

# Intelligence Reference Manual

**Revision**  
1.6

Generated by Doxygen 1.3.8

Sun Aug 15 04:30:08 2004



# Contents

<b>1</b>	<b>Intelligence</b>	<b>1</b>
<b>2</b>	<b>Intelligence Hierarchical Index</b>	<b>3</b>
2.1	Intelligence Class Hierarchy . . . . .	3
<b>3</b>	<b>Intelligence Class Index</b>	<b>7</b>
3.1	Intelligence Class List . . . . .	7
<b>4</b>	<b>Intelligence File Index</b>	<b>9</b>
4.1	Intelligence File List . . . . .	9
<b>5</b>	<b>Intelligence Page Index</b>	<b>13</b>
5.1	Intelligence Related Pages . . . . .	13
<b>6</b>	<b>Intelligence Class Documentation</b>	<b>15</b>
6.1	AbstractStringFeature Class Reference . . . . .	15
6.2	AbstractStringListFeature Class Reference . . . . .	28
6.3	ActiveLezi Class Reference . . . . .	41
6.4	ActiveWindowFeature Class Reference . . . . .	48
6.5	ActiveWindowFeatureProvider Class Reference . . . . .	59
6.6	AudioBandFeature Class Reference . . . . .	64
6.7	AudioFeatureProvider Class Reference . . . . .	73
6.8	AudioFeatureProvider::SampleData Struct Reference . . . . .	78
6.9	AudioMeanFeature Class Reference . . . . .	80
6.10	AudioPeakFeature Class Reference . . . . .	89
6.11	AveragePredictor Class Reference . . . . .	98
6.12	BluetoothFeatureProvider Class Reference . . . . .	102
6.13	BluetoothLinuxFeatureProvider Class Reference . . . . .	108
6.14	BluetoothLinuxFeatureProvider::L2Connection Class Reference . . . . .	115
6.15	BluetoothNumPeersFeature Class Reference . . . . .	119

6.16 BluetoothPeersFeature Class Reference . . . . .	128
6.17 Classifier Class Reference . . . . .	140
6.18 ClassifierAlgorithm Class Reference . . . . .	145
6.19 config_param Struct Reference . . . . .	148
6.20 ConfigReader Class Reference . . . . .	149
6.21 Feature Class Reference . . . . .	153
6.22 FeatureContainer Class Reference . . . . .	159
6.23 FeatureProvider Class Reference . . . . .	167
6.24 GNG Class Reference . . . . .	171
6.25 GSMCellFeature Class Reference . . . . .	181
6.26 GSMFeatureProvider Class Reference . . . . .	192
6.27 HMM Class Reference . . . . .	199
6.28 std::list< element > Class Template Reference . . . . .	203
6.29 LoggingFeatureContainer Class Reference . . . . .	204
6.30 std::map< key, value > Class Template Reference . . . . .	209
6.31 meta_cluster Struct Reference . . . . .	210
6.32 NetworkFeatureProvider Class Reference . . . . .	212
6.33 next_context_description Struct Reference . . . . .	219
6.34 node< comparable > Struct Template Reference . . . . .	220
6.35 NumericalContinuousFeature Class Reference . . . . .	222
6.36 NumericalDiscreteFeature Class Reference . . . . .	231
6.37 std::pair< first, second > Class Template Reference . . . . .	240
6.38 PersistentFeature Class Reference . . . . .	241
6.39 PersistentReader Class Reference . . . . .	249
6.40 PersistentWriter Class Reference . . . . .	251
6.41 PHPWrapperFeatureProvider Class Reference . . . . .	252
6.42 PowerFeature Class Reference . . . . .	256
6.43 PowerFeatureProvider Class Reference . . . . .	263
6.44 Predictor Class Reference . . . . .	267
6.45 PredictorAlgorithm Class Reference . . . . .	272
6.46 PredWrapper Class Reference . . . . .	275
6.47 ReplayFeatureContainer Class Reference . . . . .	278
6.48 Scanner Class Reference . . . . .	283
6.49 ServiceHost Class Reference . . . . .	286
6.50 SingleStepDurationPredictor Class Reference . . . . .	288
6.51 SingleStepDurationPredictor::contextinfo Struct Reference . . . . .	296

6.52	<a href="#">splaytree&lt; comparable &gt; Class Template Reference</a>	297
6.53	<a href="#">splaytree_iterator&lt; comparable &gt; Class Template Reference</a>	304
6.54	<a href="#">Statistics Class Reference</a>	307
6.55	<a href="#">SystemCommandStringListFeature Class Reference</a>	313
6.56	<a href="#">SystemCommandStringListFeatureProvider Class Reference</a>	325
6.57	<a href="#">Thread Class Reference</a>	332
6.58	<a href="#">Thread::Guard Class Reference</a>	335
6.59	<a href="#">Thread::Lock Class Reference</a>	337
6.60	<a href="#">Thread::ThreadHandle Class Reference</a>	338
6.61	<a href="#">TimeFeature Class Reference</a>	339
6.62	<a href="#">TimeFeatureProvider Class Reference</a>	348
6.63	<a href="#">Trie&lt; _type &gt; Class Template Reference</a>	353
6.64	<a href="#">Trie&lt; _type &gt;::Node Class Reference</a>	358
6.65	<a href="#">Unit Class Reference</a>	361
6.66	<a href="#">unittree&lt; comparator &gt; Class Template Reference</a>	372
6.67	<a href="#">std::vector&lt; element &gt; Class Template Reference</a>	375
6.68	<a href="#">VideoFeatureProvider Class Reference</a>	376
6.69	<a href="#">VideoFeatureProvider::SampleData Struct Reference</a>	383
6.70	<a href="#">WlanAccessPointFeatureProvider Class Reference</a>	384
6.71	<a href="#">WlanActiveEssidFeature Class Reference</a>	391
6.72	<a href="#">WlanActiveMacAddressFeature Class Reference</a>	402
6.73	<a href="#">WlanActiveModeFeature Class Reference</a>	413
6.74	<a href="#">WlanActiveSignalLevelFeature Class Reference</a>	420
6.75	<a href="#">WlanFeatureProvider Class Reference</a>	429
6.76	<a href="#">WlanLinuxFeatureProvider Class Reference</a>	437
6.77	<a href="#">WlanNumPeersFeature Class Reference</a>	445
6.78	<a href="#">WlanPeerInfo Struct Reference</a>	454
6.79	<a href="#">WlanPeersFeature Class Reference</a>	456
<b>7</b>	<b>Intelligence File Documentation</b>	<b>469</b>
7.1	<a href="#">AbstractString.cpp File Reference</a>	469
7.2	<a href="#">AbstractString.h File Reference</a>	471
7.3	<a href="#">ActiveLezi.cpp File Reference</a>	473
7.4	<a href="#">ActiveLezi.h File Reference</a>	475
7.5	<a href="#">ActiveWindow.cpp File Reference</a>	477
7.6	<a href="#">ActiveWindow.h File Reference</a>	479
7.7	<a href="#">ActiveWindowLinux.cpp File Reference</a>	481

7.8	ActiveWindowWindows.cpp File Reference	482
7.9	Audio.cpp File Reference	484
7.10	Audio.h File Reference	486
7.11	AveragePredictor.cpp File Reference	488
7.12	AveragePredictor.h File Reference	490
7.13	Bluetooth.cpp File Reference	492
7.14	Bluetooth.h File Reference	493
7.15	BluetoothLinux.cpp File Reference	495
7.16	BluetoothLinux.h File Reference	497
7.17	BluetoothWindows.cpp File Reference	499
7.18	Classifier.h File Reference	502
7.19	ConfigReader.cpp File Reference	505
7.20	ConfigReader.h File Reference	506
7.21	debug.cpp File Reference	508
7.22	debug.h File Reference	511
7.23	defines.h File Reference	514
7.24	dll.cpp File Reference	517
7.25	doxygen.h File Reference	518
7.26	Feature.h File Reference	519
7.27	FeatureContainer.cpp File Reference	523
7.28	FeatureContainer.h File Reference	525
7.29	featurevector.cpp File Reference	527
7.30	featurevector.h File Reference	529
7.31	GNG.cpp File Reference	531
7.32	GNG.h File Reference	533
7.33	GSM.cpp File Reference	535
7.34	GSM.h File Reference	537
7.35	GSMLinux.cpp File Reference	539
7.36	GSMWindows.cpp File Reference	541
7.37	HMM.cpp File Reference	543
7.38	HMM.h File Reference	545
7.39	JNI_Bluetooth.cpp File Reference	547
7.40	JNI_Context.cpp File Reference	550
7.41	JNI_Samples.cpp File Reference	553
7.42	LinuxClassifier.cpp File Reference	558
7.43	LinuxFeatureContainer.cpp File Reference	559

