28.3.5 reserve . . . . .	123
5.28.3.6 show . . . . .	123
5.28.3.7 size . . . . .	124
5.28.3.8 validate . . . . .	124
5.28.4 Friends And Related Function Documentation . . . . .	124
5.28.4.1 SimpleContainerIterator< T > . . . . .	124
5.28.5 Member Data Documentation . . . . .	124
5.28.5.1 d_collection . . . . .	125
5.29 SimpleContainerIterator< T > Class Template Reference . . . . .	125
5.29.1 Detailed Description . . . . .	127
5.29.2 Constructor & Destructor Documentation . . . . .	127
5.29.2.1 SimpleContainerIterator . . . . .	127
5.29.2.2 ~SimpleContainerIterator . . . . .	127
5.29.3 Member Function Documentation . . . . .	127
5.29.3.1 currentItem . . . . .	127
5.29.3.2 first . . . . .	127
5.29.3.3 isDone . . . . .	127
5.29.3.4 next . . . . .	128
5.29.4 Friends And Related Function Documentation . . . . .	128
5.29.4.1 SimpleContainer< T > . . . . .	128
5.29.5 Member Data Documentation . . . . .	128
5.29.5.1 d_container . . . . .	128
5.29.5.2 d_iterator . . . . .	128
5.30 SimpleNeuralCreator Class Reference . . . . .	128

5.30.1	Detailed Description	130
5.30.2	Constructor & Destructor Documentation	130
5.30.2.1	SimpleNeuralCreator	130
5.30.3	Member Function Documentation	131
5.30.3.1	createCon	131
5.30.3.2	createNeuron	131
5.31	SimpleNeuron Class Reference	132
5.31.1	Detailed Description	135
5.31.2	Constructor & Destructor Documentation	135
5.31.2.1	SimpleNeuron	135
5.31.3	Member Function Documentation	135
5.31.3.1	getId	135
5.31.3.2	setId	136
5.31.3.3	show	136
5.31.3.4	validate	137
5.31.4	Member Data Documentation	138
5.31.4.1	d_Id	138
5.32	TrainingBehavior Class Reference	138
5.32.1	Detailed Description	138
5.32.2	Member Function Documentation	139
5.32.2.1	adjustParameters	139
<b>6</b>	<b>File Documentation</b>	<b>141</b>
6.1	pkg/AMORE/src/AMORE.h File Reference	141
6.1.1	Define Documentation	142
6.1.1.1	foreach	142
6.1.1.2	size_type	142
6.1.2	Typedef Documentation	142
6.1.2.1	ConContainer	142
6.1.2.2	ConIteratorPtr	143
6.1.2.3	ConPtr	143
6.1.2.4	Handler	143
6.1.2.5	NeuralCreatorPtr	143
6.1.2.6	NeuralFactoryPtr	143

6.1.2.7	NeuronContainer . . . . .	143
6.1.2.8	NeuronIteratorPtr . . . . .	143
6.1.2.9	NeuronPtr . . . . .	143
6.1.2.10	NeuronRef . . . . .	143
6.1.2.11	PredictBehaviorRef . . . . .	143
6.1.2.12	TrainingBehaviorRef . . . . .	144
6.2	pkg/AMORE/src/Con.cpp File Reference . . . . .	144
6.3	pkg/AMORE/src/old/Con.cpp File Reference . . . . .	144
6.4	pkg/AMORE/src/Container.cpp File Reference . . . . .	145
6.5	pkg/AMORE/src/old/Container.cpp File Reference . . . . .	145
6.6	pkg/AMORE/src/containerInterface.cpp File Reference . . . . .	145
6.7	pkg/AMORE/src/ContainerIterator.cpp File Reference . . . . .	146
6.8	pkg/AMORE/src/dia/AdaptBehavior.h File Reference . . . . .	146
6.9	pkg/AMORE/src/dia/ADAPTgd.h File Reference . . . . .	147
6.10	pkg/AMORE/src/dia/ADAPTgdwm.h File Reference . . . . .	148
6.11	pkg/AMORE/src/dia/BatchBehavior.h File Reference . . . . .	149
6.12	pkg/AMORE/src/dia/BATCHgd.h File Reference . . . . .	150
6.13	pkg/AMORE/src/dia/BATCHgdwm.h File Reference . . . . .	151
6.14	pkg/AMORE/src/dia/Con.h File Reference . . . . .	152
6.15	pkg/AMORE/src/old/Con.h File Reference . . . . .	152
6.16	pkg/AMORE/src/dia/Container.h File Reference . . . . .	153
6.17	pkg/AMORE/src/old/Container.h File Reference . . . . .	154
6.18	pkg/AMORE/src/dia/Iterator.h File Reference . . . . .	154
6.19	pkg/AMORE/src/dia/MLPbehavior.h File Reference . . . . .	155
6.20	pkg/AMORE/src/dia/MLPfactory.h File Reference . . . . .	155
6.21	pkg/AMORE/src/dia/NeuralCreator.h File Reference . . . . .	157
6.22	pkg/AMORE/src/dia/NeuralFactory.h File Reference . . . . .	158
6.23	pkg/AMORE/src/dia/Neuron.h File Reference . . . . .	159
6.24	pkg/AMORE/src/old/Neuron.h File Reference . . . . .	160
6.25	pkg/AMORE/src/dia/PredictBehavior.h File Reference . . . . .	161
6.26	pkg/AMORE/src/dia/RBFbehavior.h File Reference . . . . .	161
6.27	pkg/AMORE/src/dia/RBFfactory.h File Reference . . . . .	162
6.28	pkg/AMORE/src/dia/SimpleContainer.h File Reference . . . . .	162
6.29	pkg/AMORE/src/dia/SimpleContainerIterator.h File Reference . . . . .	163

6.30	<a href="#">pkg/AMORE/src/dia/SimpleNeuralCreator.h File Reference</a>	164
6.31	<a href="#">pkg/AMORE/src/dia/SimpleNeuron.h File Reference</a>	165
6.32	<a href="#">pkg/AMORE/src/dia/TrainingBehavior.h File Reference</a>	166
6.33	<a href="#">pkg/AMORE/src/IteratorInterface.cpp File Reference</a>	166
6.34	<a href="#">pkg/AMORE/src/MLPfactory.cpp File Reference</a>	166
6.35	<a href="#">pkg/AMORE/src/old/ConContainer.cpp File Reference</a>	167
6.36	<a href="#">pkg/AMORE/src/old/ConContainer.h File Reference</a>	167
6.37	<a href="#">pkg/AMORE/src/old/MLPlayer.h File Reference</a>	168
6.38	<a href="#">pkg/AMORE/src/old/MLPlayerContainer.h File Reference</a>	168
6.39	<a href="#">pkg/AMORE/src/old/MLPneuralNet.h File Reference</a>	168
6.40	<a href="#">pkg/AMORE/src/old/MLPneuralNetFactory.cpp File Reference</a>	168
6.40.1	<a href="#">Function Documentation</a>	168
6.40.1.1	<a href="#">CreateMLPneuralNet</a>	168
6.41	<a href="#">pkg/AMORE/src/old/MLPneuron.h File Reference</a>	169
6.42	<a href="#">pkg/AMORE/src/old/MLPneuronContainer.h File Reference</a>	169
6.43	<a href="#">pkg/AMORE/src/old/NeuralNet.h File Reference</a>	169
6.44	<a href="#">pkg/AMORE/src/old/Neuron.cpp File Reference</a>	169
6.45	<a href="#">pkg/AMORE/src/old/NeuronContainer.cpp File Reference</a>	170
6.46	<a href="#">pkg/AMORE/src/old/NeuronContainer.h File Reference</a>	170
6.47	<a href="#">pkg/AMORE/src/old/RBFneuralNet.h File Reference</a>	170
6.48	<a href="#">pkg/AMORE/src/SimpleNeuralCreator.cpp File Reference</a>	170
6.49	<a href="#">pkg/AMORE/src/SimpleNeuron.cpp File Reference</a>	171

# 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 > . . . . .	118
Container< Con > . . . . .	54
ConContainer . . . . .	37
Container< MLPlayer > . . . . .	54
MLPlayerContainer . . . . .	81
Container< Neuron > . . . . .	54
NeuronContainer . . . . .	103
Iterator< T > . . . . .	70
SimpleContainerIterator< T > . . . . .	125
NeuralCreator . . . . .	92
SimpleNeuralCreator . . . . .	128
NeuralFactory . . . . .	94
MLPfactory . . . . .	76
RBFfactory . . . . .	115
NeuralNet . . . . .	96
MLPneuralNet . . . . .	84
RBFneuralNet . . . . .	117
Neuron . . . . .	97
MLPneuron . . . . .	86
SimpleNeuron . . . . .	132
NeuronContainer< MLP > . . . . .	103
MLPneuronContainer . . . . .	89
MLPlayer . . . . .	78

PredictBehavior . . . . .	110
MLPbehavior . . . . .	72
RBFbehavior . . . . .	111
TrainingBehavior . . . . .	138
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:

<a href="#">AdaptBehavior</a> (Class <a href="#">AdaptBehavior</a> - ) . . . . .	9
<a href="#">ADAPTgd</a> (Class <a href="#">ADAPTgd</a> - ) . . . . .	12
<a href="#">ADAPTgdwm</a> (Class <a href="#">ADAPTgdwm</a> - ) . . . . .	14
<a href="#">BatchBehavior</a> (Class <a href="#">BatchBehavior</a> - ) . . . . .	17
<a href="#">BATCHgd</a> (Class <a href="#">BATCHgd</a> - ) . . . . .	20
<a href="#">BATCHgdwm</a> (Class <a href="#">BATCHgdwm</a> - ) . . . . .	22
<a href="#">CompareId</a> . . . . .	25
<a href="#">Con</a> (Class <a href="#">Con</a> - ) . . . . .	26
<a href="#">ConContainer</a> (A vector of connections ) . . . . .	37
<a href="#">Container&lt; T &gt;</a> (Class <a href="#">Container</a> - ) . . . . .	54
<a href="#">Iterator&lt; T &gt;</a> (Class <a href="#">Iterator</a> - ) . . . . .	70
<a href="#">MLPbehavior</a> (Class <a href="#">MLPbehavior</a> - ) . . . . .	72
<a href="#">MLPfactory</a> (Class <a href="#">MLPfactory</a> - ) . . . . .	76
<a href="#">MLPlayer</a> . . . . .	78
<a href="#">MLPlayerContainer</a> . . . . .	81
<a href="#">MLPneuralNet</a> . . . . .	84
<a href="#">MLPneuron</a> . . . . .	86
<a href="#">MLPneuronContainer</a> (A vector of connections ) . . . . .	89
<a href="#">NeuralCreator</a> (Class <a href="#">NeuralCreator</a> - ) . . . . .	92
<a href="#">NeuralFactory</a> (Class <a href="#">NeuralFactory</a> - ) . . . . .	94
<a href="#">NeuralNet</a> . . . . .	96
<a href="#">Neuron</a> (Class <a href="#">Neuron</a> - ) . . . . .	97
<a href="#">NeuronContainer</a> (A vector of neurons ) . . . . .	103
<a href="#">PredictBehavior</a> (Class <a href="#">PredictBehavior</a> - ) . . . . .	110
<a href="#">RBFbehavior</a> (Class <a href="#">RBFbehavior</a> - ) . . . . .	111
<a href="#">RBFfactory</a> (Class <a href="#">RBFfactory</a> - ) . . . . .	115
<a href="#">RBFneuralNet</a> . . . . .	117
<a href="#">SimpleContainer&lt; T &gt;</a> (Class <a href="#">SimpleContainer</a> - ) . . . . .	118
<a href="#">SimpleContainerIterator&lt; T &gt;</a> (Class <a href="#">SimpleContainerIterator</a> - ) . . . . .	125

<a href="#">SimpleNeuralCreator</a> (Class <a href="#">SimpleNeuralCreator</a> - ) . . . . .	128
<a href="#">SimpleNeuron</a> (Class <a href="#">SimpleNeuron</a> - ) . . . . .	132
<a href="#">TrainingBehavior</a> (Class <a href="#">TrainingBehavior</a> - ) . . . . .	138

## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

pkg/AMORE/src/AMORE.h	141
pkg/AMORE/src/Con.cpp	144
pkg/AMORE/src/Container.cpp	145
pkg/AMORE/src/containerInterface.cpp	145
pkg/AMORE/src/ContainerIterator.cpp	146
pkg/AMORE/src/IteratorInterface.cpp	166
pkg/AMORE/src/MLPfactory.cpp	166
pkg/AMORE/src/SimpleNeuralCreator.cpp	170
pkg/AMORE/src/SimpleNeuron.cpp	171
pkg/AMORE/src/dia/AdaptBehavior.h	146
pkg/AMORE/src/dia/ADAPTgd.h	147
pkg/AMORE/src/dia/ADAPTgdwm.h	148
pkg/AMORE/src/dia/BatchBehavior.h	149
pkg/AMORE/src/dia/BATCHgd.h	150
pkg/AMORE/src/dia/BATCHgdwm.h	151
pkg/AMORE/src/dia/Con.h	152
pkg/AMORE/src/dia/Container.h	153
pkg/AMORE/src/dia/Iterator.h	154
pkg/AMORE/src/dia/MLPbehavior.h	155
pkg/AMORE/src/dia/MLPfactory.h	155
pkg/AMORE/src/dia/NeuralCreator.h	157
pkg/AMORE/src/dia/NeuralFactory.h	158
pkg/AMORE/src/dia/Neuron.h	159
pkg/AMORE/src/dia/PredictBehavior.h	161
pkg/AMORE/src/dia/RBFbehavior.h	161
pkg/AMORE/src/dia/RBFfactory.h	162
pkg/AMORE/src/dia/SimpleContainer.h	162
pkg/AMORE/src/dia/SimpleContainerIterator.h	163
pkg/AMORE/src/dia/SimpleNeuralCreator.h	164

pkg/AMORE/src/dia/ <a href="#">SimpleNeuron.h</a> . . . . .	165
pkg/AMORE/src/dia/ <a href="#">TrainingBehavior.h</a> . . . . .	166
pkg/AMORE/src/old/ <a href="#">Con.cpp</a> . . . . .	144
pkg/AMORE/src/old/ <a href="#">Con.h</a> . . . . .	152
pkg/AMORE/src/old/ <a href="#">ConContainer.cpp</a> . . . . .	167
pkg/AMORE/src/old/ <a href="#">ConContainer.h</a> . . . . .	167
pkg/AMORE/src/old/ <a href="#">Container.cpp</a> . . . . .	145
pkg/AMORE/src/old/ <a href="#">Container.h</a> . . . . .	154
pkg/AMORE/src/old/ <a href="#">MLPlayer.h</a> . . . . .	168
pkg/AMORE/src/old/ <a href="#">MLPlayerContainer.h</a> . . . . .	168
pkg/AMORE/src/old/ <a href="#">MLPneuralNet.h</a> . . . . .	168
pkg/AMORE/src/old/ <a href="#">MLPneuralNetFactory.cpp</a> . . . . .	168
pkg/AMORE/src/old/ <a href="#">MLPneuron.h</a> . . . . .	169
pkg/AMORE/src/old/ <a href="#">MLPneuronContainer.h</a> . . . . .	169
pkg/AMORE/src/old/ <a href="#">NeuralNet.h</a> . . . . .	169
pkg/AMORE/src/old/ <a href="#">Neuron.cpp</a> . . . . .	169
pkg/AMORE/src/old/ <a href="#">Neuron.h</a> . . . . .	160
pkg/AMORE/src/old/ <a href="#">NeuronContainer.cpp</a> . . . . .	170
pkg/AMORE/src/old/ <a href="#">NeuronContainer.h</a> . . . . .	170
pkg/AMORE/src/old/ <a href="#">RBFneuralNet.h</a> . . . . .	170

## Chapter 5

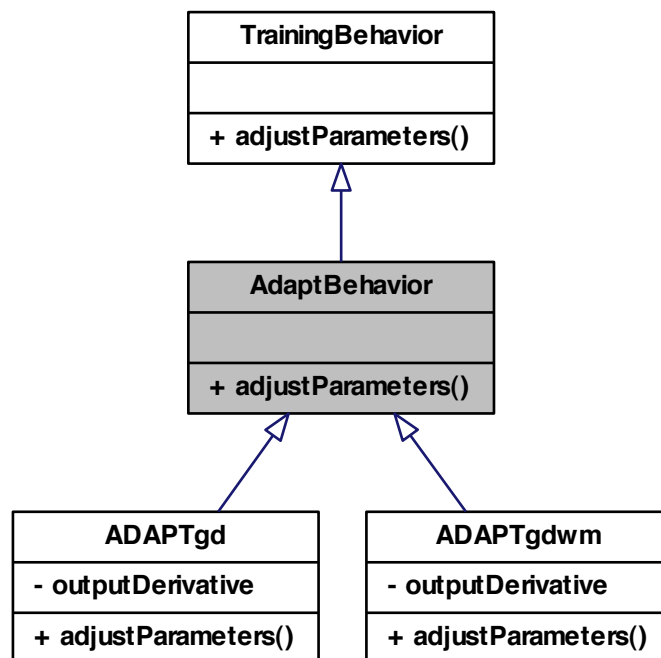
# Class Documentation

### 5.1 `AdaptBehavior` Class Reference

class `AdaptBehavior` -

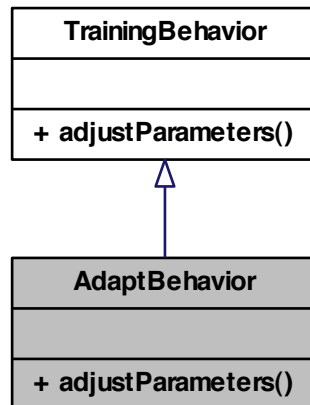
```
#include <AdaptBehavior.h>
```

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:

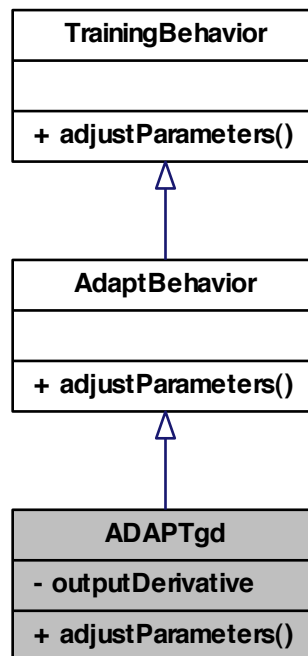
- [pkg/AMORE/src/dia/AdaptBehavior.h](#)