7.44	LoggingFeatureContainer.cpp File Reference	560
7.45	LoggingFeatureContainer.h File Reference	562
7.46	main-classification.cpp File Reference	564
7.47	main-features.cpp File Reference	567
7.48	main-prediction.cpp File Reference	569
7.49	main.cpp File Reference	571
7.50	Network.cpp File Reference	572
7.51	Network.h File Reference	574
7.52	Numerical.cpp File Reference	576
7.53	Numerical.h File Reference	577
7.54	PaWrapper.h File Reference	579
7.55	PersistentReader.cpp File Reference	581
7.56	PersistentReader.h File Reference	582
7.57	PersistentWriter.cpp File Reference	584
7.58	PersistentWriter.h File Reference	585
7.59	PHPCommon.h File Reference	587
7.60	PHPWrapper.c File Reference	588
7.61	PHPWrapper.cpp File Reference	592
7.62	PHPWrapper.h File Reference	595
7.63	Power.cpp File Reference	597
7.64	Power.h File Reference	599
7.65	PowerLinux.cpp File Reference	601
7.66	PowerWindows.cpp File Reference	602
7.67	Predictor.h File Reference	603
7.68	ReplayFeatureContainer.cpp File Reference	606
7.69	ReplayFeatureContainer.h File Reference	608
7.70	servicehost.cpp File Reference	610
7.71	servicehost.h File Reference	611
7.72	SingleStepDuration.cpp File Reference	612
7.73	SingleStepDuration.h File Reference	614
7.74	splaytree.h File Reference	616
7.75	Statistics.cpp File Reference	617
7.76	Statistics.h File Reference	618
7.77	symbianstr.c File Reference	620
7.78	symbianstr.h File Reference	621
7.79	SystemCommandStringList.cpp File Reference	622

7.80	SystemCommandStringList.h File Reference . . . . .	624
7.81	Thread.cpp File Reference . . . . .	626
7.82	Thread.h File Reference . . . . .	627
7.83	ThreadPosix.cpp File Reference . . . . .	629
7.84	ThreadSymbian.cpp File Reference . . . . .	630
7.85	ThreadWindows.cpp File Reference . . . . .	632
7.86	Time.cpp File Reference . . . . .	633
7.87	TimeFeature.h File Reference . . . . .	635
7.88	trie.h File Reference . . . . .	637
7.89	Unit.cpp File Reference . . . . .	639
7.90	Unit.h File Reference . . . . .	640
7.91	version.h File Reference . . . . .	642
7.92	Video.cpp File Reference . . . . .	643
7.93	Video.h File Reference . . . . .	645
7.94	WindowsClassifier.cpp File Reference . . . . .	647
7.95	WindowsFeatureContainer.cpp File Reference . . . . .	648
7.96	WindowsPredictor.cpp File Reference . . . . .	649
7.97	Wlan.cpp File Reference . . . . .	650
7.98	Wlan.h File Reference . . . . .	651
7.99	WlanAP.cpp File Reference . . . . .	653
7.100	WlanAP.h File Reference . . . . .	655
7.101	WlanLinux.cpp File Reference . . . . .	657
7.102	WlanLinux.h File Reference . . . . .	659
7.103	WlanSymbolAPI.cpp File Reference . . . . .	661
7.104	WlanWindows.cpp File Reference . . . . .	663
<b>8</b>	<b>Intelligence Page Documentation</b>	<b>665</b>
8.1	Introduction . . . . .	665
8.2	Installation . . . . .	666
8.3	Framework . . . . .	667
8.4	Todo List . . . . .	668



# Chapter 1

## Intelligence

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

©2003-2004 by Rene Mayrhofer, Harald Radi



## Chapter 2

# Intelligence Hierarchical Index

### 2.1 Intelligence Class Hierarchy

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

AudioFeatureProvider::SampleData . . . . .	78
CIntelligence	
CIntelligenceFactory	
ClassifierAlgorithm . . . . .	145
Classifier . . . . .	140
GNG . . . . .	171
config	
config_param . . . . .	148
ConfigReader . . . . .	149
context	
ContextApplicationsBinding	
coord	
description	
draw_char	
event_handlers	
feature	
Feature . . . . .	153
PersistantFeature . . . . .	241
AbstractStringFeature . . . . .	15
AbstractStringListFeature . . . . .	28
BluetoothPeersFeature . . . . .	128
SystemCommandStringListFeature . . . . .	313
WlanPeersFeature . . . . .	456
ActiveWindowFeature . . . . .	48
GSMCellFeature . . . . .	181
WlanActiveEssidFeature . . . . .	391
WlanActiveMacAddressFeature . . . . .	402
NumericalContinuousFeature . . . . .	222
AudioBandFeature . . . . .	64
AudioMeanFeature . . . . .	80
VideoMotionFeature	
WlanActiveSignalLevelFeature . . . . .	420

NumericalDiscreteFeature . . . . .	231
AudioPeakFeature . . . . .	89
BluetoothNumPeersFeature . . . . .	119
WlanNumPeersFeature . . . . .	445
TimeFeature . . . . .	339
PowerFeature . . . . .	256
WlanActiveModeFeature . . . . .	413
FeatureContainer . . . . .	159
LoggingFeatureContainer . . . . .	204
ReplayFeatureContainer . . . . .	278
FeatureProvider . . . . .	167
ActiveWindowFeatureProvider . . . . .	59
AudioFeatureProvider . . . . .	73
BluetoothFeatureProvider . . . . .	102
BluetoothLinuxFeatureProvider . . . . .	108
GSMFeatureProvider . . . . .	192
PHPWrapperFeatureProvider . . . . .	252
PowerFeatureProvider . . . . .	263
SystemCommandStringListFeatureProvider . . . . .	325
NetworkFeatureProvider . . . . .	212
WlanAccessPointFeatureProvider . . . . .	384
TimeFeatureProvider . . . . .	348
VideoFeatureProvider . . . . .	376
WlanFeatureProvider . . . . .	429
WlanLinuxFeatureProvider . . . . .	437
ffmpeg	
histo	
images	
std::list< element > . . . . .	203
std::map< key, value > . . . . .	209
meta_cluster . . . . .	210
next_context_description . . . . .	219
node< comparable > . . . . .	220
node< comparator > . . . . .	220
ns1__getActiveId	
ns1__getActiveIdResponse	
std::pair< first, second > . . . . .	240
PersistentReader . . . . .	249
PersistentWriter . . . . .	251
predict	
predict_set	
PredictorAlgorithm . . . . .	272
ActiveLezi . . . . .	41
AveragePredictor . . . . .	98
HMM . . . . .	199
Predictor . . . . .	267
SingleStepDurationPredictor . . . . .	288
PredWrapper . . . . .	275
Preferences	
PreferencesImpl	
region	
segment	

ServiceHost . . . . .	286
SingleStepDurationPredictor::contextinfo . . . . .	296
SOAP_ENV__Code	
SOAP_ENV__Detail	
SOAP_ENV__Fault	
SOAP_ENV__Header	
splaytree< comparable > . . . . .	297
splaytree< comparator > . . . . .	297
unittree< comparator > . . . . .	372
splaytree_iterator< comparable > . . . . .	304
Statistics . . . . .	307
Thread . . . . .	332
BluetoothLinuxFeatureProvider . . . . .	108
BluetoothLinuxFeatureProvider::L2Connection . . . . .	115
GSMFeatureProvider . . . . .	192
Scanner . . . . .	283
SystemCommandStringListFeatureProvider . . . . .	325
VideoFeatureProvider . . . . .	376
WlanLinuxFeatureProvider . . . . .	437
Thread::Guard . . . . .	335
Thread::Lock . . . . .	337
Thread::ThreadHandle . . . . .	338
trackoptions	
Trie< _type > . . . . .	353
Trie< _type >::Node . . . . .	358
Unit . . . . .	361
std::vector< element > . . . . .	375
video_dev	
VideoFeatureProvider::SampleData . . . . .	383
webcam	
webcam_tmpfile	
WlanPeerInfo . . . . .	454