## 5.2 ADAPTgd Class Reference

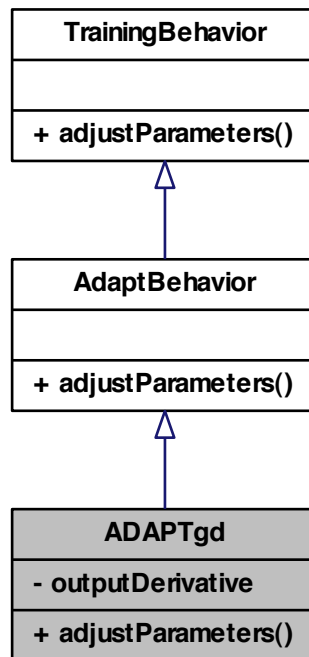
class [ADAPTgd](#) -

```
#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:

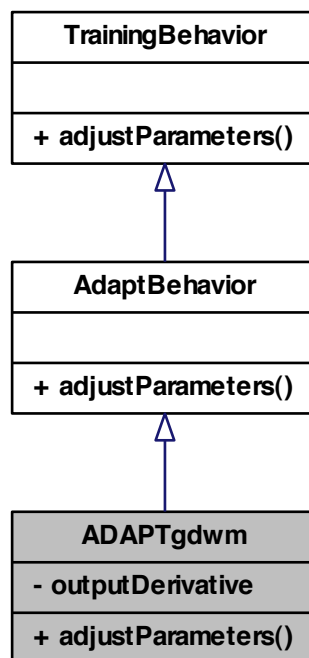
- pkg/AMORE/src/dia/[ADAPTgd.h](#)

## 5.3 ADAPTgdwm Class Reference

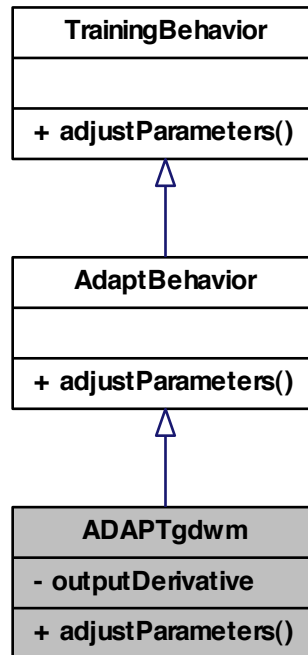
class [ADAPTgdwm](#) -

#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 ADAPTgdmw::adjustParameters ( ) [virtual]

Implements [AdaptBehavior](#).

### 5.3.3 Member Data Documentation

5.3.3.1 double ADAPTgdmw::outputDerivative [private]

Definition at line 8 of file ADAPTgdmw.h.

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

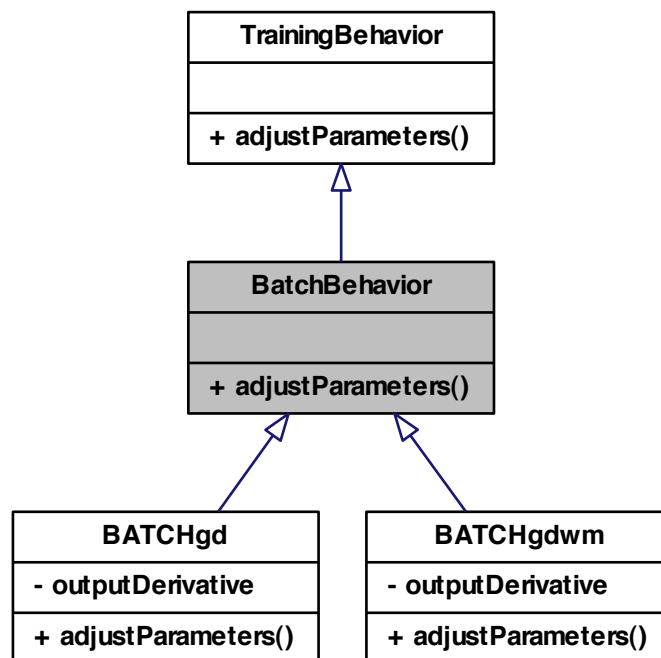
- pkg/AMORE/src/dia/[ADAPTgdmw.h](#)

## 5.4 BatchBehavior Class Reference

class [BatchBehavior](#) -

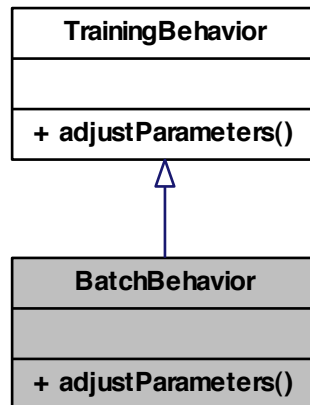
```
#include <BatchBehavior.h>
```

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:

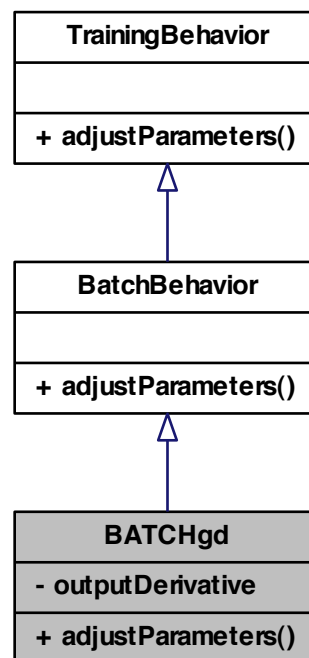
- [pkg/AMORE/src/dia/BatchBehavior.h](#)

## 5.5 BATCHgd Class Reference

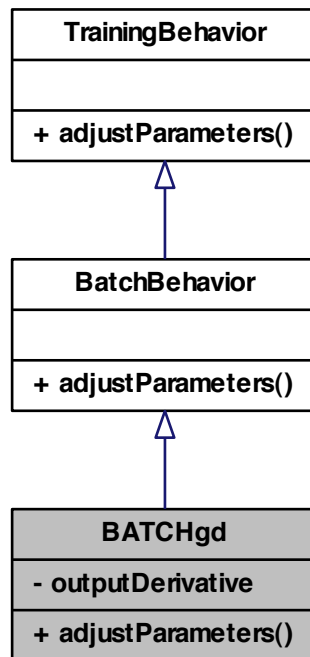
class [BATCHgd](#) -

```
#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:

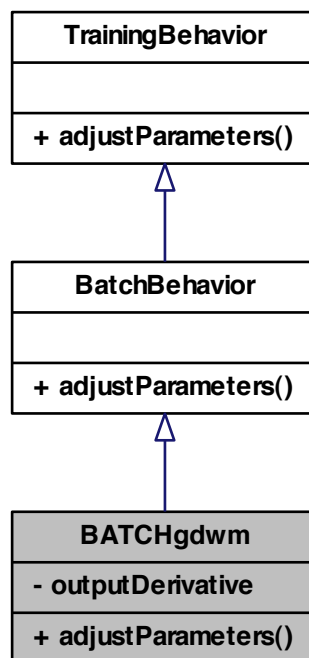
- pkg/AMORE/src/dia/[BATCHgd.h](#)

## 5.6 BATCHgdwm Class Reference

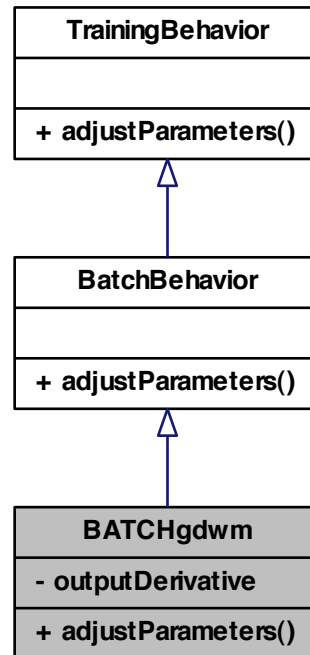
class [BATCHgdwm](#) -

#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](#) ([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](#) ()

*Default Constructor.*

- [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 ( Neuron & *neuron* )

Constructor.

Definition at line 19 of file Con.cpp.

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

### 5.8.2.2 Con::Con ( Neuron & *neuron*, double *weight* )

Constructor.

Definition at line 30 of file Con.cpp.

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

### 5.8.2.3 Con::Con ( )

Default Constructor.

Definition at line 17 of file Con.cpp.

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

### 5.8.2.4 Con::Con ( NeuronPtr *neuronPtr* )

Constructor.

Definition at line 40 of file Con.cpp.

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

### 5.8.2.5 Con::Con ( NeuronPtr neuronPtr, double value )

Constructor.

Definition at line 29 of file Con.cpp.

```

        from(neuronPtr), weight(value)
    {
    }

```

### 5.8.2.6 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 56 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.

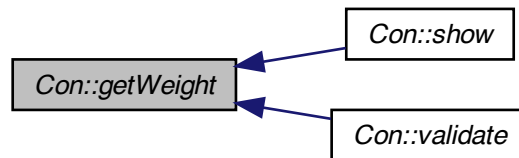
Definition at line 116 of file `Con.cpp`.

References `d_weight`.

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

```
{
    return d_weight;
}
```

Here is the caller graph for this function:

**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.

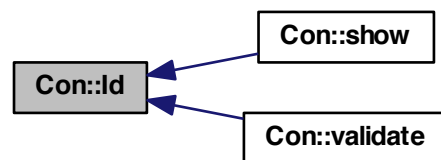
Definition at line 88 of file `Con.cpp`.

References `d_neuron`.

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

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

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

```

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

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

// Now, result is equal to 1

```

**See also**

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

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

References from.

```
{
    from = neuronPtr;
}
```

**5.8.3.8 void Con::setNeuron ( Neuron & neuron )**

Definition at line 63 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**

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

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

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

**See also**

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



Definition at line 123 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

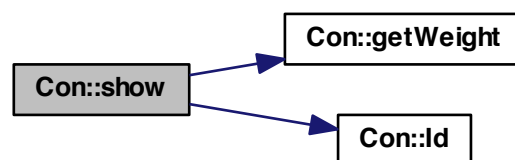
[setWeight](#) and the unit test files, e.g., `runit.Cpp.Con.R`, for usage examples.

Definition at line 135 of file Con.cpp.

References `getWeight()`, 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 , getWeight() );  
    }  
}
```

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

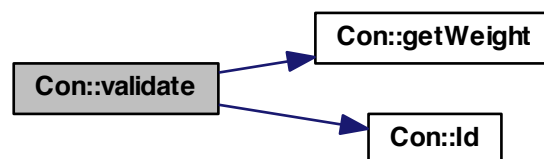
<i>An</i> std::range error if weight or from are not finite.
--

Definition at line 155 of file Con.cpp.

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

```
{  
    BEGIN_RCPP  
    if (! R_FINITE(getWeight()) ) 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()`, and `setWeight()`.

#### 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()`.

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

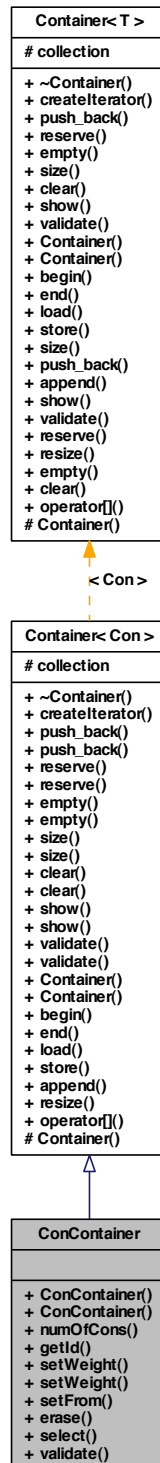
- `pkg/AMORE/src/dia/Con.h`
- `pkg/AMORE/src/old/Con.h`
- `pkg/AMORE/src/Con.cpp`
- `pkg/AMORE/src/old/Con.cpp`

## 5.9 ConContainer Class Reference

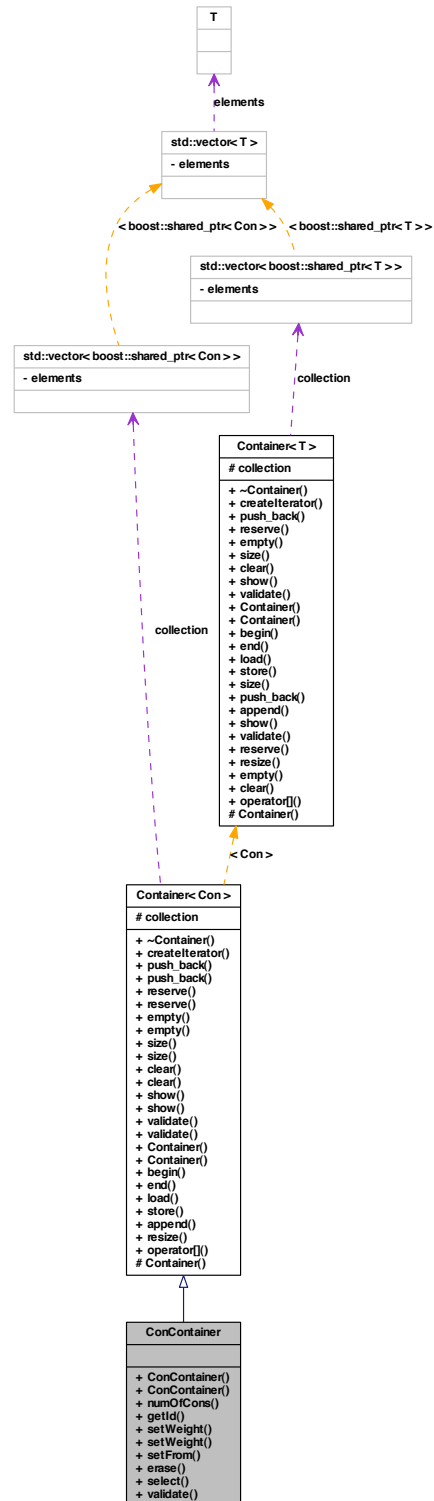
A vector of connections.

```
#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&amp; ConContainer::const\_reference

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

Definition at line 27 of file ConContainer.h.

## 5.9.2.3 typedef std::vector&lt;boost::shared\_ptr&lt;Con&gt; &gt;::iterator ConContainer::iterator

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

Definition at line 21 of file ConContainer.h.

## 5.9.2.4 typedef boost::shared\_ptr&lt;Con&gt; ConContainer::value\_type

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

Definition at line 25 of file ConContainer.h.

## 5.9.3 Constructor &amp; Destructor Documentation

## 5.9.3.1 ConContainer::ConContainer ( )

Definition at line 8 of file ConContainer.cpp.

```
{
}
```

## 5.9.3.2 ConContainer::ConContainer ( std::vector&lt; ConPtr &gt; 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&lt; int &gt; 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`.

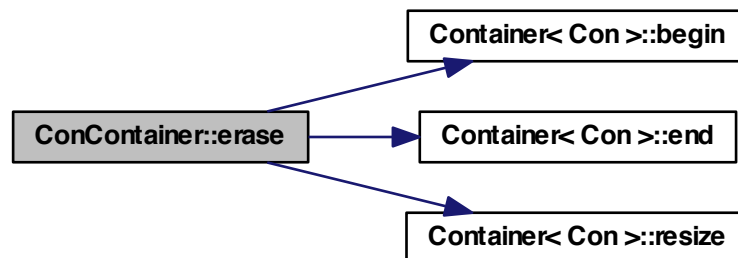
References `Container< Con >::begin()`, `Container< Con >::end()`, and `Container<`



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().

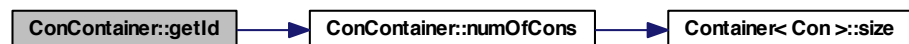
Referenced by validate().

```

{
    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->numOfCon
ns()); // Append numOfCons to result, create new Con and push_back into
conContainer
    ptC.reset( new Con( neuronContainerPtr->l
oad().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&lt;int&gt;</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`.

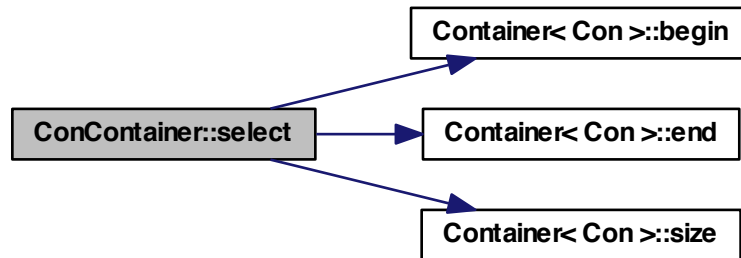
References `Container< Con >::begin()`, `Container< Con >::end()`, and `Container< Con >::size()`.

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&lt;NeuronPtr&gt;</code> with the pointers to be set in the from fields of the <a href="#">ConContainer</a> 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()`.

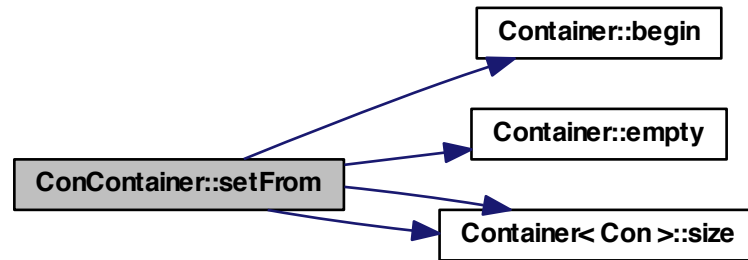
```

{
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 <a href="#">Con</a> objects contained in the <a href="#">ConContainer</a> object.
<i>vFrom</i>	An <code>std::vector&lt;int&gt;</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`.

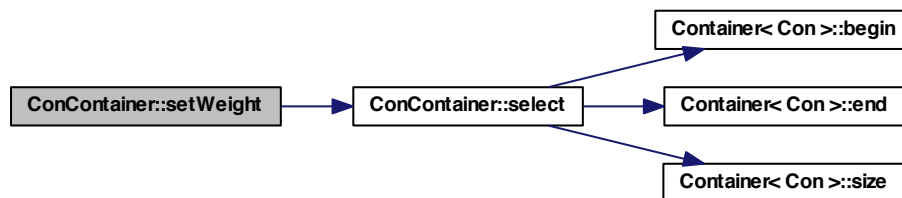
References `select()`.

```

{
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 <a href="#">Con</a> objects contained in the <a href="#">ConContainer</a> object.
-----------------	---

#### Returns

true in case no exception is thrown

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

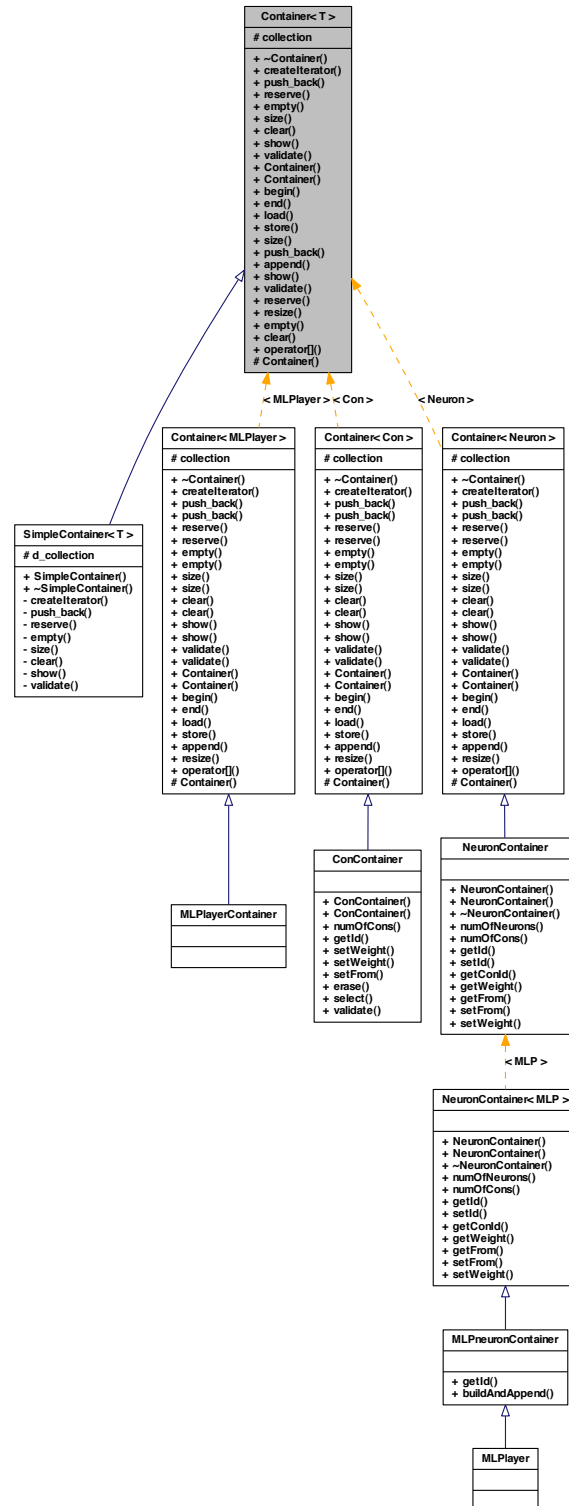
- pkg/AMORE/src/old/[ConContainer.h](#)
- pkg/AMORE/src/old/[ConContainer.cpp](#)

## 5.10 Container< T > Class Template Reference

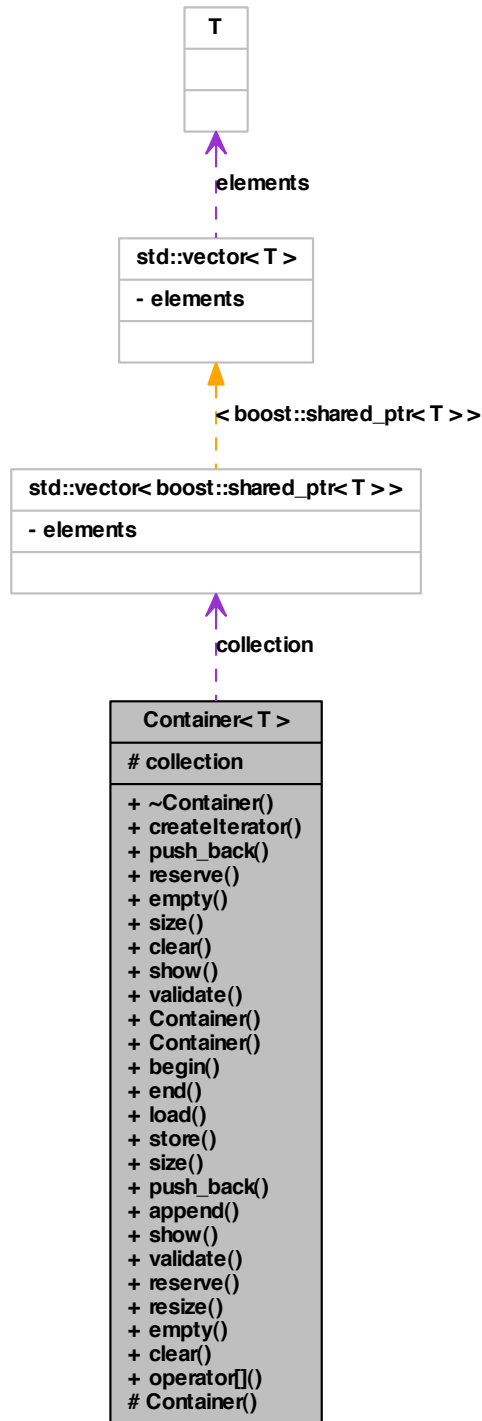
class [Container](#) -

```
#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

<b>v</b>	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`.

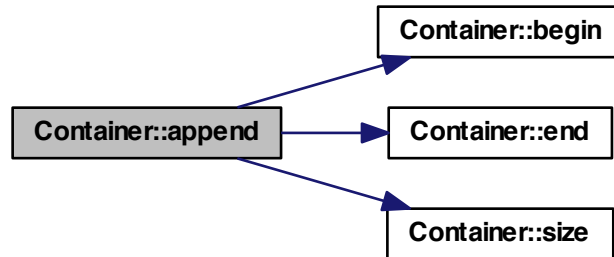
References `Container< T >::begin()`, `Container< T >::end()`, and `Container< T >::size()`.

```

{
    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 ( )`

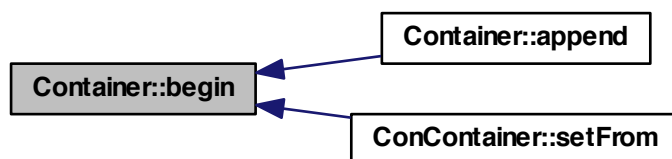
Definition at line 22 of file Container.cpp.

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`.

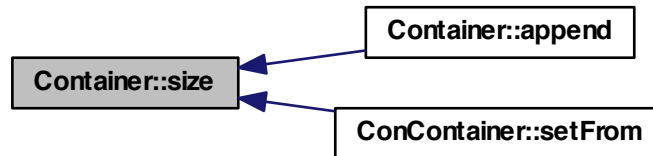
Referenced by `Container< T >::append()`, and `ConContainer::setFrom()`.

```

{
    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

<a href="#">v</a>	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:

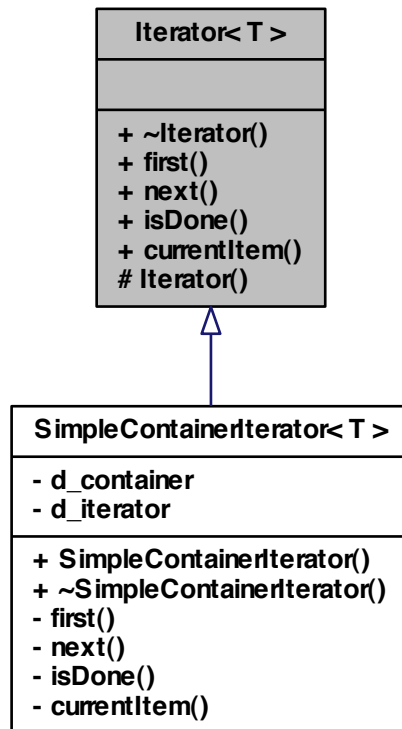
- `pkg/AMORE/src/dia/Container.h`
- `pkg/AMORE/src/old/Container.h`
- `pkg/AMORE/src/Container.cpp`
- `pkg/AMORE/src/old/Container.cpp`

## 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:

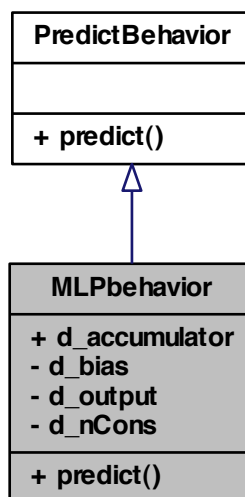
- pkg/AMORE/src/dia/[Iterator.h](#)

## 5.12 MLPbehavior Class Reference

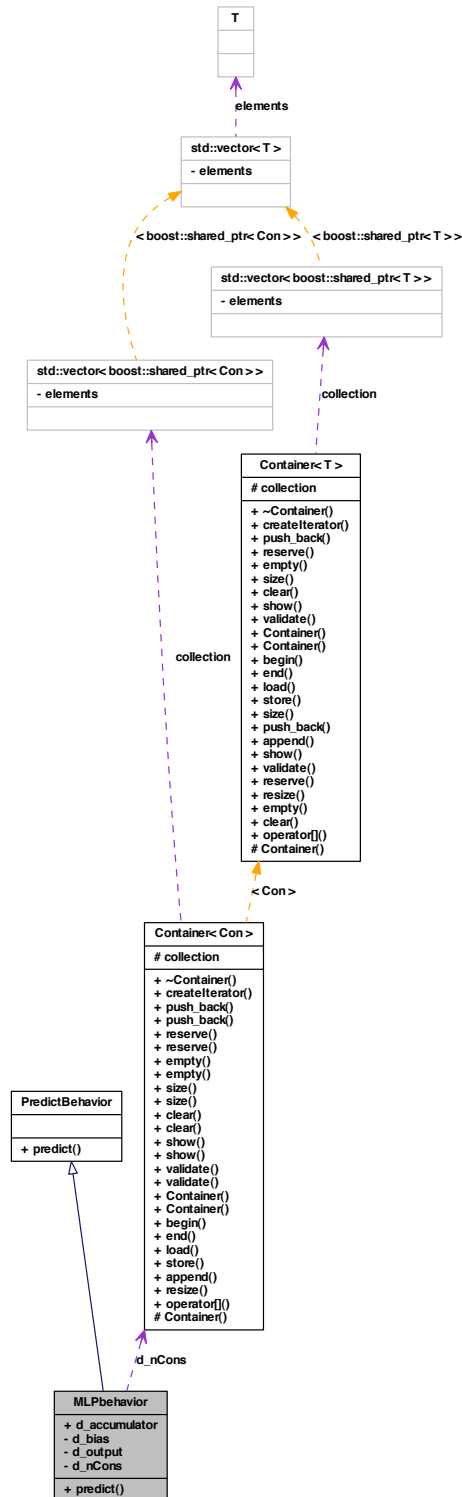
class [MLPbehavior](#) -

```
#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:

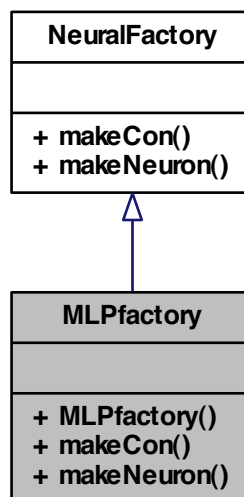
- pkg/AMORE/src/dia/[MLPbehavior.h](#)

### 5.13 MLPfactory Class Reference

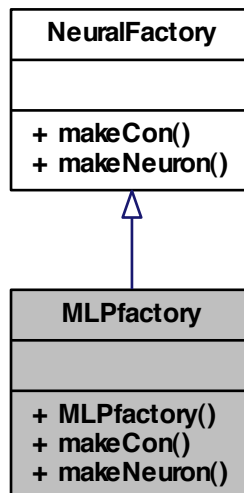
class [MLPfactory](#) -

```
#include <MLPfactory.h>
```

Inheritance diagram for MLPfactory:



Collaboration diagram for MLPfactory:



### Public Member Functions

- [MLPfactory \(\)](#)
- [ConPtr makeCon \(Neuron &neuron\)](#)
- [NeuronPtr makeNeuron \(\)](#)

#### 5.13.1 Detailed Description

class [MLPfactory](#) -

Definition at line 5 of file MLPfactory.h.

#### 5.13.2 Constructor & Destructor Documentation

##### 5.13.2.1 MLPfactory::MLPfactory ( )

Definition at line 13 of file MLPfactory.cpp.

```
{  
}
```

### 5.13.3 Member Function Documentation

#### 5.13.3.1 `ConPtr MLPfactory::makeCon ( Neuron & neuron )` [virtual]

Implements [NeuralFactory](#).

Definition at line 19 of file MLPfactory.cpp.

```
{
    ConPtr conPtr(new Con(neuron));
    return conPtr;
}
```

#### 5.13.3.2 `NeuronPtr MLPfactory::makeNeuron ( )` [virtual]

Implements [NeuralFactory](#).

Definition at line 26 of file MLPfactory.cpp.

```
{
    NeuronPtr neuronPtr(new SimpleNeuron());
    return neuronPtr;
}
```

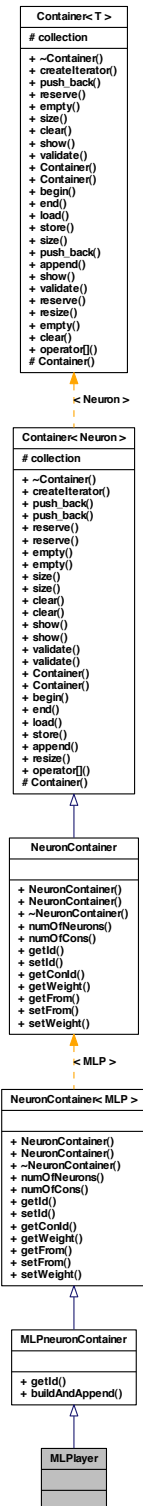
The documentation for this class was generated from the following files:

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

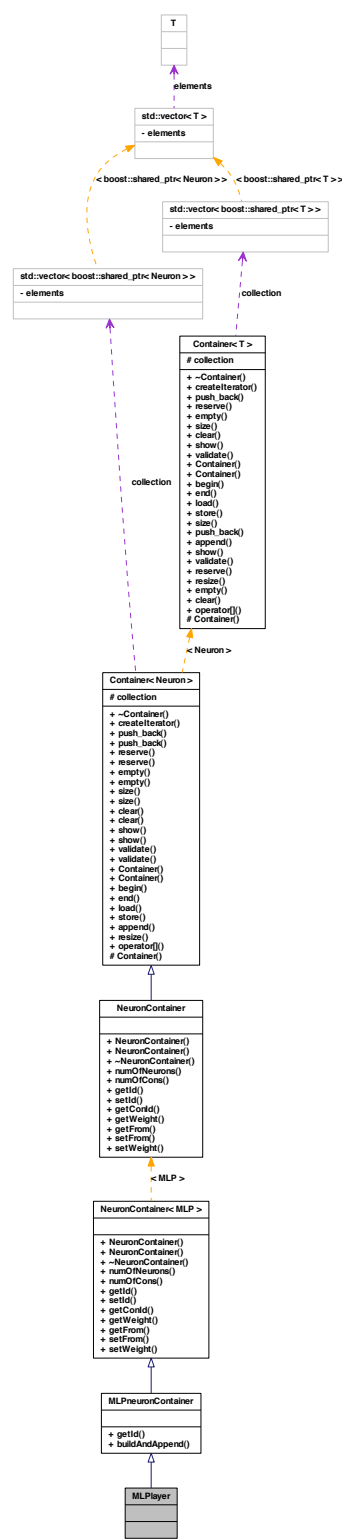
## 5.14 MLPlayer Class Reference

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

Inheritance diagram for MLPlayer:



Collaboration diagram for MLPlayer:



### 5.14.1 Detailed Description

Definition at line 1 of file MLPlayer.h.

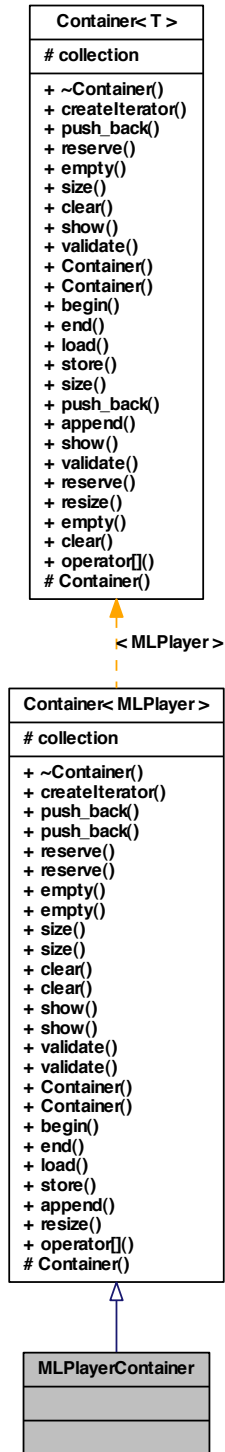
The documentation for this class was generated from the following file:

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

## 5.15 MLPlayerContainer Class Reference

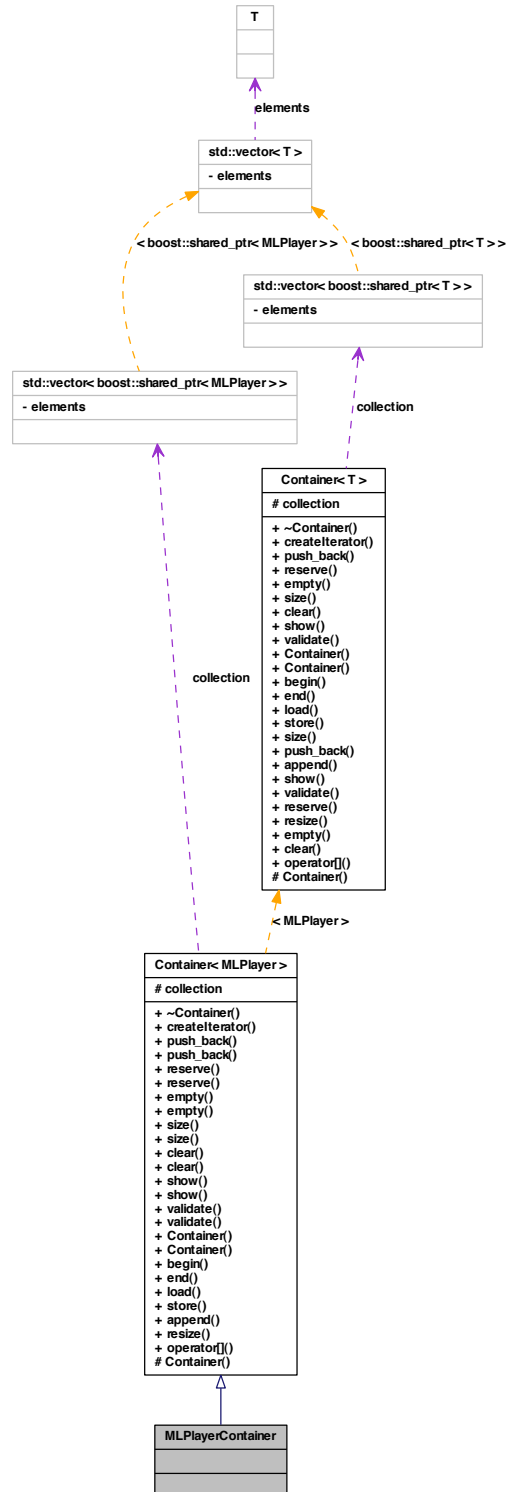
```
#include <MLPlayerContainer.h>
```

Inheritance diagram for MLPlayerContainer:





Collaboration diagram for MLPlayerContainer:



### 5.15.1 Detailed Description

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

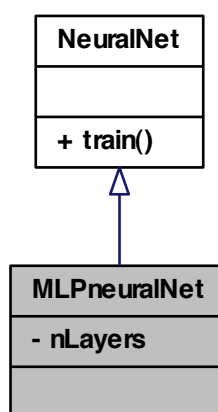
The documentation for this class was generated from the following file:

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

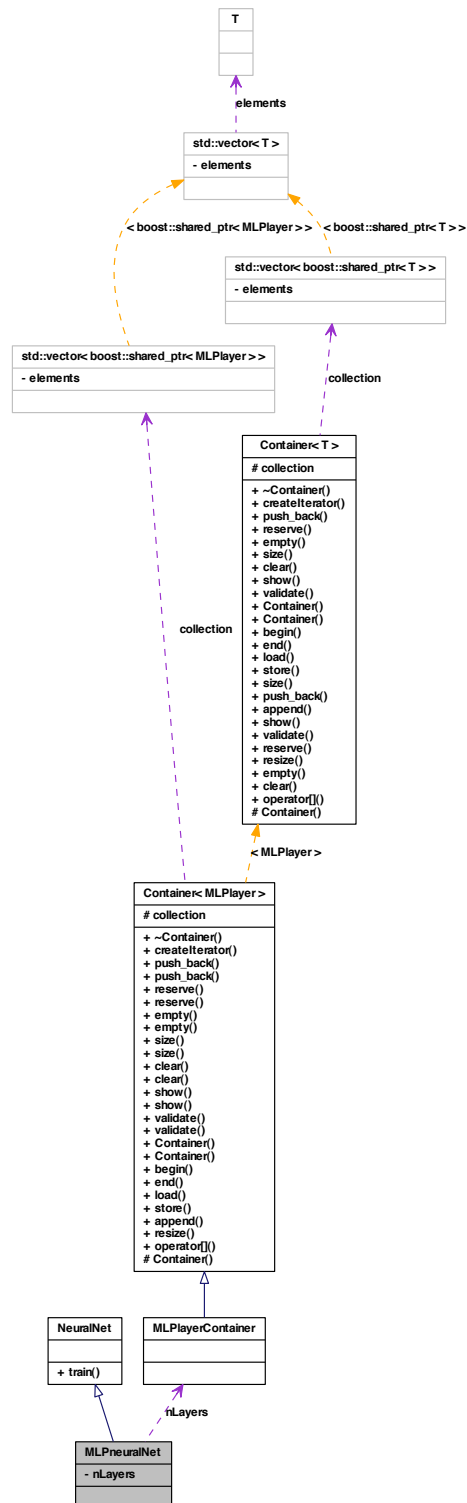
## 5.16 MLPneuralNet Class Reference

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

Inheritance diagram for MLPneuralNet:



Collaboration diagram for MLPneuralNet:



## Private Attributes

- [MLPlayerContainer nLayers](#)

### 5.16.1 Detailed Description

Definition at line 1 of file MLPneuralNet.h.