## Chapter 3

# Intelligence Class Index

### 3.1 Intelligence Class List

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

<a href="#">AbstractStringFeature</a> (Baseclass for features providing string values ) . . . . .	15
<a href="#">AbstractStringListFeature</a> (Baseclass for features providing lists of string values ) . . . . .	28
<a href="#">ActiveLezi</a> . . . . .	41
<a href="#">ActiveWindowFeature</a> (Returns the application with focus ) . . . . .	48
<a href="#">ActiveWindowFeatureProvider</a> (Provider for the <a href="#">ActiveWindowFeature</a> ) . . . . .	59
<a href="#">AudioBandFeature</a> (Represents a subband ) . . . . .	64
<a href="#">AudioFeatureProvider</a> (Provider for the audio features ) . . . . .	73
<a href="#">AudioFeatureProvider::SampleData</a> . . . . .	78
<a href="#">AudioMeanFeature</a> (Calculates the mean level of the input signal ) . . . . .	80
<a href="#">AudioPeakFeature</a> (Counts the peaks in the input signal ) . . . . .	89
<a href="#">AveragePredictor</a> . . . . .	98
<a href="#">BluetoothFeatureProvider</a> (Provider for the Bluetooth features ) . . . . .	102
<a href="#">BluetoothLinuxFeatureProvider</a> . . . . .	108
<a href="#">BluetoothLinuxFeatureProvider::L2Connection</a> . . . . .	115
<a href="#">BluetoothNumPeersFeature</a> (Number of peers in range ) . . . . .	119
<a href="#">BluetoothPeersFeature</a> (List of peers in range ) . . . . .	128
<a href="#">Classifier</a> (Classifier Wrapper ) . . . . .	140
<a href="#">ClassifierAlgorithm</a> (ClassifierAlgorithm Interface ) . . . . .	145
<a href="#">config_param</a> (Description for parameters in the config file ) . . . . .	148
<a href="#">ConfigReader</a> (Configuration parser ) . . . . .	149
<a href="#">Feature</a> (Feature interface ) . . . . .	153
<a href="#">FeatureContainer</a> (Container class ) . . . . .	159
<a href="#">FeatureProvider</a> (FeatureProvider interface ) . . . . .	167
<a href="#">GNG</a> (This is lifelong GNG ) . . . . .	171
<a href="#">GSMCellFeature</a> . . . . .	181
<a href="#">GSMFeatureProvider</a> . . . . .	192
<a href="#">HMM</a> . . . . .	199
<a href="#">std::list&lt; element &gt;</a> (STL list template ) . . . . .	203
<a href="#">LoggingFeatureContainer</a> . . . . .	204
<a href="#">std::map&lt; key, value &gt;</a> (STL map template ) . . . . .	209
<a href="#">meta_cluster</a> . . . . .	210
<a href="#">NetworkFeatureProvider</a> . . . . .	212
<a href="#">next_context_description</a> . . . . .	219

<a href="#">node&lt; comparable &gt;</a>	220
<a href="#">NumericalContinuousFeature</a> (Abstract continuous numerical feature )	222
<a href="#">NumericalDiscreteFeature</a> (Abstract discrete numerical feature )	231
<a href="#">std::pair&lt; first, second &gt;</a> (STL pair template )	240
<a href="#">PersistantFeature</a> (PersistantFeature interface )	241
<a href="#">PersistantReader</a>	249
<a href="#">PersistantWriter</a>	251
<a href="#">PHPWrapperFeatureProvider</a>	252
<a href="#">PowerFeature</a>	256
<a href="#">PowerFeatureProvider</a>	263
<a href="#">Predictor</a> (Predictor Wrapper )	267
<a href="#">PredictorAlgorithm</a> (Predictor Interface )	272
<a href="#">PredWrapper</a>	275
<a href="#">ReplayFeatureContainer</a>	278
<a href="#">Scanner</a>	283
<a href="#">ServiceHost</a>	286
<a href="#">SingleStepDurationPredictor</a>	288
<a href="#">SingleStepDurationPredictor::contextinfo</a>	296
<a href="#">splaytree&lt; comparable &gt;</a>	297
<a href="#">splaytree_iterator&lt; comparable &gt;</a>	304
<a href="#">Statistics</a> (This class can be used to compute statistics on a _single_, numerical continuous variable )	307
<a href="#">SystemCommandStringListFeature</a>	313
<a href="#">SystemCommandStringListFeatureProvider</a> (This class returns a feature out of any system command that can return a newline-separated list of strings )	325
<a href="#">Thread</a> (Platform independent thread class )	332
<a href="#">Thread::Guard</a> (Guard )	335
<a href="#">Thread::Lock</a> (Lock )	337
<a href="#">Thread::ThreadHandle</a> (Thread handle )	338
<a href="#">TimeFeature</a>	339
<a href="#">TimeFeatureProvider</a>	348
<a href="#">Trie&lt; _type &gt;</a>	353
<a href="#">Trie&lt; _type &gt;::Node</a>	358
<a href="#">Unit</a>	361
<a href="#">unittestree&lt; comparator &gt;</a>	372
<a href="#">std::vector&lt; element &gt;</a> (STL vector template )	375
<a href="#">VideoFeatureProvider</a> (Provider for the video features )	376
<a href="#">VideoFeatureProvider::SampleData</a>	383
<a href="#">WlanAccessPointFeatureProvider</a>	384
<a href="#">WlanActiveEssidFeature</a> (This feature encapsulates the ESSID which is currently set for a WLAN network interface or the ESSID which it is currently bound to (if not set explicitly) )	391
<a href="#">WlanActiveMacAddressFeature</a> (This feature encapsulates the ESSID which is currently set for a WLAN network interface or the ESSID which it is currently bound to (if not set explicitly) )	402
<a href="#">WlanActiveModeFeature</a>	413
<a href="#">WlanActiveSignalLevelFeature</a>	420
<a href="#">WlanFeatureProvider</a>	429
<a href="#">WlanLinuxFeatureProvider</a>	437
<a href="#">WlanNumPeersFeature</a>	445
<a href="#">WlanPeerInfo</a>	454
<a href="#">WlanPeersFeature</a>	456



## Chapter 4

# Intelligence File Index

### 4.1 Intelligence File List

Here is a list of all documented files with brief descriptions:

<a href="#">AbstractString.cpp</a> (Abstract string feature class implementation ) . . . . .	469
<a href="#">AbstractString.h</a> (Abstract string feature class declaration ) . . . . .	471
<a href="#">ActiveLezi.cpp</a> (Active lezi predictor ) . . . . .	473
<a href="#">ActiveLezi.h</a> (Active lezi predictor ) . . . . .	475
<a href="#">ActiveWindow.cpp</a> (Active window feature ) . . . . .	477
<a href="#">ActiveWindow.h</a> (Active window feature ) . . . . .	479
<a href="#">ActiveWindowLinux.cpp</a> (Active window feature ) . . . . .	481
<a href="#">ActiveWindowWindows.cpp</a> (Active window feature ) . . . . .	482
<a href="#">alg.c</a> . . . . .	??
<a href="#">motion/alg.c</a> . . . . .	??
<a href="#">alg.h</a> . . . . .	??
<a href="#">Audio.cpp</a> (Audio feature ) . . . . .	484
<a href="#">Audio.h</a> (Audio feature ) . . . . .	486
<a href="#">AveragePredictor.cpp</a> (A simple Predictor using averages (median for ordinal time series, most frequent for categorical/nominal) as predicted next values ) . . . . .	488
<a href="#">AveragePredictor.h</a> (A simple Predictor using averages (median for ordinal time series, most frequent for categorical/nominal) as predicted next values ) . . . . .	490
<a href="#">Bluetooth.cpp</a> (Bluetooth feature ) . . . . .	492
<a href="#">Bluetooth.h</a> (Bluetooth feature ) . . . . .	493
<a href="#">BluetoothLinux.cpp</a> (Bluetooth feature ) . . . . .	495
<a href="#">BluetoothLinux.h</a> (Bluetooth feature ) . . . . .	497
<a href="#">BluetoothWindows.cpp</a> (Bluetooth feature ) . . . . .	499
<a href="#">Classifier.h</a> (Classifier interface ) . . . . .	502
<a href="#">combine.c</a> . . . . .	??
<a href="#">conf.c</a> . . . . .	??
<a href="#">conf.h</a> . . . . .	??
<a href="#">debug/config.h</a> . . . . .	??
<a href="#">Features/Video/motion/config.h</a> . . . . .	??
<a href="#">ConfigReader.cpp</a> (Configuration parser ) . . . . .	505
<a href="#">ConfigReader.h</a> (Configuration parser ) . . . . .	506
<a href="#">configure.c</a> . . . . .	??
<a href="#">context.h</a> . . . . .	??
<a href="#">control.c</a> . . . . .	??

<b>control.h</b>	??
<a href="#">debug.cpp</a> (Debug log implementation)	508
<a href="#">debug.h</a> (Debug log declaration)	511
<a href="#">defines.h</a> (Global defines)	514
<a href="#">dll.cpp</a> (DLL wrapper)	517
<a href="#">doxygen.h</a> (Doxygen documentation file)	518
<b>draw.c</b>	??
<b>event.c</b>	??
<b>event.h</b>	??
<a href="#">Feature.h</a> (Feature interface declaration)	519
<a href="#">FeatureContainer.cpp</a> (Feature container)	523
<a href="#">FeatureContainer.h</a> (Feature container)	525
<a href="#">featurevector.cpp</a> (Feature helper functions)	527
<a href="#">featurevector.h</a> (Feature helper function declarations)	529
<b>ffmpeg.c</b>	??
<b>ffmpeg.h</b>	??
<a href="#">GNG.cpp</a> (Growing neural gas)	531
<a href="#">GNG.h</a> (Growing neural gas)	533
<a href="#">GSM.cpp</a> (GSM feature)	535
<a href="#">GSM.h</a> (GSM feature)	537
<a href="#">GSMLinux.cpp</a> (GSM feature)	539
<a href="#">GSMWindows.cpp</a> (GSM feature)	541
<b>gtkwin.c</b>	??
<b>helper.c</b>	??
<b>helper.c.cpp</b>	??
<a href="#">HMM.cpp</a> (Hidden Markov Model ( <a href="#">HMM</a> ) predictor)	543
<a href="#">HMM.h</a> (Hidden Markov Model ( <a href="#">HMM</a> ) predictor)	545
<b>Intelligence.cpp</b>	??
<b>Intelligence.h</b>	??
<b>IntelligenceFactory.cpp</b>	??
<b>IntelligenceFactory.h</b>	??
<a href="#">JNI_Bluetooth.cpp</a> (Bluetooth feature JNI wrapper)	547
<a href="#">JNI_Context.cpp</a> (JNI wrapper)	550
<a href="#">JNI_Samples.cpp</a> (JNI wrapper)	553
<b>libmain.moc.cpp</b>	??
<a href="#">LinuxClassifier.cpp</a> (Classifier linux implementation)	558
<a href="#">LinuxFeatureContainer.cpp</a> (Feature container)	559
<b>LinuxScanner.cpp</b>	??
<a href="#">LoggingFeatureContainer.cpp</a> (Feature container)	560
<a href="#">LoggingFeatureContainer.h</a> (Feature container)	562
<a href="#">main-classification.cpp</a> (Main)	564
<a href="#">main-features.cpp</a> (Main)	567
<a href="#">main-prediction.cpp</a> (Main)	569
<a href="#">servicehost/main.cpp</a> (Windows NT service)	571
<b>trayicon/main.cpp</b>	??
<b>motion-control.c</b>	??
<b>motion/motion.c</b>	??
<b>motion.c</b>	??
<b>motion.h</b>	??
<b>netcam.c</b>	??
<a href="#">Network.cpp</a> (Network feature)	572
<a href="#">Network.h</a> (Wlan feature)	574
<a href="#">Numerical.cpp</a> (Abstract string feature class implementation)	576
<a href="#">Numerical.h</a> (Numerical feature class declaration)	577

<a href="#">PaWrapper.h</a> (Microphone feature ) . . . . .	579
<a href="#">PersistantReader.cpp</a> (Persistant file reader ) . . . . .	581
<a href="#">PersistantReader.h</a> (Persistant file reader ) . . . . .	582
<a href="#">PersistantWriter.cpp</a> (Persistant file writer ) . . . . .	584
<a href="#">PersistantWriter.h</a> (Persistant file writer ) . . . . .	585
<a href="#">PHPCommon.h</a> (PHP feature ) . . . . .	587
<a href="#">PHPWrapper.c</a> (PHP feature ) . . . . .	588
<a href="#">PHPWrapper.cpp</a> (PHP feature ) . . . . .	592
<a href="#">PHPWrapper.h</a> (PHP feature ) . . . . .	595
<b>picture.c</b> . . . . .	??
<b>picture.h</b> . . . . .	??
<b>post.c</b> . . . . .	??
<b>post.h</b> . . . . .	??
<a href="#">Power.cpp</a> (Power feature ) . . . . .	597
<a href="#">Power.h</a> (Power feature ) . . . . .	599
<a href="#">PowerLinux.cpp</a> (Power feature ) . . . . .	601
<a href="#">PowerWindows.cpp</a> (Power feature ) . . . . .	602
<a href="#">Predictor.h</a> (Predictor interface ) . . . . .	603
<b>PreferencesImpl.cpp</b> . . . . .	??
<b>PreferencesImpl.h</b> . . . . .	??
<a href="#">ReplayFeatureContainer.cpp</a> (Feature container ) . . . . .	606
<a href="#">ReplayFeatureContainer.h</a> (Feature container ) . . . . .	608
<b>Scanner.h</b> . . . . .	??
<a href="#">servicehost.cpp</a> (Windows NT service ) . . . . .	610
<a href="#">servicehost.h</a> (Windows NT service ) . . . . .	611
<a href="#">SingleStepDuration.cpp</a> (©2003-2004 by Rene Mayrhofer, Harald Radi ) . . . . .	612
<a href="#">SingleStepDuration.h</a> (A simple Predictor which tries to predict the duration of staying in the current state ) . . . . .	614
<b>soapC.cpp</b> . . . . .	??
<b>soapClient.cpp</b> . . . . .	??
<b>soapClientLib.cpp</b> . . . . .	??
<b>soapContextApplicationsBindingObject.h</b> . . . . .	??
<b>soapContextApplicationsBindingProxy.h</b> . . . . .	??
<b>soapH.h</b> . . . . .	??
<b>soapServer.cpp</b> . . . . .	??
<b>soapServerLib.cpp</b> . . . . .	??
<b>soapStub.h</b> . . . . .	??
<a href="#">splaytree.h</a> (Splaytree template ) . . . . .	616
<a href="#">Statistics.cpp</a> (Predictor interface ) . . . . .	617
<a href="#">Statistics.h</a> (Statistics of time series ) . . . . .	618
<a href="#">symbianstr.c</a> (Missing Symbian functions implementation ) . . . . .	620
<a href="#">symbianstr.h</a> (Missing Symbian functions declaration ) . . . . .	621
<a href="#">SystemCommandStringList.cpp</a> (System command string feature class implementation ) . . . . .	622
<a href="#">SystemCommandStringList.h</a> (System command string feature class declaration ) . . . . .	624
<b>test-preproc.cpp</b> . . . . .	??
<a href="#">Thread.cpp</a> (Abstract thread function implementation ) . . . . .	626
<a href="#">Thread.h</a> (Abstract thread function declaration ) . . . . .	627
<a href="#">ThreadPosix.cpp</a> (Abstract thread function implementation ) . . . . .	629
<a href="#">ThreadSymbian.cpp</a> (Abstract thread function implementation ) . . . . .	630
<a href="#">ThreadWindows.cpp</a> (Abstract thread function implementation ) . . . . .	632
<a href="#">Time.cpp</a> (Time feature ) . . . . .	633
<a href="#">TimeFeature.h</a> (Time feature ) . . . . .	635
<b>track.c</b> . . . . .	??
<b>track.h</b> . . . . .	??

<b>trayicon.moc.cpp</b>	??
<a href="#">trie.h</a> (Trie template)	637
<b>uic_Preferences.cpp</b>	??
<b>uic_Preferences.h</b>	??
<b>uic_Preferences.moc.cpp</b>	??
<b>unicode.cpp</b>	??
<a href="#">Unit.cpp</a> (Growing neural gas)	639
<a href="#">Unit.h</a> (Growing neural gas)	640
<a href="#">version.h</a> (Windows NT service)	642
<b>motion/video.c</b>	??
<b>video.c</b>	??
<a href="#">Video.cpp</a> (Video feature)	643
<b>motion/video.h</b>	??
<a href="#">video.h</a> (Video feature)	645
<b>webcam.c</b>	??
<b>webcam.h</b>	??
<a href="#">WindowsClassifier.cpp</a> (Classifier windows implementation)	647
<a href="#">WindowsFeatureContainer.cpp</a> (Feature container)	648
<a href="#">WindowsPredictor.cpp</a> (Predictor windows implementation)	649
<b>WindowsScanner.cpp</b>	??
<a href="#">Wlan.cpp</a> (Wlan feature)	650
<a href="#">Wlan.h</a> (Wlan feature)	651
<a href="#">WlanAP.cpp</a> (Wlan feature)	653
<a href="#">WlanAP.h</a> (Wlan feature)	655
<a href="#">WlanLinux.cpp</a> (Wlan feature)	657
<a href="#">WlanLinux.h</a> (Wlan feature)	659
<a href="#">WlanSymbolAPI.cpp</a> (Wlan feature)	661
<a href="#">WlanWindows.cpp</a> (Wlan feature)	663
<b>xmlrpc-httpd.c</b>	??
<b>xmlrpc-httpd.h</b>	??