### 5.16.2 Member Data Documentation

#### 5.16.2.1 MLPlayerContainer MLPneuralNet::nLayers `[private]`

Definition at line 2 of file MLPneuralNet.h.

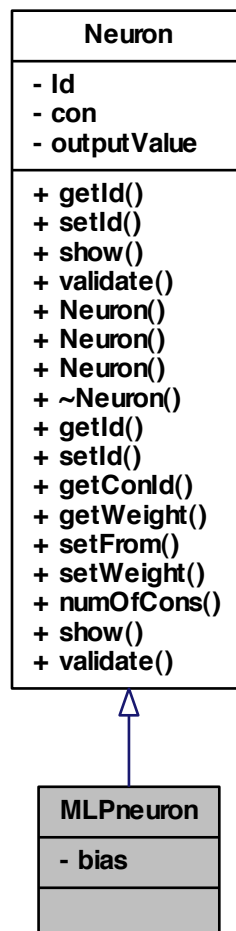
The documentation for this class was generated from the following file:

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

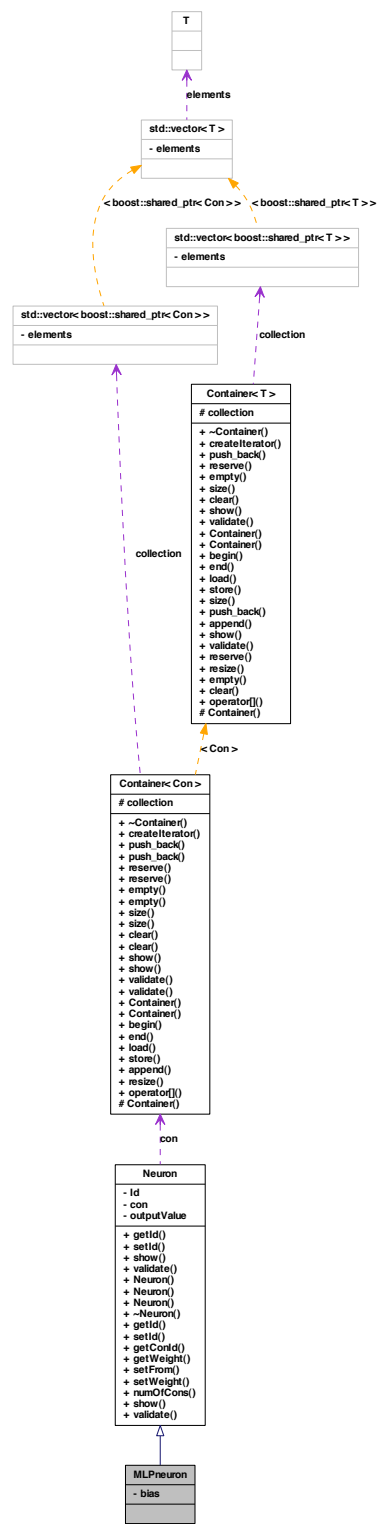
## 5.17 MLPneuron Class Reference

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

Inheritance diagram for MLPneuron:



Collaboration diagram for MLPneuron:



### Private Attributes

- int [bias](#)

#### 5.17.1 Detailed Description

Definition at line 1 of file MLPneuron.h.

#### 5.17.2 Member Data Documentation

##### 5.17.2.1 int MLPneuron::bias [private]

Definition at line 2 of file MLPneuron.h.

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

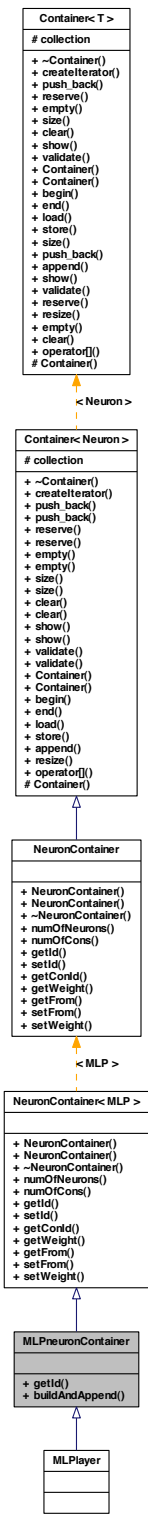
- pkg/AMORE/src/old/[MLPneuron.h](#)

## 5.18 MLPneuronContainer Class Reference

A vector of connections.

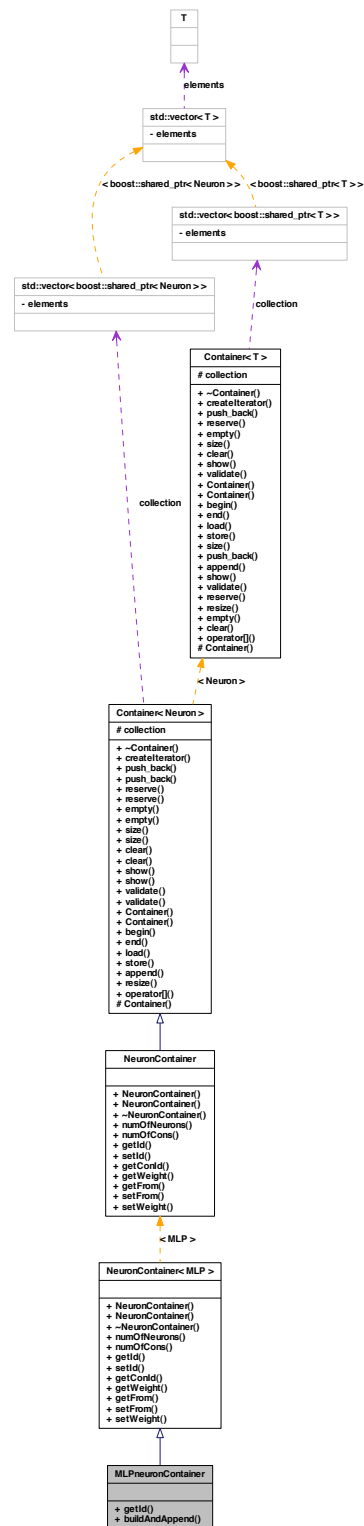
```
#include <MLPneuronContainer.h>
```

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)`

### 5.18.1 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`.

### 5.18.2 Member Function Documentation

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

**5.18.2.2** `std::vector<int> MLPneuronContainer::getId ( )`

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

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

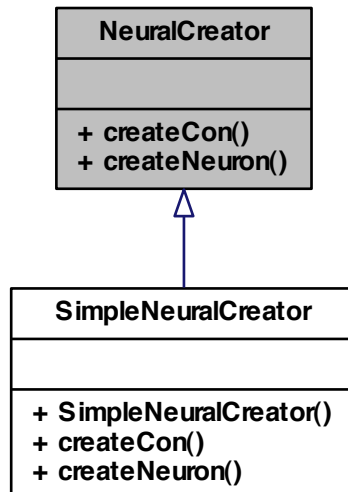
- `pkg/AMORE/src/old/MLPneuronContainer.h`

## 5.19 NeuralCreator Class Reference

class [NeuralCreator](#) -

```
#include <NeuralCreator.h>
```

Inheritance diagram for NeuralCreator:



### Public Member Functions

- virtual `ConPtr createCon (NeuralFactory &neuralFactory, Neuron &neuron)=0`
- virtual `NeuronPtr createNeuron (NeuralFactory &neuralFactory)=0`

#### 5.19.1 Detailed Description

class [NeuralCreator](#) -

Definition at line 4 of file `NeuralCreator.h`.

#### 5.19.2 Member Function Documentation

5.19.2.1 virtual `ConPtr NeuralCreator::createCon ( NeuralFactory & neuralFactory, Neuron & neuron ) [pure virtual]`

Implemented in [SimpleNeuralCreator](#).

5.19.2.2 virtual **NeuronPtr** NeuralCreator::createNeuron ( **NeuralFactory** & *neuralFactory* )  
[pure virtual]

Implemented in [SimpleNeuralCreator](#).

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

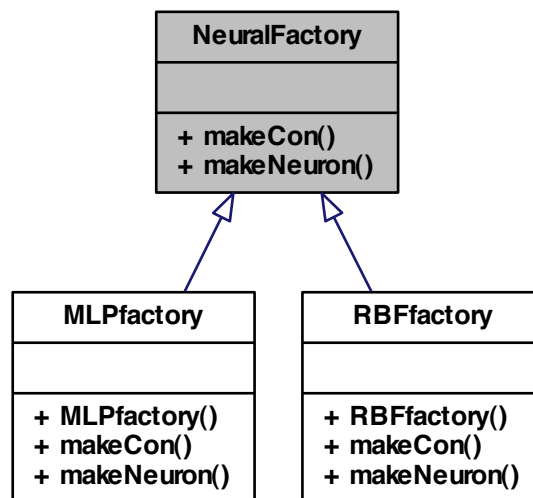
- pkg/AMORE/src/dia/[NeuralCreator.h](#)

## 5.20 NeuralFactory Class Reference

class [NeuralFactory](#) -

```
#include <NeuralFactory.h>
```

Inheritance diagram for NeuralFactory:



### Public Member Functions

- virtual [ConPtr](#) `makeCon` ([Neuron](#) &neuron)=0
- virtual [NeuronPtr](#) `makeNeuron` ()=0

### 5.20.1 Detailed Description

class [NeuralFactory](#) -

Definition at line 4 of file NeuralFactory.h.

### 5.20.2 Member Function Documentation

5.20.2.1 `virtual ConPtr NeuralFactory::makeCon ( Neuron & neuron )` [pure virtual]

Implemented in [MLPfactory](#), and [RBFfactory](#).

Referenced by `SimpleNeuralCreator::createCon()`.

Here is the caller graph for this function:



5.20.2.2 `virtual NeuronPtr NeuralFactory::makeNeuron ( )` [pure virtual]

Implemented in [MLPfactory](#), and [RBFfactory](#).

Referenced by `SimpleNeuralCreator::createNeuron()`.

Here is the caller graph for this function:



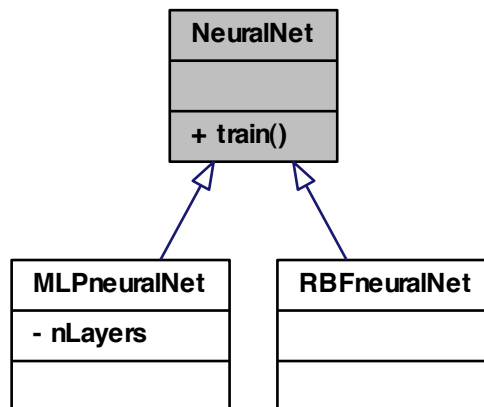
The documentation for this class was generated from the following file:

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

## 5.21 NeuralNet Class Reference

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

Inheritance diagram for NeuralNet:



### Public Member Functions

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

#### 5.21.1 Detailed Description

Definition at line 1 of file NeuralNet.h.

#### 5.21.2 Member Function Documentation

##### 5.21.2.1 virtual void NeuralNet::train ( ) [pure virtual]

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

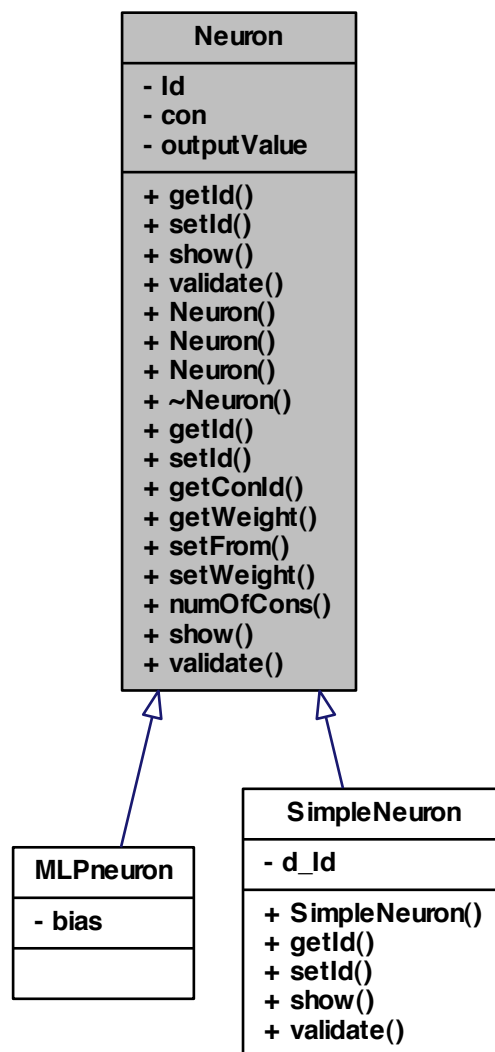
- pkg/AMORE/src/old/[NeuralNet.h](#)

## 5.22 Neuron Class Reference

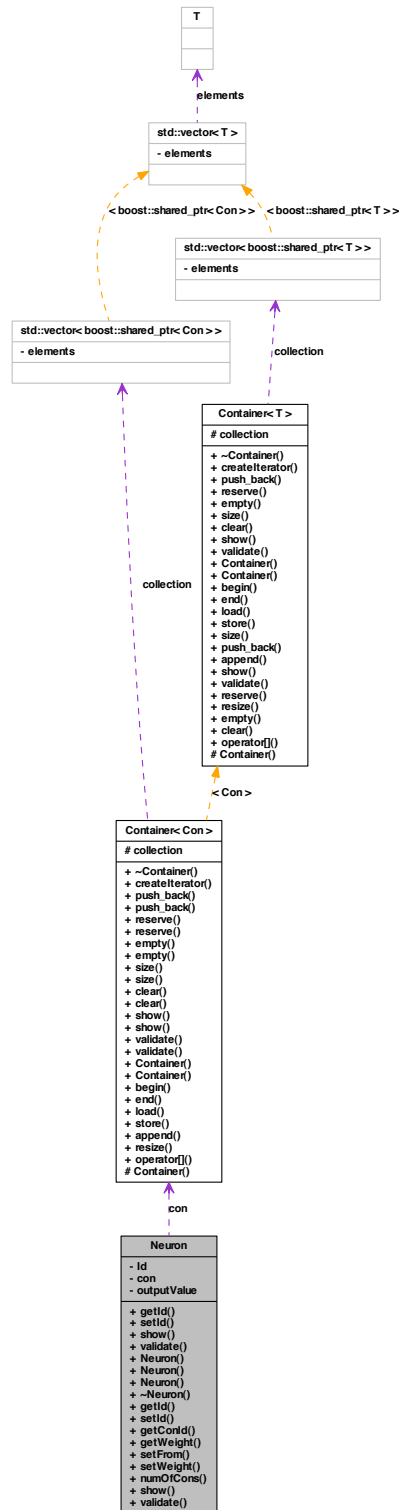
class [Neuron](#) -

```
#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 [show](#) ()=0
- virtual bool [validate](#) ()=0
- [Neuron](#) ()
- [Neuron](#) (int [Id](#))
- [Neuron](#) (int [Id](#), [ConContainer con](#))
- [~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](#) ()

## Private Attributes

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

### 5.22.1 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](#).

### 5.22.2 Constructor & Destructor Documentation

#### 5.22.2.1 [Neuron::Neuron](#) ( )

Definition at line 10 of file [Neuron.cpp](#).

```

        :
        Id (NA_INTEGER), con ()
    {
    }

```

### 5.22.2.2 `Neuron::Neuron ( int Id )`

Definition at line 15 of file `Neuron.cpp`.

```

        :
        Id(Id), con()
    {
    }

```

### 5.22.2.3 `Neuron::Neuron ( int Id, ConContainer con )`

### 5.22.2.4 `Neuron::~~Neuron ( )`

## 5.22.3 Member Function Documentation

### 5.22.3.1 `std::vector<int> Neuron::getConId ( )`

### 5.22.3.2 `int Neuron::getId ( )`

Reimplemented in [SimpleNeuron](#).

### 5.22.3.3 `int Neuron::getId ( )` [pure virtual]

Implemented in [SimpleNeuron](#).

Definition at line 26 of file `Neuron.cpp`.

References `Id`.

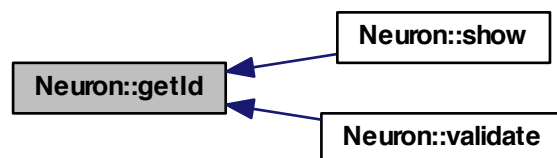
Referenced by `show()`, and `validate()`.