# Chapter 5

## Intelligence Page Index

### 5.1 Intelligence Related Pages

Here is a list of all related documentation pages:

Introduction . . . . .	665
Installation . . . . .	666
Framework . . . . .	667
Todo List . . . . .	668



## Chapter 6

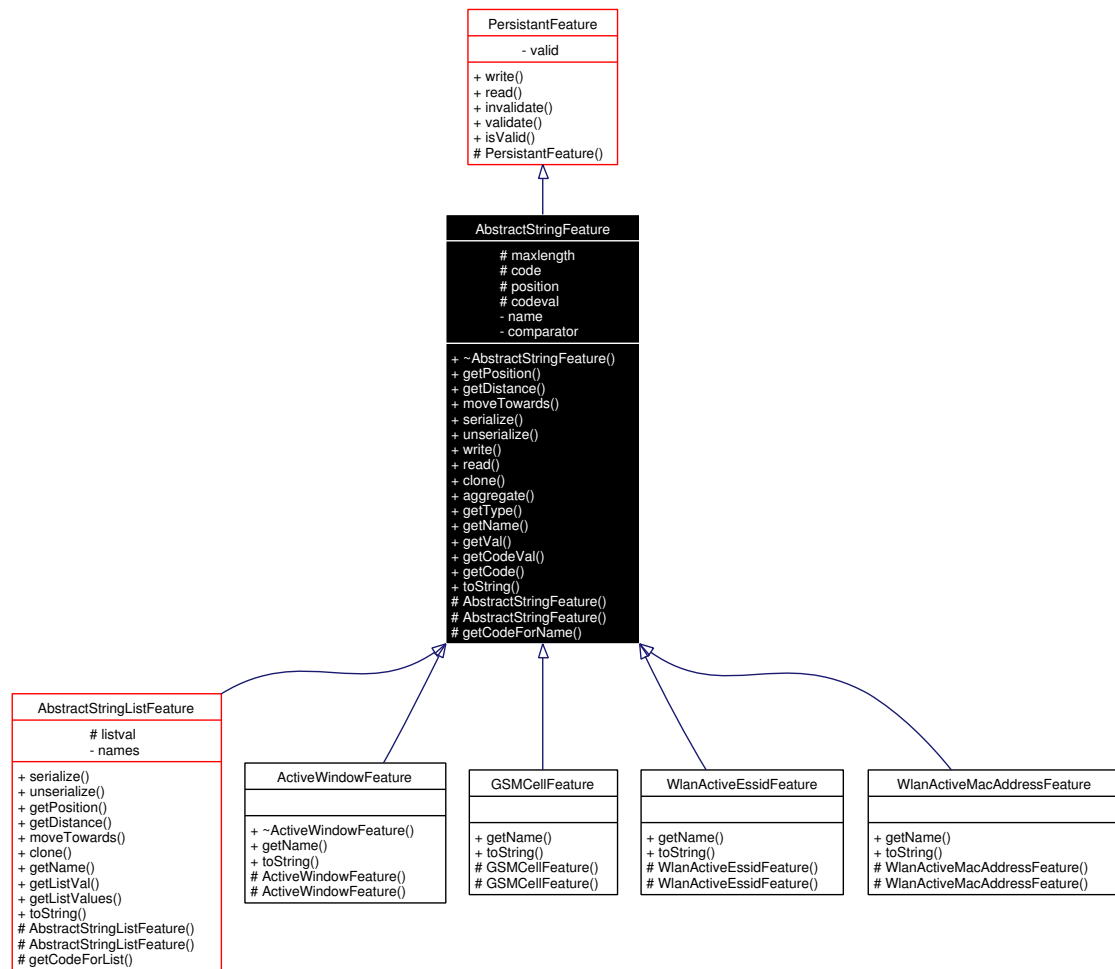
# Intelligence Class Documentation

### 6.1 AbstractStringFeature Class Reference

Baseclass for features providing string values.

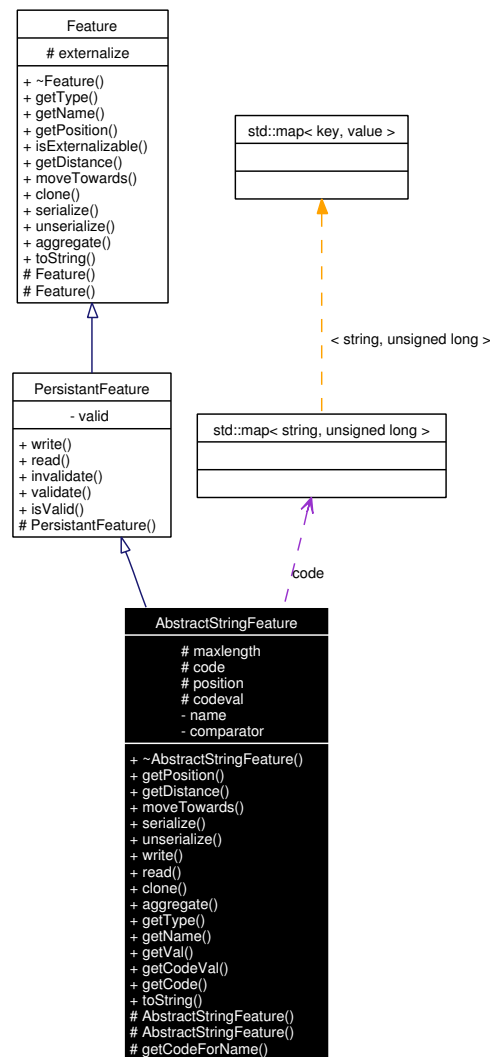
```
#include <AbstractString.h>
```

Inheritance diagram for AbstractStringFeature:



Collaboration diagram for AbstractStringFeature:





## Public Types

- enum [ComparatorType](#) { **None**, **Levenshtein** }

*The distance metric to use.*

- enum [FeatureType](#) {  
**boolean**, **nominal**, **ordinal**, **numerical\_discrete**,  
**numerical\_continuous** }

*Possible feature types.*

## Public Member Functions

- virtual double [getPosition](#) () const

*Query a features position.*

- virtual double `getDistance (Feature *f)` const  
*Calculates the distance between two features.*
- virtual void `moveTowards (Feature *f, double factor)`  
*Move feature.*
- virtual string `serialize ()` const  
*Serialize a samples data to a string.*
- virtual void `unserialize (string value)`  
*Unserialize a samples data from a string.*
- virtual `featureparams write ()` const  
*Externalize feature.*
- virtual void `read (featureparams *param)`  
*Load feature from persistant data.*
- virtual `Feature * clone ()` const  
*Clone a feature.*
- virtual void `aggregate (aggregatelist samples)`  
*Aggregate a sample values from other sources.*
- virtual `FeatureType getType ()` const  
*Query a features type.*
- virtual const string `getName ()` const  
*Query a features name.*
- const string & `getVal ()` const  
*Query the string values.*
- const unsigned long `getCodeVal ()` const
- const `stringcode * getCode ()`
- virtual string `toString ()` const  
*Get feature as string.*
- void `invalidate ()`  
*Invalidate feature.*
- void `validate ()`  
*Set the validation flag to true.*
- bool `isValid ()`

*Query validation flag.*

- bool `isExternalizable()`  
*Query externalization flag.*

## Protected Member Functions

- `AbstractStringFeature` (`ComparatorType` comp, `stringcode` \*code, long \*maxlen)  
*Feature constructor*
- `AbstractStringFeature` (`ComparatorType` comp, `stringcode` \*code, long \*maxlen, const string &name)  
*Sample constructor.*
- unsigned long `getCodeForName` (const string &name)  
*Get a code value for a given string.*

## Protected Attributes

- long \* `maxlength`  
*A reference to the static, persistent maximum length of strings seen so far.*
- `stringcode` \* `code`  
*A reference to the static, persistent code table of strings seen so far.*
- char \* `position`  
*Only for internal usage in `moveTowards` and `getDistance`.*
- unsigned long `codeval`  
*The coded value of the feature.*
- const bool `externalize`  
*Externalization flag.*

## Private Attributes

- string `name`  
*The feature value.*
- `ComparatorType` `comparator`  
*The selected distance metric.*

## Related Functions

(Note that these are not member functions.)

- long [levenshtein](#) (char \*\*x, const char \*t, double \*factor)  
*Modified Levenshtein algorithm.*

### 6.1.1 Detailed Description

Baseclass for features providing string values.