```

{
    return Id;
}

```

Here is the caller graph for this function:



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

5.22.3.5 `int Neuron::numOfCons ( )`

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

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

Implemented in [SimpleNeuron](#).

Definition at line 32 of file Neuron.cpp.

References `Id`.

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

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

Reimplemented in [SimpleNeuron](#).

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

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

Implemented in [SimpleNeuron](#).

Definition at line 46 of file Neuron.cpp.

References `getId()`.

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

```
}
```

Here is the call graph for this function:



#### 5.22.3.11 `bool Neuron::show ( )`

Reimplemented in [SimpleNeuron](#).

#### 5.22.3.12 `bool Neuron::validate ( )` [pure virtual]

Implemented in [SimpleNeuron](#).

Definition at line 73 of file `Neuron.cpp`.

References `getId()`.

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

Here is the call graph for this function:



5.22.3.13 `bool Neuron::validate ( )`

Reimplemented in [SimpleNeuron](#).

## 5.22.4 Member Data Documentation

5.22.4.1 `ConContainer Neuron::con` `[private]`

A vector of input connections.

Definition at line 29 of file `Neuron.h`.

5.22.4.2 `int Neuron::ld` `[private]`

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

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

Definition at line 22 of file `Neuron.h`.

Referenced by `getId()`, `SimpleNeuron::setId()`, and `setId()`.

5.22.4.3 `double Neuron::outputValue` `[private]`

Definition at line 30 of file `Neuron.h`.

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

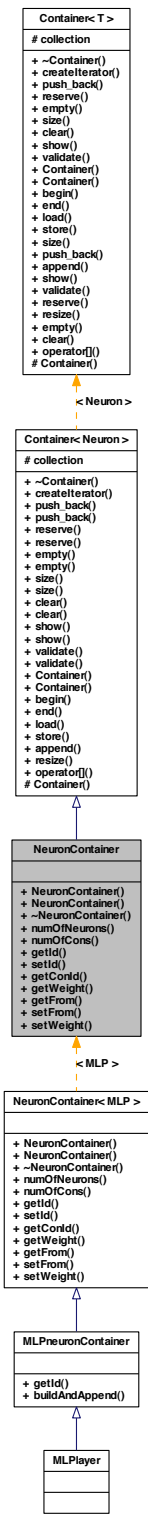
- `pkg/AMORE/src/dia/Neuron.h`
- `pkg/AMORE/src/old/Neuron.h`
- `pkg/AMORE/src/old/Neuron.cpp`

## 5.23 NeuronContainer Class Reference

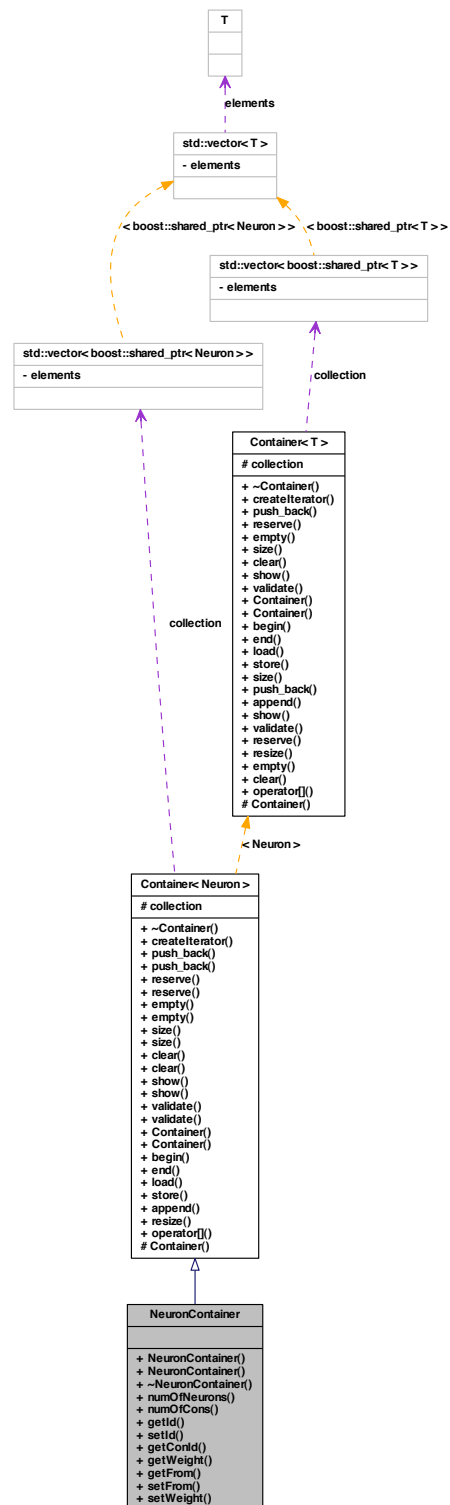
A vector of neurons.

```
#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)

### 5.23.1 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.

### 5.23.2 Member Typedef Documentation

#### 5.23.2.1 typedef NeuronContainer\_const\_iterator NeuronContainer::const\_iterator

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

Definition at line 23 of file NeuronContainer.h.

#### 5.23.2.2 typedef value\_type const& NeuronContainer::const\_reference

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

Definition at line 27 of file NeuronContainer.h.



5.23.2.3 `typedef NeuronContainer_iterator NeuronContainer::iterator`

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

Definition at line 21 of file NeuronContainer.h.

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

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

Definition at line 25 of file NeuronContainer.h.

## 5.23.3 Constructor &amp; Destructor Documentation

5.23.3.1 `NeuronContainer::NeuronContainer ( )`

Definition at line 8 of file NeuronContainer.cpp.

```
{
}
```

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

Definition at line 12 of file NeuronContainer.cpp.

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

5.23.3.3 `NeuronContainer::~~NeuronContainer ( )`

Definition at line 17 of file NeuronContainer.cpp.

```
{
}
```

## 5.23.4 Member Function Documentation

5.23.4.1 `std::vector< std::vector< int > > NeuronContainer::getConId ( )`

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;
}

```

#### 5.23.4.2 `std::vector<NeuronContainer> NeuronContainer::getFrom ( )`

#### 5.23.4.3 `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;
}

```

#### 5.23.4.4 `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;
}

```

#### 5.23.4.5 `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;
}

```

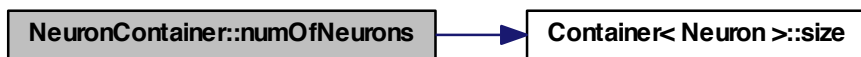
## 5.23.4.6 int NeuronContainer::numOfNeurons ( )

Definition at line 22 of file NeuronContainer.cpp.

References Container< Neuron >::size().

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

Here is the call graph for this function:



## 5.23.4.7 void NeuronContainer::setFrom ( std::vector&lt; NeuronContainer &gt; neuronArray )

Definition at line 83 of file NeuronContainer.cpp.

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

## 5.23.4.8 void NeuronContainer::setId ( std::vector&lt; int &gt; nIds )

Definition at line 50 of file NeuronContainer.cpp.

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

#### 5.23.4.9 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:

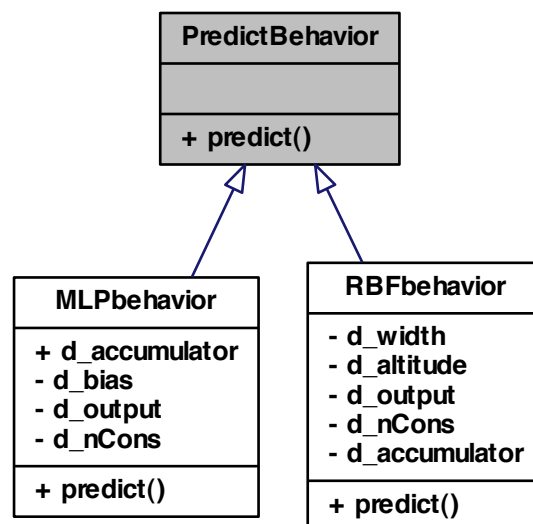
- pkg/AMORE/src/old/[NeuronContainer.h](#)
- pkg/AMORE/src/old/[NeuronContainer.cpp](#)

## 5.24 PredictBehavior Class Reference

class [PredictBehavior](#) -

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

Inheritance diagram for PredictBehavior:



## Public Member Functions

- void [predict](#) ()

### 5.24.1 Detailed Description

class [PredictBehavior](#) -

Definition at line 4 of file [PredictBehavior.h](#).

### 5.24.2 Member Function Documentation

#### 5.24.2.1 void [PredictBehavior::predict](#) ( )

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

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

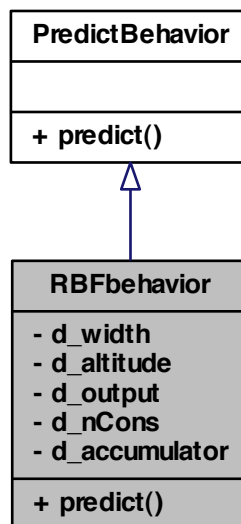
- [pkg/AMORE/src/dia/PredictBehavior.h](#)

## 5.25 RBFbehavior Class Reference

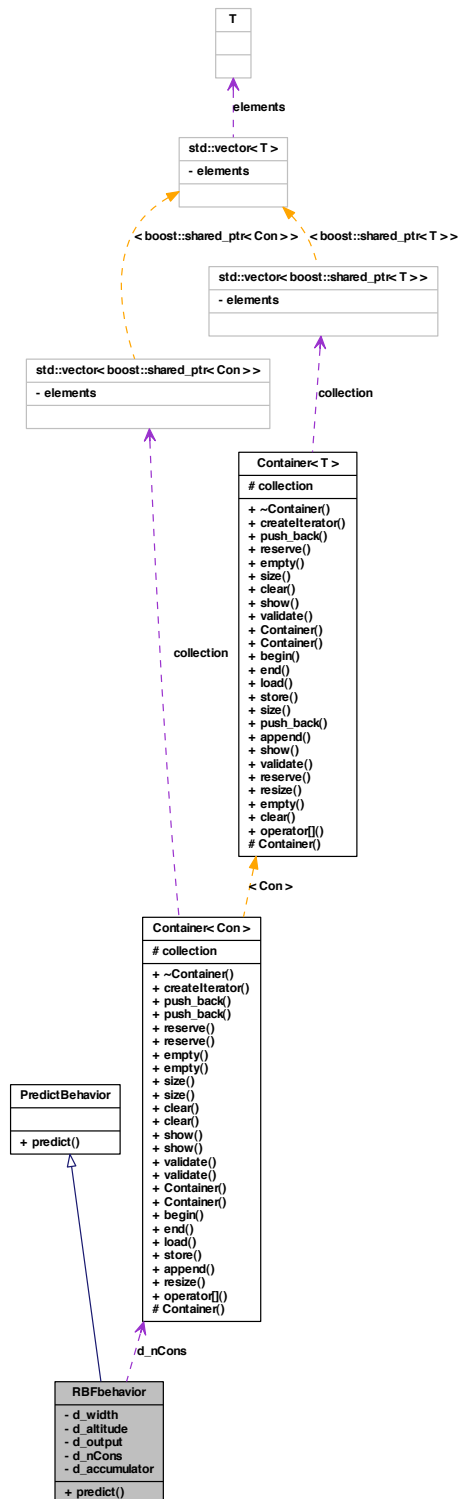
class [RBFbehavior](#) -

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

Inheritance diagram for RBFbehavior:



Generated on Thu Jul 14 2011 19:12:28 for AMORE++ by Doxygen



## Public Member Functions

- void [predict](#) ()

## Private Attributes

- double [d\\_width](#)
- double [d\\_altitude](#)
- double [d\\_output](#)
- [Container](#)< [Con](#) > [d\\_nCons](#)
- double [d\\_accumulator](#)

### 5.25.1 Detailed Description

class [RBFbehavior](#) -

Definition at line 5 of file RBFbehavior.h.

### 5.25.2 Member Function Documentation

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

Reimplemented from [PredictBehavior](#).

### 5.25.3 Member Data Documentation

#### 5.25.3.1 double [RBFbehavior::d\\_accumulator](#) [private]

Definition at line 12 of file RBFbehavior.h.

#### 5.25.3.2 double [RBFbehavior::d\\_altitude](#) [private]

Definition at line 9 of file RBFbehavior.h.

#### 5.25.3.3 [Container](#)<[Con](#)> [RBFbehavior::d\\_nCons](#) [private]

Definition at line 11 of file RBFbehavior.h.

#### 5.25.3.4 double [RBFbehavior::d\\_output](#) [private]

Definition at line 10 of file RBFbehavior.h.



### 5.25.3.5 double RBFbehavior::d\_width [private]

Definition at line 8 of file RBFbehavior.h.

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

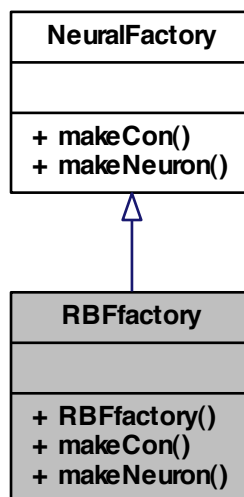
- [pkg/AMORE/src/dia/RBFbehavior.h](#)

## 5.26 RBFfactory Class Reference

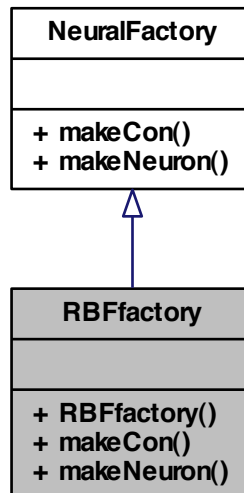
class [RBFfactory](#) -

```
#include <RBFfactory.h>
```

Inheritance diagram for RBFfactory:



Collaboration diagram for RBFactory:



### Public Member Functions

- [RBFactory](#) ()
- [ConPtr makeCon](#) ([Neuron](#) &neuron)
- [NeuronPtr makeNeuron](#) ()

### 5.26.1 Detailed Description

class [RBFactory](#) -

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

### 5.26.2 Constructor & Destructor Documentation

#### 5.26.2.1 [RBFactory::RBFactory](#) ( )

### 5.26.3 Member Function Documentation

#### 5.26.3.1 [ConPtr RBFactory::makeCon](#) ( [Neuron](#) & *neuron* ) [virtual]

Implements [NeuralFactory](#).

### 5.26.3.2 NeuronPtr RBFfactory::makeNeuron ( ) [virtual]

Implements [NeuralFactory](#).

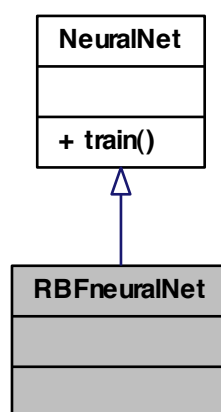
The documentation for this class was generated from the following file:

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

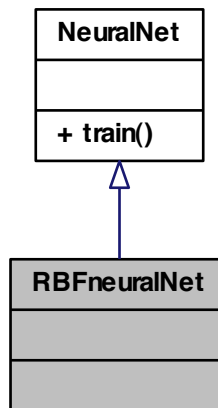
## 5.27 RBFneuralNet Class Reference

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

Inheritance diagram for RBFneuralNet:



Collaboration diagram for RBFneuralNet:



### 5.27.1 Detailed Description

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

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

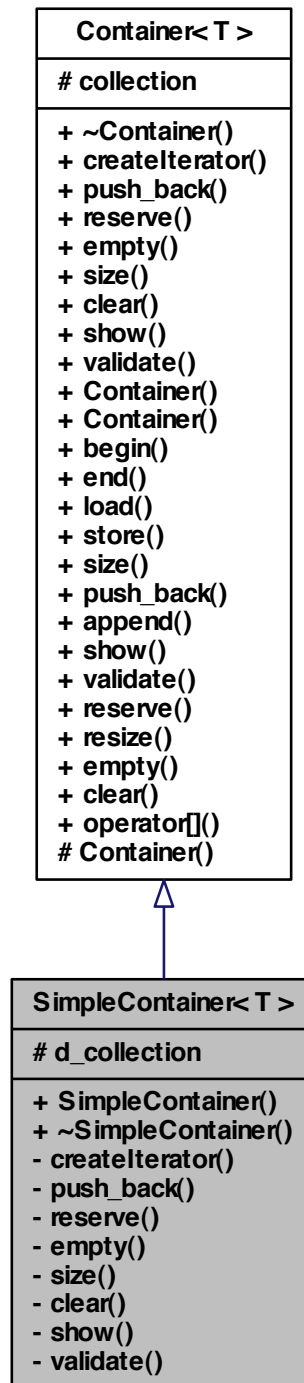
- `pkg/AMORE/src/old/RBFneuralNet.h`

## 5.28 SimpleContainer< T > Class Template Reference

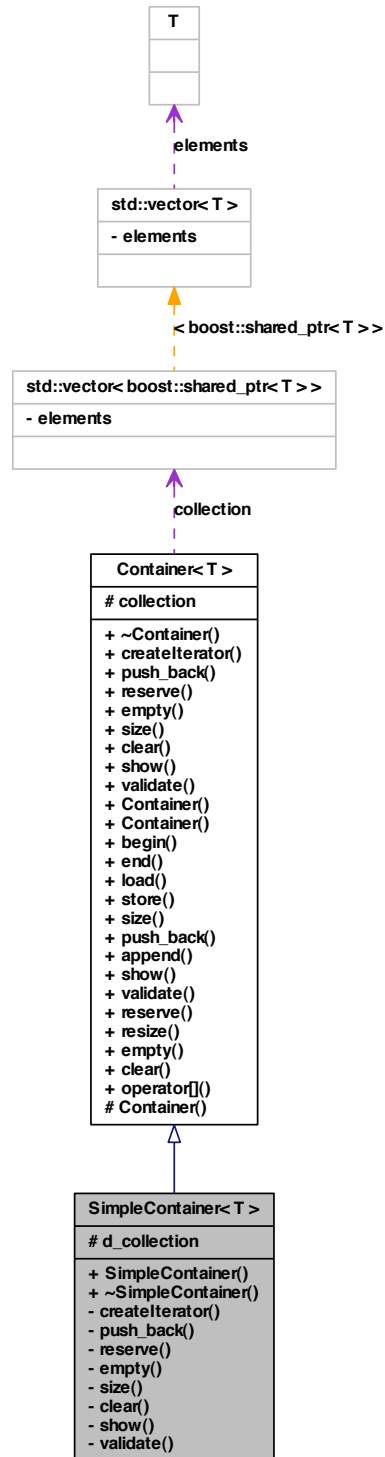
class `SimpleContainer` -

```
#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 >

### 5.28.1 Detailed Description

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

class [SimpleContainer](#) -

Definition at line 6 of file SimpleContainer.h.

### 5.28.2 Constructor & Destructor Documentation

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

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

### 5.28.3 Member Function Documentation

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

Implements [Container<T>](#).

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

Implements [Container<T>](#).

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

Implements [Container<T>](#).

**5.28.3.4** `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 >](#).

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

Implements [Container< T >](#).

**5.28.3.6** `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 >](#).

**5.28.3.7** `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 >](#).

**5.28.3.8** `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 >](#).

**5.28.4 Friends And Related Function Documentation**

**5.28.4.1** `template<typename T> friend class SimpleContainerIterator< T >`  
`[friend]`

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

**5.28.5 Member Data Documentation**

5.28.5.1 `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:

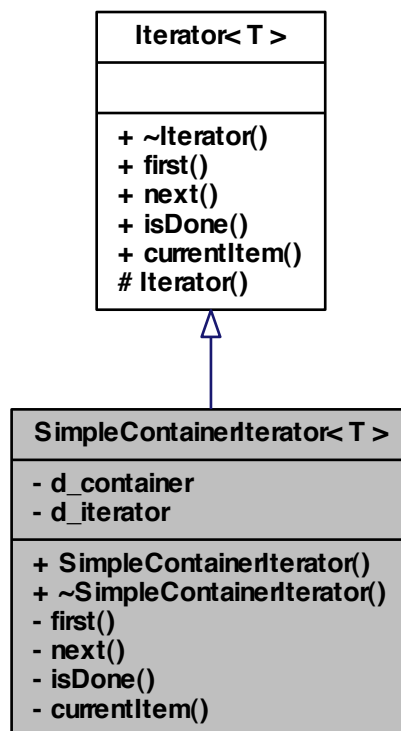
- `pkg/AMORE/src/dia/SimpleContainer.h`

## 5.29 SimpleContainerIterator< T > Class Template Reference

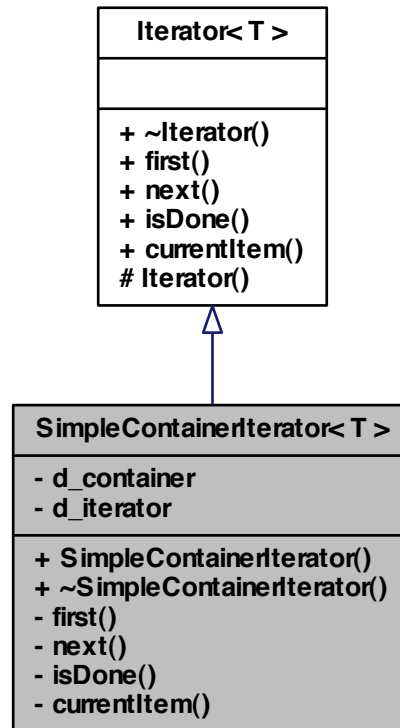
class [SimpleContainerIterator](#) -

```
#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 >](#)

#### 5.29.1 Detailed Description

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

class [SimpleContainerIterator](#) -

Definition at line 6 of file SimpleContainerIterator.h.