A feature of type nominal which has character strings as values. For comparing values of this feature, either the normal string comparison with the number of different characters (Manhattan distance) or the Levenshtein distance can be used.

Definition at line 43 of file AbstractString.h.

### 6.1.2 Constructor & Destructor Documentation

#### 6.1.2.1 AbstractStringFeature::AbstractStringFeature ([ComparatorType](#) comp, [stringcode](#) \* code, long \* maxlen) [protected]

Feature constructor

This constructor initializes the feature as prototypes value. For creating a specific sample of a feature, the sample constructor should be used.

##### Parameters:

- comp** The distance metric that should be used.
- code** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the code table.
- maxlen** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum string length.

Definition at line 189 of file AbstractString.cpp.

References `codeval`, `comparator`, `maxlength`, `position`, `rand_double`, and `stringcode`.

Referenced by `clone()`.

#### 6.1.2.2 AbstractStringFeature::AbstractStringFeature ([ComparatorType](#) comp, [stringcode](#) \* code, long \* maxlen, const string & name) [protected]

Sample constructor.

This constructor initializes the feature with a specific value and should thus be used for samples.

##### Parameters:

- comp** The distance metric that should be used.
- code** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the code table.

**maxlen** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum string length.

**name** The string representation of the sample value.

Definition at line 212 of file `AbstractString.cpp`.

References `codeval`, `comparator`, `getCodeForName()`, `maxlength`, `position`, and `stringcode`.

### 6.1.3 Member Function Documentation

#### 6.1.3.1 `void AbstractStringFeature::aggregate (aggregatelist samples) [virtual]`

Aggregate a sample values from other sources.

**Parameters:**

**samples** A list of `<timestamp, sample>` tuples.

Implements [Feature](#).

Definition at line 421 of file `AbstractString.cpp`.

References `aggregatelist`.

#### 6.1.3.2 `Feature * AbstractStringFeature::clone () const [virtual]`

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 403 of file `AbstractString.cpp`.

References `AbstractStringFeature()`, `code`, `codeval`, `comparator`, `maxlength`, `name`, and `position`.

#### 6.1.3.3 `unsigned long AbstractStringFeature::getCodeForName (const string & name) [protected]`

Get a code value for a given string.

A helper function to return the respective code for a given string. If the string is encountered the first time a new code is allocated.

**Parameters:**

**name** The string to code

**Returns:**

A code value

Definition at line 303 of file `AbstractString.cpp`.

References `code`, `PersistentFeature::invalidate()`, and `maxlength`.

Referenced by `AbstractStringFeature()`, and `AbstractStringListFeature::getCodeForList()`.

#### 6.1.3.4 double AbstractStringFeature::getDistance ([Feature](#) \**f*) const [virtual]

Calculates the distance between two features.

**Parameters:**

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 337 of file AbstractString.cpp.

References code, codeval, comparator, levenshtein(), maxlength, and position.

#### 6.1.3.5 virtual const string AbstractStringFeature::getName () const [inline, virtual]

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#), [ActiveWindowFeature](#), [BluetoothPeersFeature](#), [GSMCellFeature](#), [SystemCommandStringListFeature](#), [WlanActiveEssidFeature](#), [WlanActiveMacAddressFeature](#), and [WlanPeersFeature](#).

Definition at line 101 of file AbstractString.h.

#### 6.1.3.6 double AbstractStringFeature::getPosition () const [virtual]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 323 of file AbstractString.cpp.

References maxlength, and position.

**6.1.3.7 virtual [FeatureType](#) AbstractStringFeature::getType () const** [inline, virtual]

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 100 of file AbstractString.h.

**6.1.3.8 const string& AbstractStringFeature::getVal () const** [inline]

Query the string values.

**Returns:**

Values list

Definition at line 108 of file AbstractString.h.

References name.

Referenced by Java\_at\_jku\_intelligence\_samples\_StringSample\_nativeGetVal().

**6.1.3.9 void PersistentFeature::invalidate ()** [inline, inherited]

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature::NumericalDiscreteFeature(), TimeFeature::TimeFeature(), NumericalContinuousFeature::unserialize(), and NumericalDiscreteFeature::unserialize().

**6.1.3.10 bool Feature::isExternalizable ()** [inline, inherited]

Query externalization flag.

**Returns:**

*true* if the feature is externalizeable ( i.e. it has persistent data that should be stored across restarts), *false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.1.3.11 bool PersistentFeature::isValid ()** [inline, inherited]

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

#### 6.1.3.12 void AbstractStringFeature::moveTowards ([Feature](#) \**f*, double *factor*) [virtual]

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

##### Parameters:

*f* [Feature](#) used for distance measurement.

*factor* Distance weight.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 371 of file AbstractString.cpp.

References code, codeval, comparator, levenshtein(), position, and rand\_double.

#### 6.1.3.13 void AbstractStringFeature::read ([featureparams](#) \**param*) [virtual]

Load feature from persistant data.

Initializes the persistant feature data from the given representation.

##### Parameters:

*param* Persistant feature data.

##### See also:

[write](#)

Implements [PersistantFeature](#).

Definition at line 296 of file AbstractString.cpp.

References code, and featureparams.

#### 6.1.3.14 string AbstractStringFeature::serialize () const [virtual]

Serialize a samples data to a string.

##### Returns:

String representation of the samples data.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 228 of file AbstractString.cpp.

References codeval, comparator, and position.

Referenced by WlanActiveMacAddressFeature::toString(), WlanActiveEssidFeature::toString(), toString(), and toupperstr().



**6.1.3.15** `string AbstractStringFeature::toString () const` [virtual]

Get feature as string.

**Note:**

This is only for testing.

**Returns:**

[Feature](#) as string.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#), [ActiveWindowFeature](#), [BluetoothPeersFeature](#), [GSMCellFeature](#), [SystemCommandStringListFeature](#), [WlanActiveEssidFeature](#), [WlanActiveMacAddressFeature](#), and [WlanPeersFeature](#).

Definition at line 426 of file `AbstractString.cpp`.

References `name`, and `serialize()`.

**6.1.3.16** `void AbstractStringFeature::unserialize (string value)` [virtual]

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 251 of file `AbstractString.cpp`.

References `code`, `codeval`, `comparator`, `maxlength`, and `position`.

**6.1.3.17** `featureparams AbstractStringFeature::write () const` [virtual]

Externalize feature.

**Returns:**

Persistent feature data.

**See also:**

[read](#)

Implements [PersistentFeature](#).

Definition at line 283 of file `AbstractString.cpp`.

References `code`, and `featureparams`.

**6.1.4 Friends And Related Function Documentation****6.1.4.1** `long levenshtein (char ** x, const char * t, double * factor)` [related]

Modified Levenshtein algorithm.

This function implements a slightly modified version of the Levenshtein algorithm to not only calculate the distance of two strings but also modify the string  $x$  towards the string  $t$  with the propability *factor*. In other words, every transformation operation found by the algorithm is performed on the string  $x$  with the propability *factor*.

**Parameters:**

$x$  Input string.

$t$  String to compare the input string to.

*factor* Propability with witch transformations are performed. Has to be a value out of the intervall  $[0; 1]$ .

**Returns:**

The levenshtein distance between  $x$  and  $t$ .

**Todo**

alloc once

Definition at line 46 of file AbstractString.cpp.

References rand\_double.

Referenced by getDistance(), and moveTowards().

## 6.1.5 Member Data Documentation

### 6.1.5.1 unsigned long [AbstractStringFeature::codeval](#) [protected]

The coded value of the feature.

**See also:**

[name](#)

[code](#)

Definition at line 144 of file AbstractString.h.

Referenced by AbstractStringFeature(), clone(), getCodeVal(), getDistance(), moveTowards(), serialize(), and unserialize().

### 6.1.5.2 const bool [Feature::externalize](#) [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistant features, the object should be cast to PersistentFeature, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**

[PersistantFeature](#)

Definition at line 187 of file Feature.h.