#### 5.29.2 Constructor & Destructor Documentation

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

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

#### 5.29.3 Member Function Documentation

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

Implements [Iterator< T >](#).

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

Implements [Iterator< T >](#).

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

Implements [Iterator< T >](#).

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

Implements [Iterator< T>](#).

## 5.29.4 Friends And Related Function Documentation

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

Definition at line 13 of file SimpleContainerIterator.h.

## 5.29.5 Member Data Documentation

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

Definition at line 9 of file SimpleContainerIterator.h.

5.29.5.2 `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:

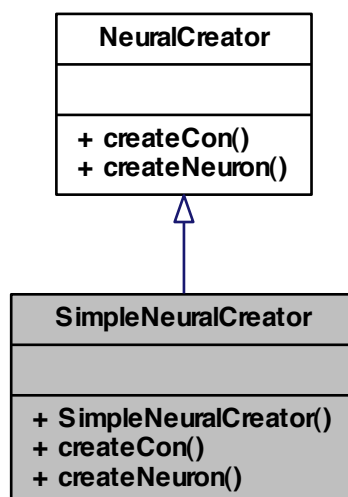
- pkg/AMORE/src/dia/[SimpleContainerIterator.h](#)

## 5.30 SimpleNeuralCreator Class Reference

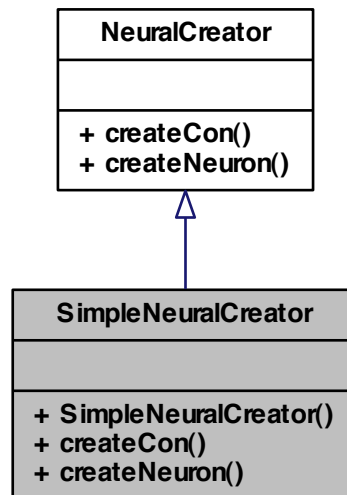
class [SimpleNeuralCreator](#) -

```
#include <SimpleNeuralCreator.h>
```

Inheritance diagram for SimpleNeuralCreator:



Collaboration diagram for SimpleNeuralCreator:



### Public Member Functions

- [SimpleNeuralCreator](#) ()
- [ConPtr](#) [createCon](#) ([NeuralFactory](#) &neuralFactory, [Neuron](#) &neuron)
- [NeuronPtr](#) [createNeuron](#) ([NeuralFactory](#) &neuralFactory)

### 5.30.1 Detailed Description

class [SimpleNeuralCreator](#) -

Definition at line 5 of file SimpleNeuralCreator.h.

### 5.30.2 Constructor & Destructor Documentation

#### 5.30.2.1 SimpleNeuralCreator::SimpleNeuralCreator ( )

Definition at line 15 of file SimpleNeuralCreator.cpp.

```
{  
}
```



### 5.30.3 Member Function Documentation

#### 5.30.3.1 ConPtr SimpleNeuralCreator::createCon ( NeuralFactory & *neuralFactory*, Neuron & *neuron* ) [virtual]

Implements [NeuralCreator](#).

Definition at line 21 of file SimpleNeuralCreator.cpp.

References [NeuralFactory::makeCon\(\)](#).

```
{  
    return neuralFactory.makeCon(neuron);  
}
```

Here is the call graph for this function:



#### 5.30.3.2 NeuronPtr SimpleNeuralCreator::createNeuron ( NeuralFactory & *neuralFactory* ) [virtual]

Implements [NeuralCreator](#).

Definition at line 28 of file SimpleNeuralCreator.cpp.

References [NeuralFactory::makeNeuron\(\)](#).

```
{  
    return neuralFactory.makeNeuron();  
}
```

Here is the call graph for this function:



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

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

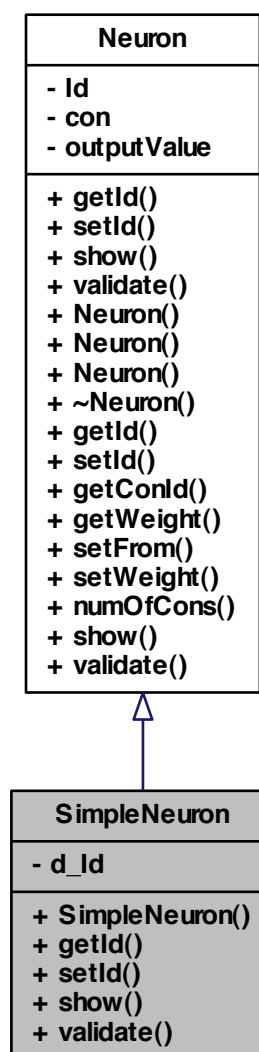
- pkg/AMORE/src/[SimpleNeuralCreator.cpp](#)

## 5.31 SimpleNeuron Class Reference

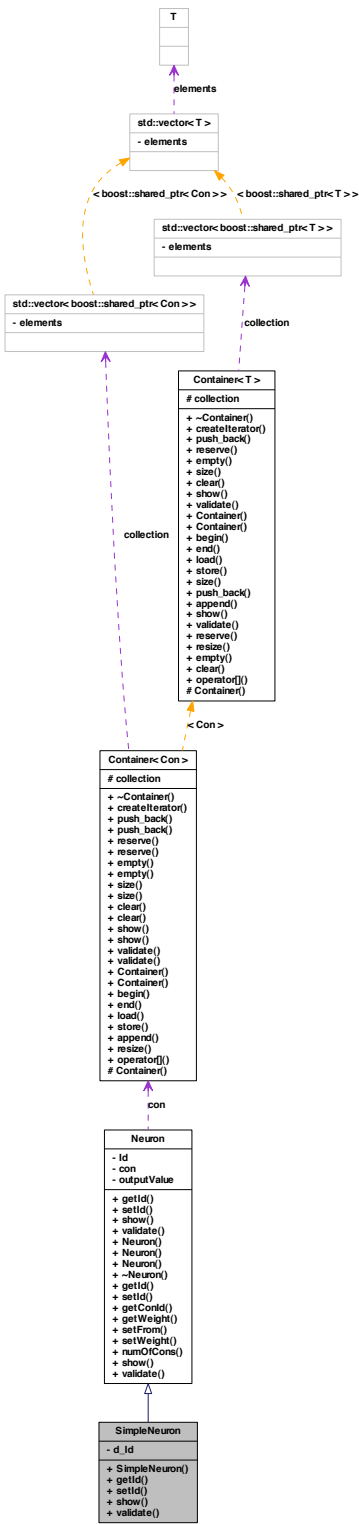
class [SimpleNeuron](#) -

```
#include <SimpleNeuron.h>
```

Inheritance diagram for SimpleNeuron:



Collaboration diagram for SimpleNeuron:



## Public Member Functions

- [SimpleNeuron](#) ()
- [Handler](#) [getId](#) ()
- void [setId](#) ([Handler](#) [Id](#))
- void [show](#) ()
- bool [validate](#) ()

## Private Attributes

- int [d\\_Id](#)

### 5.31.1 Detailed Description

class [SimpleNeuron](#) -

Definition at line 5 of file SimpleNeuron.h.

### 5.31.2 Constructor & Destructor Documentation

#### 5.31.2.1 SimpleNeuron::SimpleNeuron ( )

Definition at line 10 of file SimpleNeuron.cpp.

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

### 5.31.3 Member Function Documentation

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

Implements [Neuron](#).

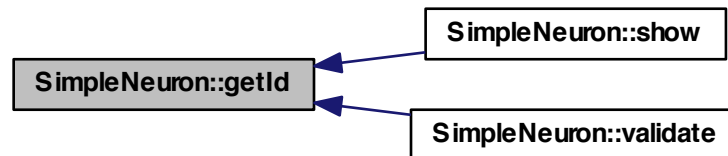
Definition at line 28 of file SimpleNeuron.cpp.

References [d\\_Id](#).

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

```
{  
    return d_Id;  
}
```

Here is the caller graph for this function:



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

Implements [Neuron](#).

Definition at line 36 of file SimpleNeuron.cpp.

References `d_Id`, and `Neuron::Id`.

```

{
    d_Id=Id;
}
  
```

#### 5.31.3.3 void SimpleNeuron::show ( ) [virtual]

Implements [Neuron](#).

Definition at line 59 of file SimpleNeuron.cpp.

References `getId()`.

```

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

        if (nCons.size() == 0)
        {
  
```

```
        Rprintf("\n No connections defined");
    }
    else
    {
        nCons.show();
    }
    Rprintf("\n-----\n");
#endif
}
```

Here is the call graph for this function:



#### 5.31.3.4 bool SimpleNeuron::validate ( ) [virtual]

Implements [Neuron](#).

Definition at line 87 of file SimpleNeuron.cpp.

References `getId()`.

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

Here is the call graph for this function:



### 5.31.4 Member Data Documentation

#### 5.31.4.1 `int SimpleNeuron::d_Id` [private]

Definition at line 8 of file SimpleNeuron.h.

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

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

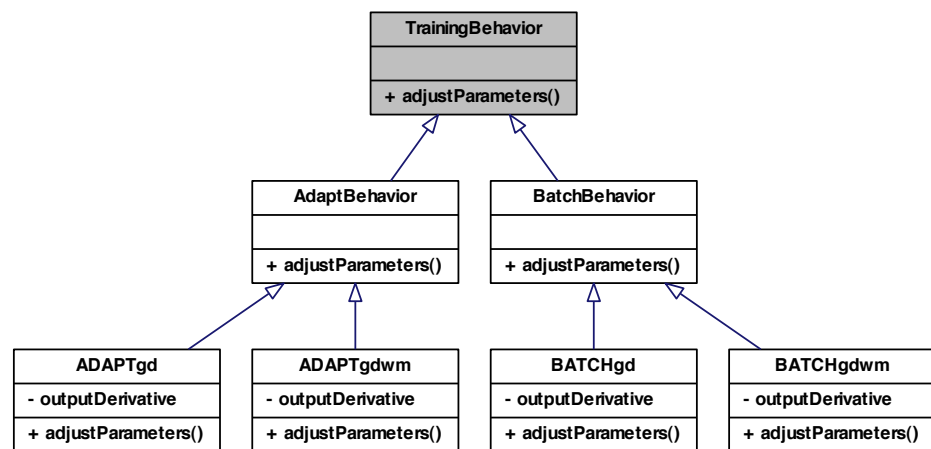
- pkg/AMORE/src/dia/[SimpleNeuron.h](#)
- pkg/AMORE/src/[SimpleNeuron.cpp](#)

## 5.32 TrainingBehavior Class Reference

class [TrainingBehavior](#) -

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

Inheritance diagram for TrainingBehavior:



### Public Member Functions

- void [adjustParameters](#) ()

#### 5.32.1 Detailed Description

class [TrainingBehavior](#) -



Definition at line 4 of file TrainingBehavior.h.

### 5.32.2 Member Function Documentation

#### 5.32.2.1 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 "dia/Neuron.h"
#include "dia/SimpleNeuron.h"
#include "dia/NeuralFactory.h"
#include "dia/MLPFactory.h"
#include "dia/NeuralCreator.h"
#include "dia/SimpleNeuralCreator.h"
#include "Con.cpp"
#include "SimpleNeuron.cpp"
#include "MLPfactory.cpp"
#include "SimpleNeuralCreator.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< [Iterator](#)< [Neuron](#) > > [NeuronIteratorPtr](#)
- `typedef` boost::shared\_ptr< [Iterator](#)< [Con](#) > > [ConIteratorPtr](#)
- `typedef` boost::shared\_ptr< [NeuralFactory](#) > [NeuralFactoryPtr](#)
- `typedef` boost::shared\_ptr< [NeuralCreator](#) > [NeuralCreatorPtr](#)
- `typedef` [Container](#)< [Con](#) > [ConContainer](#)
- `typedef` [Container](#)< [Neuron](#) > [NeuronContainer](#)

### 6.1.1 Define Documentation

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

Definition at line 61 of file AMORE.h.

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

Definition at line 64 of file AMORE.h.

### 6.1.2 Typedef Documentation

#### 6.1.2.1 `typedef` [Container](#)<[Con](#)> [ConContainer](#)

Definition at line 99 of file AMORE.h.

**6.1.2.2 typedef boost::shared\_ptr< Iterator<Con> > ConIteratorPtr**

Definition at line 79 of file AMORE.h.

**6.1.2.3 typedef boost::shared\_ptr<Con> ConPtr**

Definition at line 77 of file AMORE.h.

**6.1.2.4 typedef int Handler**

Definition at line 67 of file AMORE.h.

**6.1.2.5 typedef boost::shared\_ptr< NeuralCreator > NeuralCreatorPtr**

Definition at line 81 of file AMORE.h.

**6.1.2.6 typedef boost::shared\_ptr< NeuralFactory > NeuralFactoryPtr**

Definition at line 80 of file AMORE.h.

**6.1.2.7 typedef Container<Neuron> NeuronContainer**

Definition at line 100 of file AMORE.h.

**6.1.2.8 typedef boost::shared\_ptr< Iterator<Neuron> > NeuronIteratorPtr**

Definition at line 78 of file AMORE.h.

**6.1.2.9 typedef boost::shared\_ptr<Neuron> NeuronPtr**

Definition at line 76 of file AMORE.h.

**6.1.2.10 typedef boost::reference\_wrapper<Neuron> NeuronRef**

Definition at line 72 of file AMORE.h.

**6.1.2.11 typedef boost::reference\_wrapper<PredictBehavior> PredictBehaviorRef**

Definition at line 70 of file AMORE.h.

### 6.1.2.12 `typedef boost::reference_wrapper<TrainingBehavior> TrainingBehaviorRef`

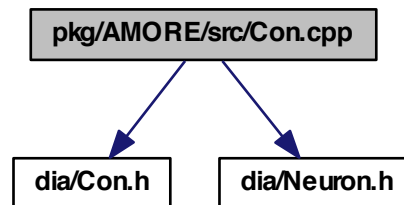
Definition at line 71 of file AMORE.h.

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

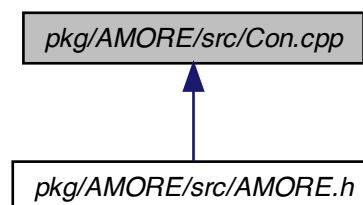
```
#include "dia/Con.h"
```

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

Include dependency graph for `Con.cpp`:



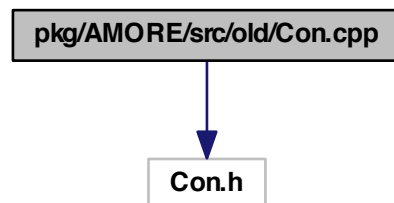
This graph shows which files directly or indirectly include this file:



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

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

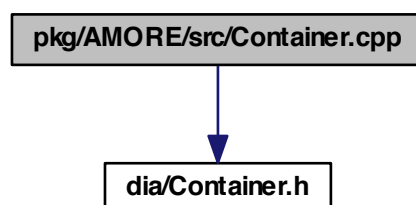
Include dependency graph for Con.cpp:



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

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

Include dependency graph for Container.cpp:

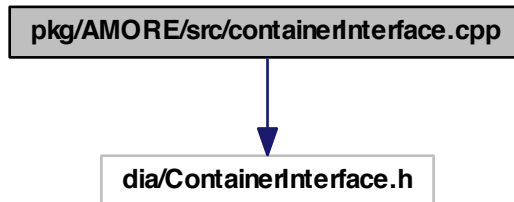


## 6.5 pkg/AMORE/src/old/Container.cpp File Reference

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

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

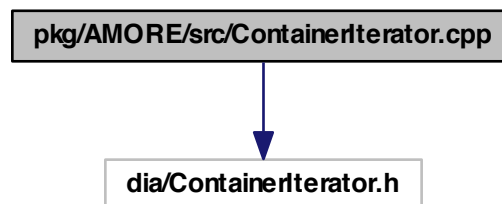
Include dependency graph for containerInterface.cpp:



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

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

Include dependency graph for ContainerIterator.cpp:

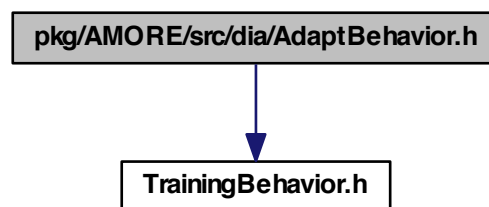


## 6.8 pkg/AMORE/src/dia/AdaptBehavior.h File Reference

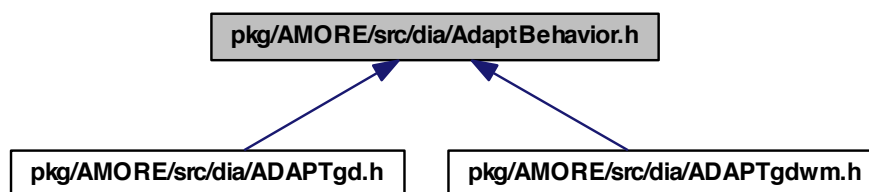
```
#include "TrainingBehavior.h"
```



Include dependency graph for AdaptBehavior.h:



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



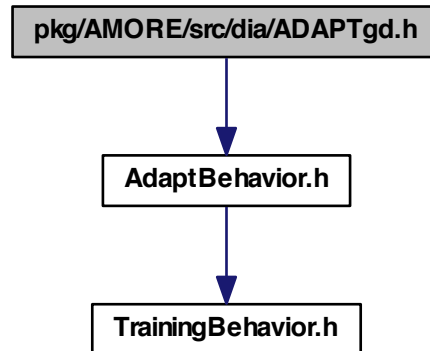
## Classes

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

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

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

Include dependency graph for ADAPTgd.h:



## Classes

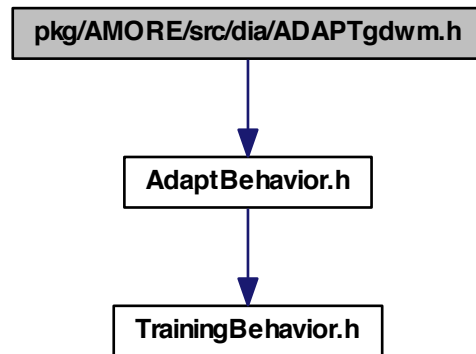
- class [ADAPTgd](#)

*class [ADAPTgd](#) -*

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

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

Include dependency graph for ADAPTgdwm.h:



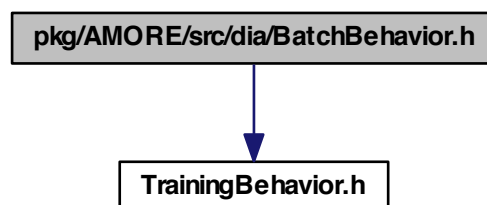
## Classes

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

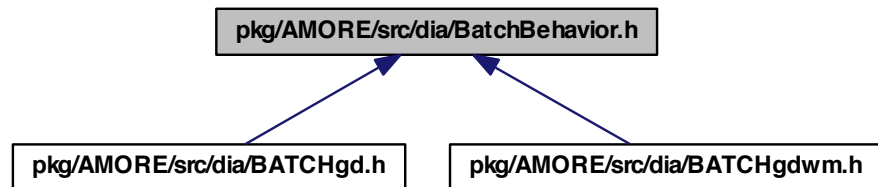
## 6.11 pkg/AMORE/src/dia/BatchBehavior.h File Reference

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

Include dependency graph for BatchBehavior.h:



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



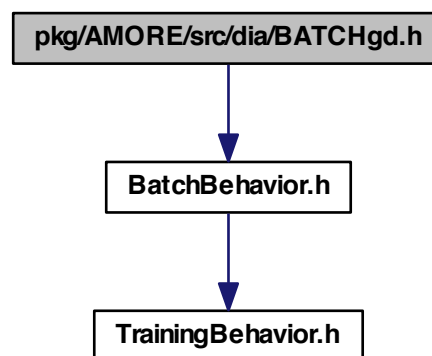
## Classes

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

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

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

Include dependency graph for BATCHgd.h:



## Classes

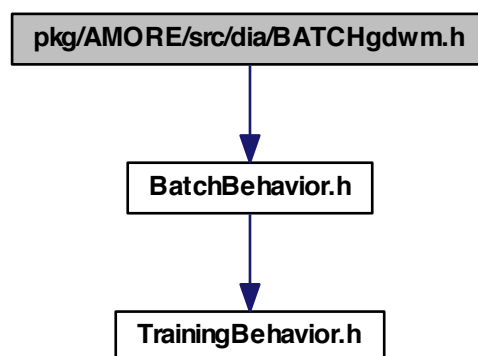
- class [BATCHgd](#)

*class [BATCHgd](#) -*

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

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

Include dependency graph for BATCHgdwm.h:



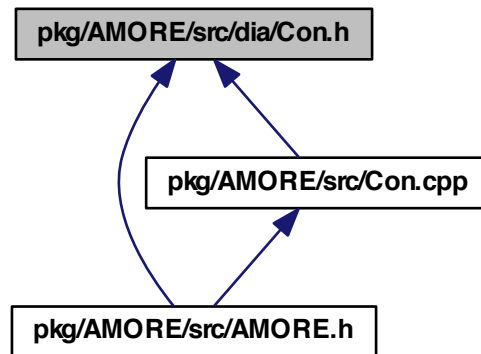
## Classes

- 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:

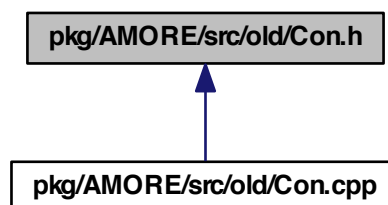


### Classes

- 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:



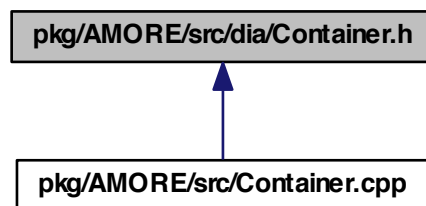
## Classes

- 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:



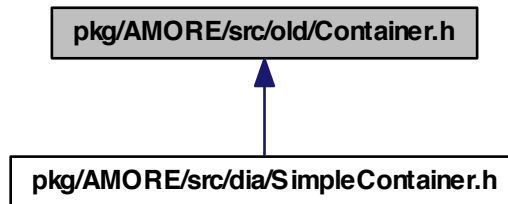
## Classes

- 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:

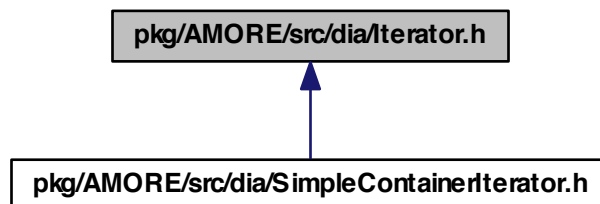


### Classes

- 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 >](#)

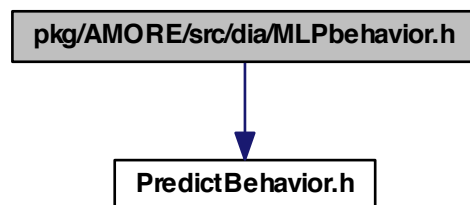


class [Iterator](#) -

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

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

Include dependency graph for MLPbehavior.h:



### Classes

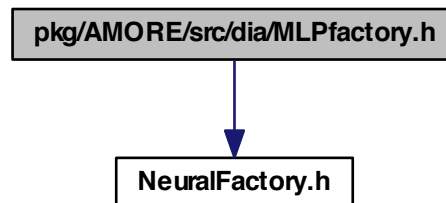
- class [MLPbehavior](#)

class [MLPbehavior](#) -

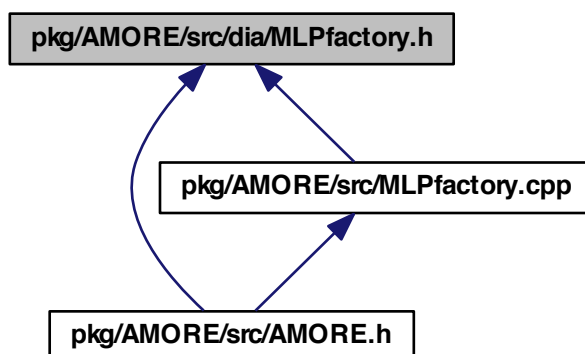
## 6.20 pkg/AMORE/src/dia/MLPfactory.h File Reference

```
#include "NeuralFactory.h"
```

Include dependency graph for MLPfactory.h:



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

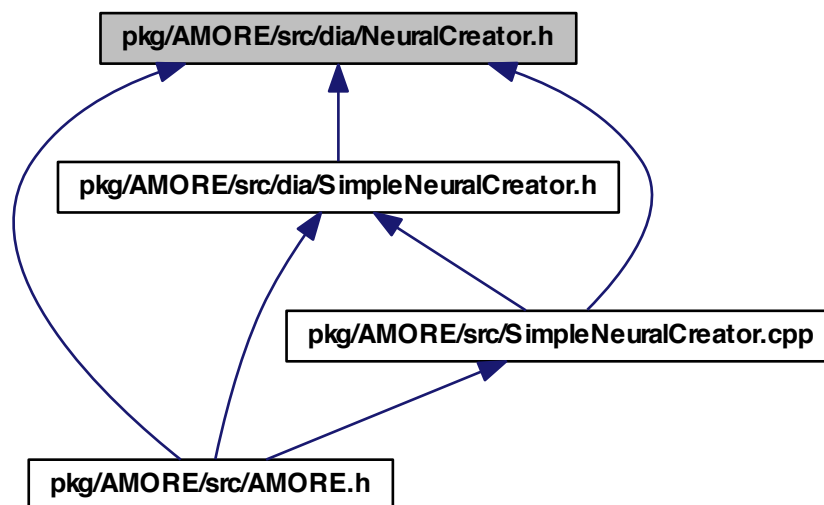


## Classes

- class [MLPfactory](#)  
*class MLPfactory -*

## 6.21 pkg/AMORE/src/dia/NeuralCreator.h File Reference

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



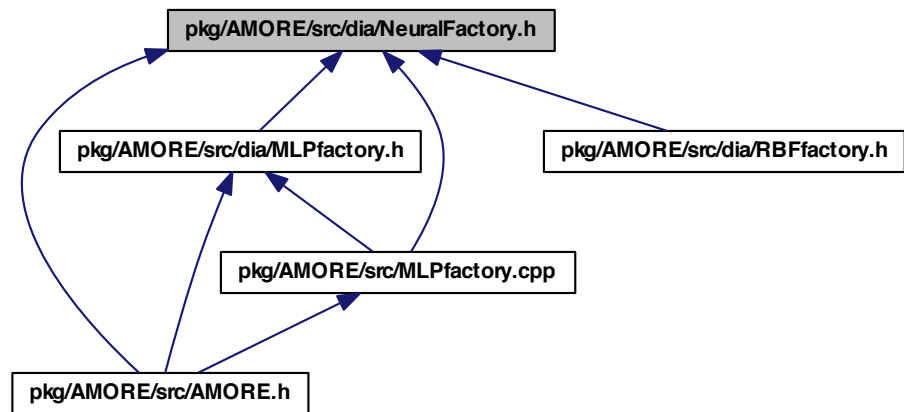
### Classes

- class [NeuralCreator](#)

*class [NeuralCreator](#) -*

## 6.22 pkg/AMORE/src/dia/NeuralFactory.h File Reference

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



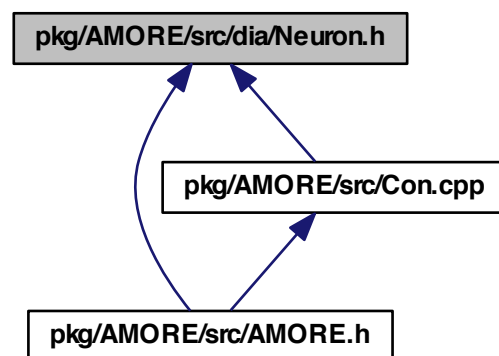
### Classes

- class [NeuralFactory](#)

*class [NeuralFactory](#) -*

## 6.23 pkg/AMORE/src/dia/Neuron.h File Reference

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



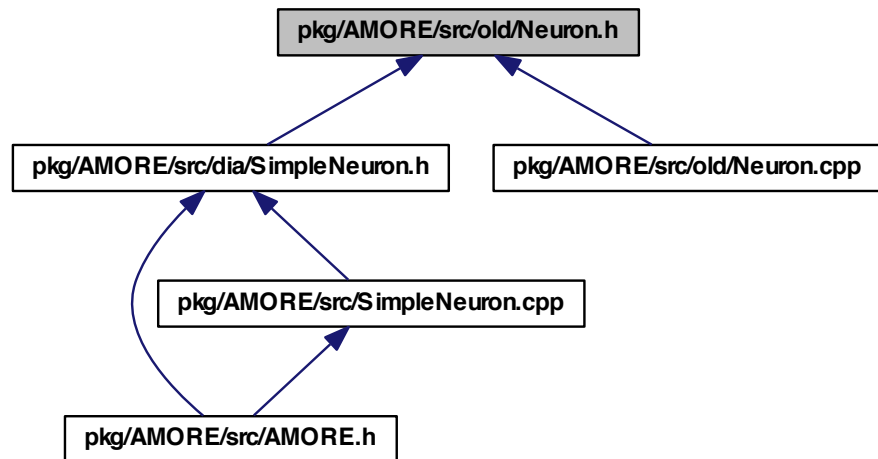
### Classes

- class [Neuron](#)

*class [Neuron](#) -*

## 6.24 pkg/AMORE/src/old/Neuron.h File Reference

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



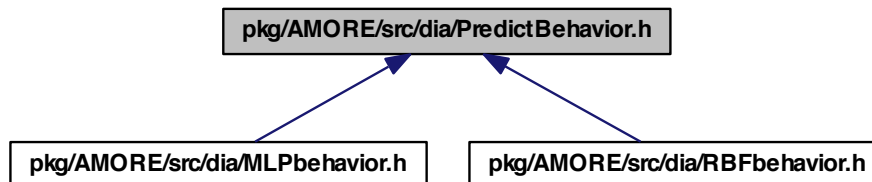
### Classes

- class `Neuron`

*class `Neuron` -*

## 6.25 pkg/AMORE/src/dia/PredictBehavior.h File Reference

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



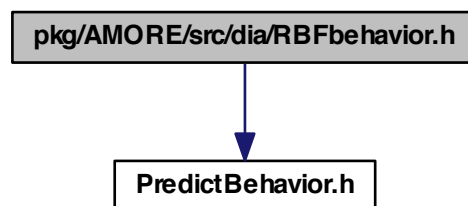
### Classes

- class [PredictBehavior](#)  
*class [PredictBehavior](#) -*

## 6.26 pkg/AMORE/src/dia/RBFbehavior.h File Reference

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

Include dependency graph for RBFbehavior.h:



### Classes

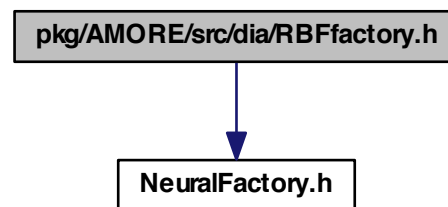
- class [RBFbehavior](#)

class [RBFbehavior](#) -

## 6.27 pkg/AMORE/src/dia/RBFfactory.h File Reference

```
#include "NeuralFactory.h"
```

Include dependency graph for RBFfactory.h:



### Classes

- class [RBFfactory](#)

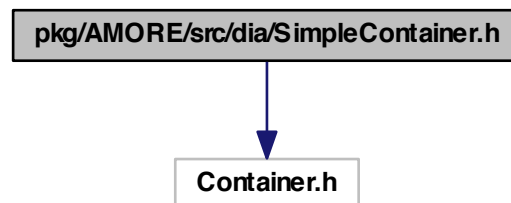
class [RBFfactory](#) -

## 6.28 pkg/AMORE/src/dia/SimpleContainer.h File Reference

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



Include dependency graph for SimpleContainer.h:



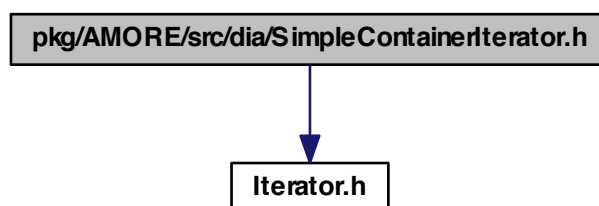
## Classes

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

## 6.29 pkg/AMORE/src/dia/SimpleContainerIterator.h File Reference

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

Include dependency graph for SimpleContainerIterator.h:



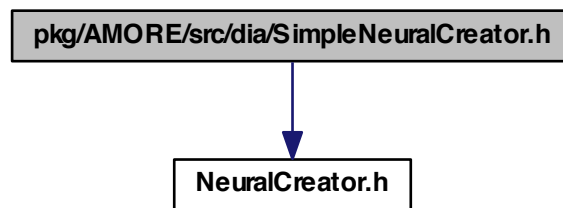
## Classes

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

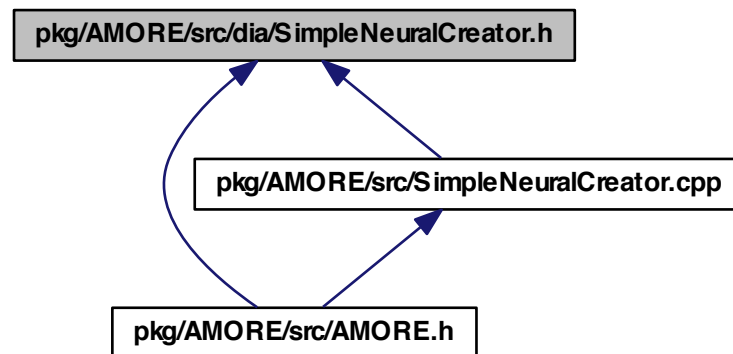
### 6.30 pkg/AMORE/src/dia/SimpleNeuralCreator.h File Reference

```
#include "NeuralCreator.h"
```

Include dependency graph for SimpleNeuralCreator.h:



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



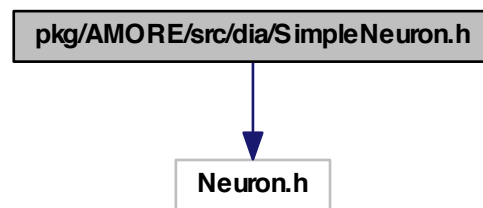
#### Classes

- class [SimpleNeuralCreator](#)  
*class [SimpleNeuralCreator](#) -*

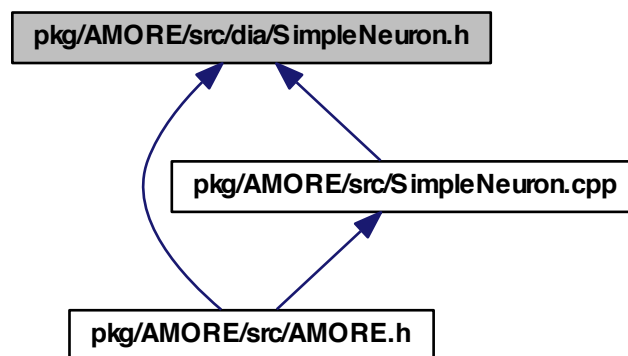
## 6.31 pkg/AMORE/src/dia/SimpleNeuron.h File Reference

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

Include dependency graph for SimpleNeuron.h:



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

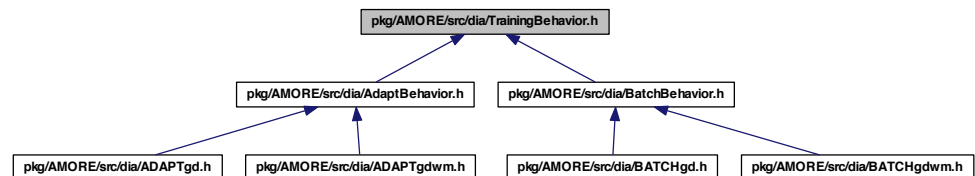


## Classes

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

### 6.32 pkg/AMORE/src/dia/TrainingBehavior.h File Reference

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



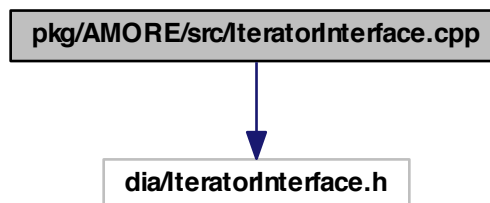
#### Classes

- class `TrainingBehavior`  
*class `TrainingBehavior` -*

### 6.33 pkg/AMORE/src/IteratorInterface.cpp File Reference

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

Include dependency graph for `IteratorInterface.cpp`:

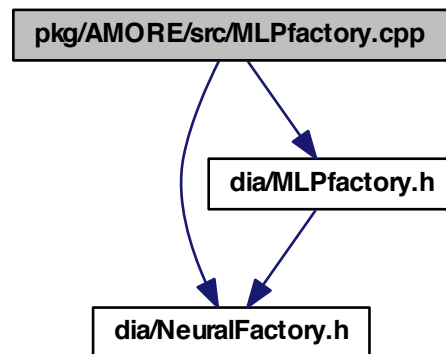


### 6.34 pkg/AMORE/src/MLPfactory.cpp File Reference

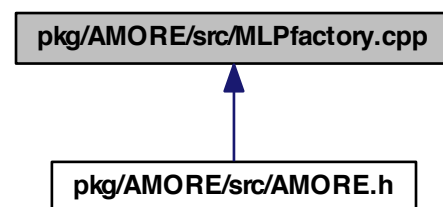
```
#include "dia/NeuralFactory.h"
```

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

Include dependency graph for MLPfactory.cpp:



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



## 6.35 pkg/AMORE/src/old/ConContainer.cpp File Reference

### Classes

- struct [CompareId](#)

## 6.36 pkg/AMORE/src/old/ConContainer.h File Reference

## Classes

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

## 6.37 pkg/AMORE/src/old/MLPlayer.h File Reference

## Classes

- class [MLPlayer](#)

## 6.38 pkg/AMORE/src/old/MLPlayerContainer.h File Reference

## Classes

- class [MLPlayerContainer](#)

## 6.39 pkg/AMORE/src/old/MLPneuralNet.h File Reference

## Classes

- class [MLPneuralNet](#)

## 6.40 pkg/AMORE/src/old/MLPneuralNetFactory.cpp File Reference

## Functions

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

### 6.40.1 Function Documentation

#### 6.40.1.1 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();
}
}
```

## 6.41 pkg/AMORE/src/old/MLPneuron.h File Reference

### Classes

- class [MLPneuron](#)

## 6.42 pkg/AMORE/src/old/MLPneuronContainer.h File Reference

### Classes

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

## 6.43 pkg/AMORE/src/old/NeuralNet.h File Reference

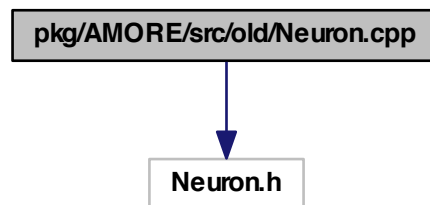
### Classes

- class [NeuralNet](#)

## 6.44 pkg/AMORE/src/old/Neuron.cpp File Reference

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

Include dependency graph for Neuron.cpp:



#### 6.45 pkg/AMORE/src/old/NeuronContainer.cpp File Reference

#### 6.46 pkg/AMORE/src/old/NeuronContainer.h File Reference

##### Classes

- class [NeuronContainer](#)

*A vector of neurons.*

#### 6.47 pkg/AMORE/src/old/RBFneuralNet.h File Reference

##### Classes

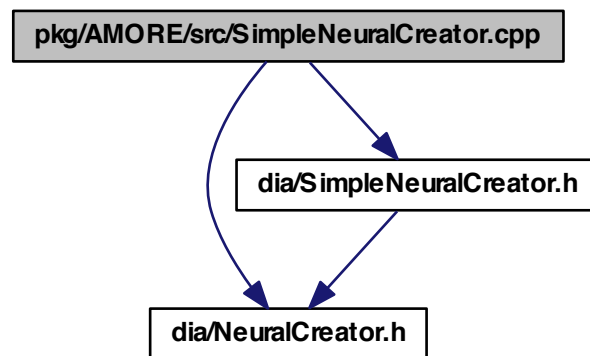
- class [RBFneuralNet](#)

#### 6.48 pkg/AMORE/src/SimpleNeuralCreator.cpp File Reference

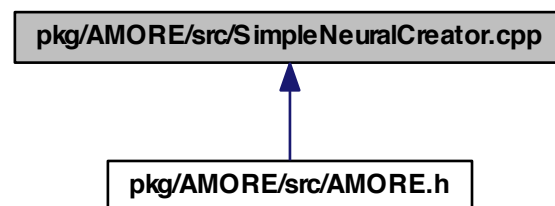
```
#include "dia/NeuralCreator.h"
#include "dia/SimpleNeuralCreator.h"
```



Include dependency graph for SimpleNeuralCreator.cpp:



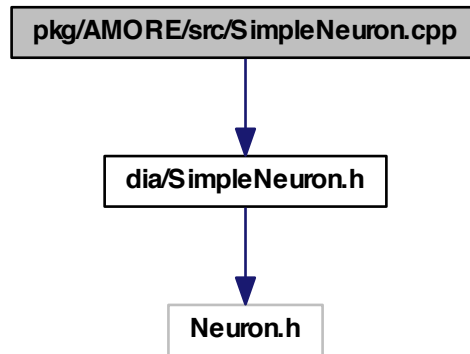
This graph shows which files directly or indirectly include this file:



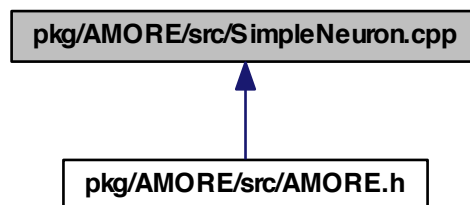
## 6.49 pkg/AMORE/src/SimpleNeuron.cpp File Reference

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

Include dependency graph for SimpleNeuron.cpp:



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



# Index

- ~Con
  - Con, [29](#)
- ~Container
  - Container, [59](#)
- ~Iterator
  - Iterator, [72](#)
- ~Neuron
  - Neuron, [100](#)
- ~NeuronContainer
  - NeuronContainer, [107](#)
- ~SimpleContainer
  - SimpleContainer, [121](#)
- ~SimpleContainerIterator
  - SimpleContainerIterator, [127](#)
- 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](#)
  - TrainingBehavior, [139](#)
- AMORE.h
  - ConContainer, [142](#)
  - ConIteratorPtr, [142](#)
  - ConPtr, [143](#)
  - foreach, [142](#)
  - Handler, [143](#)
  - NeuralCreatorPtr, [143](#)
  - NeuralFactoryPtr, [143](#)
  - NeuronContainer, [143](#)
  - NeuronIteratorPtr, [143](#)
  - NeuronPtr, [143](#)
  - NeuronRef, [143](#)
  - PredictBehaviorRef, [143](#)
  - size\_type, [142](#)
  - TrainingBehaviorRef, [143](#)
- append
  - Container, [59](#)
- BatchBehavior, [17](#)
  - adjustParameters, [19](#)
- BATCHgd, [20](#)
  - adjustParameters, [21](#)
  - outputDerivative, [22](#)
- BATCHgdwm, [22](#)
  - adjustParameters, [24](#)
  - outputDerivative, [25](#)
- begin
  - Container, [61](#)
- bias
  - MLPneuron, [89](#)
- buildAndAppend
  - MLPneuronContainer, [92](#)
- clear
  - Container, [61](#), [62](#)
  - SimpleContainer, [121](#)
- collection
  - Container, [70](#)
- CompareId, [25](#)
  - operator(), [25](#), [26](#)
- Con, [26](#)
  - ~Con, [29](#)
  - Con, [28](#)
  - d\_neuron, [36](#)
  - d\_weight, [37](#)
  - from, [37](#)
  - getFrom, [29](#)
  - getId, [30](#)
  - getNeuron, [30](#)
  - getWeight, [31](#), [32](#)
  - Id, [32](#)

- setFrom, 33
  - setNeuron, 34
  - setWeight, 34, 35
  - show, 35
  - validate, 36
  - weight, 37
- con
  - Neuron, 103
- ConContainer, 37
  - AMORE.h, 142
  - 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
- ConIteratorPtr
  - AMORE.h, 142
- ConPtr
  - AMORE.h, 143
- const\_iterator
  - ConContainer, 40
  - Container, 58
  - NeuronContainer, 106
- const\_reference
  - ConContainer, 40
  - Container, 58
  - NeuronContainer, 106
- 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
- createCon
  - NeuralCreator, 93
  - SimpleNeuralCreator, 131
- createIterator
  - Container, 62
  - SimpleContainer, 122
- CreateMLPNeuralNet
  - MLPNeuralNetFactory.cpp, 168
- createNeuron
  - NeuralCreator, 93
  - SimpleNeuralCreator, 131
- currentItem
  - Iterator, 72
  - SimpleContainerIterator, 127
- d\_accumulator
  - MLPbehavior, 75
  - RBFbehavior, 114
- d\_altitude
  - RBFbehavior, 114
- d\_bias
  - MLPbehavior, 75
- d\_collection
  - SimpleContainer, 124
- d\_container
  - SimpleContainerIterator, 128
- d\_id
  - SimpleNeuron, 138
- d\_iterator
  - SimpleContainerIterator, 128
- d\_nCons
  - MLPbehavior, 75
  - RBFbehavior, 114
- d\_neuron
  - Con, 36
- d\_output
  - MLPbehavior, 75
  - RBFbehavior, 114
- d\_weight
  - Con, 37
- d\_width
  - RBFbehavior, 114
- empty

- Container, 62
- SimpleContainer, 122
- end
  - Container, 63
- erase
  - ConContainer, 41
- first
  - Iterator, 72
  - SimpleContainerIterator, 127
- foreach
  - AMORE.h, 142
- from
  - Con, 37
- getConId
  - Neuron, 100
  - NeuronContainer, 107
- getFrom
  - Con, 29
  - NeuronContainer, 108
- getId
  - Con, 30
  - ConContainer, 43
  - MLPNeuronContainer, 92
  - Neuron, 100
  - NeuronContainer, 108
  - SimpleNeuron, 135
- getNeuron
  - Con, 30
- getWeight
  - Con, 31, 32
  - Neuron, 100
  - NeuronContainer, 108
- Handler
  - AMORE.h, 143
- Id
  - Con, 32
  - Neuron, 103
- isDone
  - Iterator, 72
  - SimpleContainerIterator, 127
- Iterator, 70
  - ~Iterator, 72
  - currentItem, 72
  - first, 72
  - isDone, 72
  - Iterator, 72
- next, 72
- iterator
  - ConContainer, 41
  - Container, 58
  - NeuronContainer, 106
- load
  - Container, 63
- makeCon
  - MLPfactory, 78
  - NeuralFactory, 95
  - RBFfactory, 116
- makeNeuron
  - MLPfactory, 78
  - NeuralFactory, 95
  - RBFfactory, 116
- MLPbehavior, 72
  - d\_accumulator, 75
  - d\_bias, 75
  - d\_nCons, 75
  - d\_output, 75
  - predict, 75
- MLPfactory, 76
  - makeCon, 78
  - makeNeuron, 78
  - MLPfactory, 77
- MLPlayer, 78
- MLPlayerContainer, 81
- MLPneuralNet, 84
  - nLayers, 86
- MLPneuralNetFactory.cpp
  - CreateMLPneuralNet, 168
- MLPneuron, 86
  - bias, 89
- MLPneuronContainer, 89
  - buildAndAppend, 92
  - getId, 92
- NeuralCreator, 92
  - createCon, 93
  - createNeuron, 93
- NeuralCreatorPtr
  - AMORE.h, 143
- NeuralFactory, 94
  - makeCon, 95
  - makeNeuron, 95
- NeuralFactoryPtr
  - AMORE.h, 143
- NeuralNet, 96

- train, 96
- Neuron, 97
  - ~Neuron, 100
  - con, 103
  - getConId, 100
  - getId, 100
  - getWeight, 100
  - Id, 103
  - Neuron, 99, 100
  - numOfCons, 101
  - outputValue, 103
  - setFrom, 101
  - setId, 101
  - setWeight, 101
  - show, 101, 102
  - validate, 102
- NeuronContainer, 103
  - ~NeuronContainer, 107
  - AMORE.h, 143
  - const\_iterator, 106
  - const\_reference, 106
  - getConId, 107
  - getFrom, 108
  - getId, 108
  - getWeight, 108
  - iterator, 106
  - NeuronContainer, 107
  - numOfCons, 108
  - numOfNeurons, 108
  - setFrom, 109
  - setId, 109
  - setWeight, 109
  - value\_type, 107
- NeuronIteratorPtr
  - AMORE.h, 143
- NeuronPtr
  - AMORE.h, 143
- NeuronRef
  - AMORE.h, 143
- next
  - Iterator, 72
  - SimpleContainerIterator, 127
- nLayers
  - MLPneuralNet, 86
- numOfCons
  - ConContainer, 45
  - Neuron, 101
  - NeuronContainer, 108
- numOfNeurons
  - NeuronContainer, 108
- operator()
  - CompareId, 25, 26
- outputDerivative
  - ADAPTgd, 14
  - ADAPTgdwm, 17
  - BATCHgd, 22
  - BATCHgdwm, 25
- outputValue
  - Neuron, 103
- pkg/AMORE/src/AMORE.h, 141
- pkg/AMORE/src/Con.cpp, 144
- pkg/AMORE/src/Container.cpp, 145
- pkg/AMORE/src/containerInterface.cpp, 145
- pkg/AMORE/src/ContainerIterator.cpp, 146
- pkg/AMORE/src/dia/AdaptBehavior.h, 146
- pkg/AMORE/src/dia/ADAPTgd.h, 147
- pkg/AMORE/src/dia/ADAPTgdwm.h, 148
- pkg/AMORE/src/dia/BatchBehavior.h, 149
- pkg/AMORE/src/dia/BATCHgd.h, 150
- pkg/AMORE/src/dia/BATCHgdwm.h, 151
- pkg/AMORE/src/dia/Con.h, 152
- pkg/AMORE/src/dia/Container.h, 153
- pkg/AMORE/src/dia/Iterator.h, 154
- pkg/AMORE/src/dia/MLPbehavior.h, 155
- pkg/AMORE/src/dia/MLPfactory.h, 155
- pkg/AMORE/src/dia/NeuralCreator.h, 157
- pkg/AMORE/src/dia/NeuralFactory.h, 158
- pkg/AMORE/src/dia/Neuron.h, 159
- pkg/AMORE/src/dia/PredictBehavior.h, 161
- pkg/AMORE/src/dia/RBFbehavior.h, 161
- pkg/AMORE/src/dia/RBFfactory.h, 162
- pkg/AMORE/src/dia/SimpleContainer.h, 162
- pkg/AMORE/src/dia/SimpleContainerIterator.h, 163
- pkg/AMORE/src/dia/SimpleNeuralCreator.h, 164
- pkg/AMORE/src/dia/SimpleNeuron.h, 165
- pkg/AMORE/src/dia/TrainingBehavior.h, 166
- pkg/AMORE/src/IteratorInterface.cpp, 166
- pkg/AMORE/src/MLPfactory.cpp, 166
- pkg/AMORE/src/old/Con.cpp, 144
- pkg/AMORE/src/old/Con.h, 152
- pkg/AMORE/src/old/ConContainer.cpp, 167
- pkg/AMORE/src/old/ConContainer.h, 167
- pkg/AMORE/src/old/Container.cpp, 145
- pkg/AMORE/src/old/Container.h, 154
- pkg/AMORE/src/old/MLPlayer.h, 168
- pkg/AMORE/src/old/MLPlayerContainer.h, 168

- pkg/AMORE/src/old/MLPneuralNet.h, 168
- pkg/AMORE/src/old/MLPneuralNetFactory.cpp, 168
- pkg/AMORE/src/old/MLPneuron.h, 169
- pkg/AMORE/src/old/MLPneuronContainer.h, 169
- pkg/AMORE/src/old/NeuralNet.h, 169
- pkg/AMORE/src/old/Neuron.cpp, 169
- pkg/AMORE/src/old/Neuron.h, 160
- pkg/AMORE/src/old/NeuronContainer.cpp, 170
- pkg/AMORE/src/old/NeuronContainer.h, 170
- pkg/AMORE/src/old/RBFneuralNet.h, 170
- pkg/AMORE/src/SimpleNeuralCreator.cpp, 170
- pkg/AMORE/src/SimpleNeuron.cpp, 171
- predict
  - MLPbehavior, 75
  - PredictBehavior, 111
  - RBFbehavior, 114
- PredictBehavior, 110
  - predict, 111
- PredictBehaviorRef
  - AMORE.h, 143
- push\_back
  - Container, 64, 65
  - SimpleContainer, 122
- RBFbehavior, 111
  - d\_accumulator, 114
  - d\_altitude, 114
  - d\_nCons, 114
  - d\_output, 114
  - d\_width, 114
  - predict, 114
- RBFfactory, 115
  - makeCon, 116
  - makeNeuron, 116
  - RBFfactory, 116
- RBFneuralNet, 117
- reserve
  - Container, 66
  - SimpleContainer, 123
- resize
  - Container, 67
- select
  - ConContainer, 46
- setFrom
  - Con, 33
- ConContainer, 48
- Neuron, 101
- NeuronContainer, 109
- setId
  - Neuron, 101
  - NeuronContainer, 109
  - SimpleNeuron, 136
- setNeuron
  - Con, 34
- setWeight
  - Con, 34, 35
  - ConContainer, 50, 51
  - Neuron, 101
  - NeuronContainer, 109
- show
  - Con, 35
  - Container, 67
  - Neuron, 101, 102
  - SimpleContainer, 123
  - SimpleNeuron, 136
- SimpleContainer, 118
  - ~SimpleContainer, 121
  - clear, 121
  - createIterator, 122
  - d\_collection, 124
  - empty, 122
  - push\_back, 122
  - reserve, 123
  - show, 123
  - SimpleContainer, 121
  - SimpleContainerIterator< T >, 124
  - size, 124
  - validate, 124
- SimpleContainer< T >
  - SimpleContainerIterator, 128
- SimpleContainerIterator, 125
  - ~SimpleContainerIterator, 127
  - currentItem, 127
  - d\_container, 128
  - d\_iterator, 128
  - first, 127
  - isDone, 127
  - next, 127
  - SimpleContainer< T >, 128
  - SimpleContainerIterator, 127
- SimpleContainerIterator< T >
  - SimpleContainer, 124
- SimpleNeuralCreator, 128
  - createCon, 131
  - createNeuron, 131

- SimpleNeuralCreator, [130](#)
- SimpleNeuron, [132](#)
  - d\_Id, [138](#)
  - getId, [135](#)
  - setId, [136](#)
  - show, [136](#)
  - SimpleNeuron, [135](#)
  - validate, [137](#)
- size
  - Container, [68](#)
  - SimpleContainer, [124](#)
- size\_type
  - AMORE.h, [142](#)
- store
  - Container, [69](#)
- train
  - NeuralNet, [96](#)
- TrainingBehavior, [138](#)
  - adjustParameters, [139](#)
- TrainingBehaviorRef
  - AMORE.h, [143](#)
- validate
  - Con, [36](#)
  - ConContainer, [53](#)
  - Container, [69](#)
  - Neuron, [102](#)
  - SimpleContainer, [124](#)
  - SimpleNeuron, [137](#)
- value\_type
  - ConContainer, [41](#)
  - Container, [58](#)
  - NeuronContainer, [107](#)
- weight
  - Con, [37](#)