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

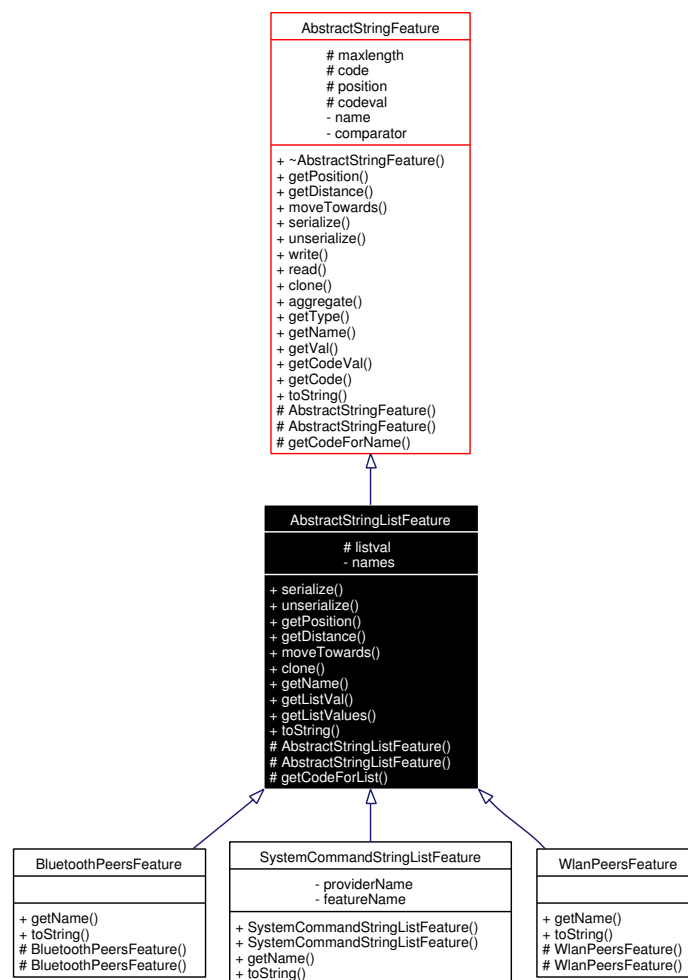
- [AbstractString.h](#)
- [AbstractString.cpp](#)

## 6.2 AbstractStringListFeature Class Reference

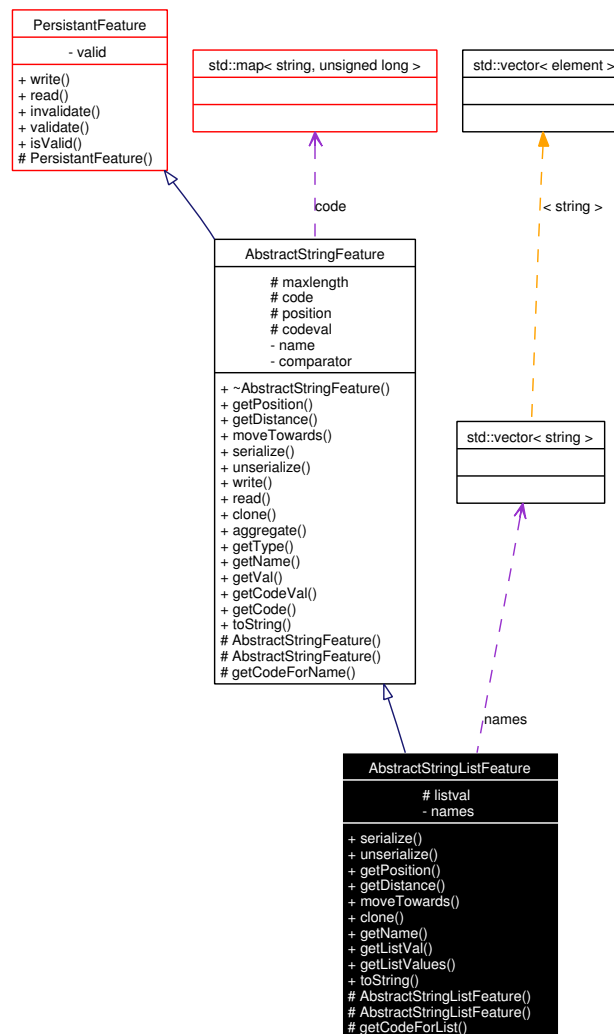
Baseclass for features providing lists of string values.

```
#include <AbstractString.h>
```

Inheritance diagram for AbstractStringListFeature:



Collaboration diagram for AbstractStringListFeature:



## Public Types

- enum **ComparatorType** { **None**, **Levenshtein** }

*The distance metric to use.*

- enum **FeatureType** {  
**boolean**, **nominal**, **ordinal**, **numerical\_discrete**,  
**numerical\_continuous** }

*Possible feature types.*

## Public Member Functions

- virtual string **serialize** () const  
*Serialize a samples data to a string.*

- virtual void `unserialize` (string value)  
*Unserialize a samples data from a string.*
- virtual double `getPosition` () const  
*Query a features position.*
- virtual double `getDistance` (Feature \*f) const  
*Calculates the distance between two features.*
- virtual void `moveTowards` (Feature \*f, double factor)  
*Move feature.*
- virtual Feature \* `clone` () const  
*Clone a feature.*
- virtual const string `getName` () const  
*Query a features name.*
- const bit\_vector & `getListVal` () const  
*Query the list of values.*
- const stringvector & `getListValues` () const  
*Query the list of values.*
- virtual string `toString` () const  
*Get feature as string.*
- virtual `featureparams write` () const  
*Externalize feature.*
- virtual void `read` (featureparams \*param)  
*Load feature from persistant data.*
- virtual void `aggregate` (aggregatelist samples)  
*Aggregate a sample values from other sources.*
- virtual FeatureType `getType` () const  
*Query a features type.*
- const string & `getVal` () const  
*Query the string values.*
- const unsigned long `getCodeVal` () const
- const stringcode \* `getCode` ()
- void `invalidate` ()  
*Invalidate feature.*

- void `validate` ()  
*Set the validation flag to true.*
- bool `isValid` ()  
*Query validation flag.*
- bool `isExternalizable` ()  
*Query externalization flag.*

## Protected Member Functions

- `AbstractStringListFeature` (`stringcode` \*code, long \*maxlen)  
*Feature constructor*
- `AbstractStringListFeature` (`stringcode` \*code, long \*maxlen, const `stringvector` \*names)  
*Sample constructor.*
- bit\_vector `getCodeForList` (const `stringvector` \*names)  
*Get a code value for a given stringvector.*
- unsigned long `getCodeForName` (const string &name)  
*Get a code value for a given string.*

## Protected Attributes

- bit\_vector `listval`  
*The coded value of the feature.*
- long \* `maxlength`  
*A reference to the static, persistent maximum length of strings seen so far.*
- `stringcode` \* `code`  
*A reference to the static, persistent code table of strings seen so far.*
- char \* `position`  
*Only for internal usage in `moveTowards` and `getDistance`.*
- unsigned long `codeval`  
*The coded value of the feature.*
- const bool `externalize`  
*Externalization flag.*

## Private Attributes

- [stringvector names](#)  
*The feature value.*

## Related Functions

(Note that these are not member functions.)

- long [levenshtein](#) (char \*\*x, const char \*t, double \*factor)  
*Modified Levenshtein algorithm.*

### 6.2.1 Detailed Description

Baseclass for features providing lists of string values.

A feature of type nominal which has lists of strings as values.

Definition at line 167 of file AbstractString.h.

### 6.2.2 Constructor & Destructor Documentation

#### 6.2.2.1 AbstractStringListFeature::AbstractStringListFeature ([stringcode](#) \* code, long \* maxlen) [protected]

Feature constructor

This constructor initializes the feature as prototypes value. For creating a specific sample of a feature, the sample constructor should be used.

##### Parameters:

- code** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the code table.
- maxlen** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum string length.

Definition at line 435 of file AbstractString.cpp.

References `listval`, `rand_double`, and `stringcode`.

Referenced by `clone()`.

#### 6.2.2.2 AbstractStringListFeature::AbstractStringListFeature ([stringcode](#) \* code, long \* maxlen, const [stringvector](#) \* names) [protected]

Sample constructor.

This constructor initializes the feature with a specific value and should thus be used for samples.

##### Parameters:

- code** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the code table.



**maxlen** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum string length.

**names** The string representation of the sample value.

Definition at line 443 of file `AbstractString.cpp`.

References `getCodeForList()`, `listval`, `stringcode`, and `stringvector`.

## 6.2.3 Member Function Documentation

### 6.2.3.1 `void AbstractStringFeature::aggregate (aggregatelist samples) [virtual, inherited]`

Aggregate a sample values from other sources.

**Parameters:**

**samples** A list of `<timestamp, sample>` tuples.

Implements [Feature](#).

Definition at line 421 of file `AbstractString.cpp`.

References `aggregatelist`.

### 6.2.3.2 `Feature * AbstractStringListFeature::clone () const [virtual]`

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Reimplemented from [AbstractStringFeature](#).

Definition at line 581 of file `AbstractString.cpp`.

References `AbstractStringListFeature()`, `listval`, and `names`.

### 6.2.3.3 `bit_vector AbstractStringListFeature::getCodeForList (const stringvector * names) [protected]`

Get a code value for a given stringvector.

The function composes a `bit_vector` wherein for each string in the stringvector the bit on the position of the strings code value is set to *true*. The remaining bits are initialized to *false*.

**Parameters:**

**names** The stringvector to code

**Returns:**

A code value as `bit_vector`

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

References `AbstractStringFeature::getCodeForName()`, and `stringvector`.

Referenced by `AbstractStringListFeature()`, and `WlanPeersFeature::WlanPeersFeature()`.

#### 6.2.3.4 unsigned long AbstractStringFeature::getCodeForName (const string & name) [protected, inherited]

Get a code value for a given string.

A helper function to return the respective code for a given string. If the string is encountered the first time a new code is allocated.

##### Parameters:

*name* The string to code

##### Returns:

A code value

Definition at line 303 of file AbstractString.cpp.

References AbstractStringFeature::code, PersistentFeature::invalidate(), and AbstractStringFeature::maxlength.

Referenced by AbstractStringFeature::AbstractStringFeature(), and getCodeForList().

#### 6.2.3.5 double AbstractStringListFeature::getDistance (Feature \* f) const [virtual]

Calculates the distance between two features.

##### Parameters:

*f* Feature used for distance measurement.

##### Returns:

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Reimplemented from AbstractStringFeature.

Definition at line 518 of file AbstractString.cpp.

References listval.

#### 6.2.3.6 const bit\_vector& AbstractStringListFeature::getListVal () const [inline]

Query the list of values.

##### Returns:

Values list

Definition at line 244 of file AbstractString.h.

#### 6.2.3.7 const stringvector& AbstractStringListFeature::getListValues () const [inline]

Query the list of values.

##### Returns:

Values list

Definition at line 253 of file AbstractString.h.

References [stringvector](#).

Referenced by [Java\\_at\\_jku\\_intelligence\\_samples\\_StringListSample\\_nativeGetValues\(\)](#), [WlanPeersFeature::toString\(\)](#), [SystemCommandStringListFeature::toString\(\)](#), [BluetoothPeersFeature::toString\(\)](#), and [toString\(\)](#).

#### 6.2.3.8 virtual const string AbstractStringListFeature::getName () const [inline, virtual]

Query a features name.

##### Returns:

Name of the [Feature](#) in the format "Featureprovider.Feature"

Reimplemented from [AbstractStringFeature](#).

Reimplemented in [BluetoothPeersFeature](#), [SystemCommandStringListFeature](#), and [WlanPeersFeature](#).

Definition at line 237 of file AbstractString.h.

#### 6.2.3.9 double AbstractStringListFeature::getPosition () const [virtual]

Query a features position.

##### Returns:

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

##### Remarks:

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Reimplemented from [AbstractStringFeature](#).

Definition at line 498 of file AbstractString.cpp.

References [listval](#).

#### 6.2.3.10 virtual FeatureType AbstractStringFeature::getType () const [inline, virtual, inherited]

Query a features type.

##### Returns:

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 100 of file AbstractString.h.

**6.2.3.11 const string& AbstractStringFeature::getVal () const** [inline, inherited]

Query the string values.

**Returns:**

Values list

Definition at line 108 of file AbstractString.h.

References AbstractStringFeature::name.

Referenced by Java\_at\_jku\_intelligence\_samples\_StringSample\_nativeGetVal().

**6.2.3.12 void PersistentFeature::invalidate ()** [inline, inherited]

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by AbstractStringFeature::getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature::NumericalDiscreteFeature(), TimeFeature::TimeFeature(), NumericalContinuousFeature::unserialize(), and NumericalDiscreteFeature::unserialize().

**6.2.3.13 bool Feature::isExternalizable ()** [inline, inherited]

Query externalization flag.

**Returns:**

*true* if the feature is externalizable ( i.e. it has persistent data that should be stored across restarts), *false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.2.3.14 bool PersistentFeature::isValid ()** [inline, inherited]

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

**6.2.3.15 void AbstractStringListFeature::moveTowards (Feature \*f, double factor)** [virtual]

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample s.

**Parameters:**

- f* [Feature](#) used for distance measurement.
- factor* Distance weight.

Reimplemented from [AbstractStringFeature](#).

Definition at line 546 of file AbstractString.cpp.

References listval, and rand\_double.

**6.2.3.16 void AbstractStringFeature::read (featureparams \*param) [virtual, inherited]**

Load feature from persistant data.

Initializes the persistant feature data from the given representation.

**Parameters:**

- param* Persistant feature data.

**See also:**

[write](#)

Implements [PersistantFeature](#).

Definition at line 296 of file AbstractString.cpp.

References AbstractStringFeature::code, and featureparams.

**6.2.3.17 string AbstractStringListFeature::serialize () const [virtual]**

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Reimplemented from [AbstractStringFeature](#).

Definition at line 470 of file AbstractString.cpp.

References listval.

Referenced by WlanPeersFeature::toString(), BluetoothPeersFeature::toString(), and toString().

**6.2.3.18 string AbstractStringListFeature::toString () const [virtual]**

Get feature as string.

**Note:**

This is only for testing.

**Returns:**

[Feature](#) as string.

Reimplemented from [AbstractStringFeature](#).

Reimplemented in [BluetoothPeersFeature](#), [SystemCommandStringListFeature](#), and [WlanPeersFeature](#).

Definition at line 593 of file AbstractString.cpp.

References [getListValues\(\)](#), and [serialize\(\)](#).

#### 6.2.3.19 void AbstractStringListFeature::unserialize (string *value*) [virtual]

Unserialize a samples data from a string.

##### Parameters:

*value* String representation of the samples data.

Reimplemented from [AbstractStringFeature](#).

Definition at line 480 of file AbstractString.cpp.

References [listval](#).

#### 6.2.3.20 [featureparams](#) AbstractStringFeature::write () const [virtual, inherited]

Externalize feature.

##### Returns:

Persistent feature data.

##### See also:

[read](#)

Implements [PersistentFeature](#).

Definition at line 283 of file AbstractString.cpp.

References [AbstractStringFeature::code](#), and [featureparams](#).

## 6.2.4 Friends And Related Function Documentation

#### 6.2.4.1 long levenshtein (char \*\* *x*, const char \* *t*, double \* *factor*) [related, inherited]

Modified Levenshtein algorithm.

This function implements a slightly modified version of the Levenshtein algorithm to not only calculate the distance of two strings but also modify the string *x* towards the string *t* with the propability *factor*. In other words, every transformation operation found by the algorithm is performed on the string *x* with the propability *factor*.

##### Parameters:

*x* Input string.

*t* String to compare the input string to.

*factor* Propability with witch transformations are performed. Has to be a value out of the intervall [0; 1].

**Returns:**

The levenshtein distance between  $x$  and  $t$ .

**Todo**

alloc once

Definition at line 46 of file AbstractString.cpp.

References rand\_double.

Referenced by AbstractStringFeature::getDistance(), and AbstractStringFeature::moveTowards().

## 6.2.5 Member Data Documentation

### 6.2.5.1 unsigned long [AbstractStringFeature::codeval](#) [protected, inherited]

The coded value of the feature.

**See also:**

[name](#)  
[code](#)

Definition at line 144 of file AbstractString.h.

Referenced by AbstractStringFeature::AbstractStringFeature(), AbstractStringFeature::clone(), AbstractStringFeature::getCodeVal(), AbstractStringFeature::getDistance(), AbstractStringFeature::moveTowards(), AbstractStringFeature::serialize(), and AbstractStringFeature::unserialize().

### 6.2.5.2 const bool [Feature::externalize](#) [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to PersistentFeature, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**

[PersistentFeature](#)

Definition at line 187 of file Feature.h.

### 6.2.5.3 bit\_vector [AbstractStringListFeature::listval](#) [protected]

The coded value of the feature.

**See also:**

[names](#)  
[code](#)

Definition at line 180 of file AbstractString.h.

Referenced by AbstractStringListFeature(), clone(), getDistance(), getPosition(), moveTowards(), serialize(), and unserialize().

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

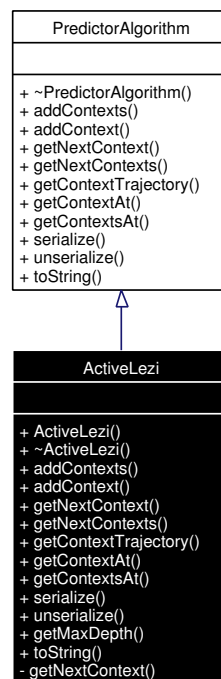
- [AbstractString.h](#)
- [AbstractString.cpp](#)



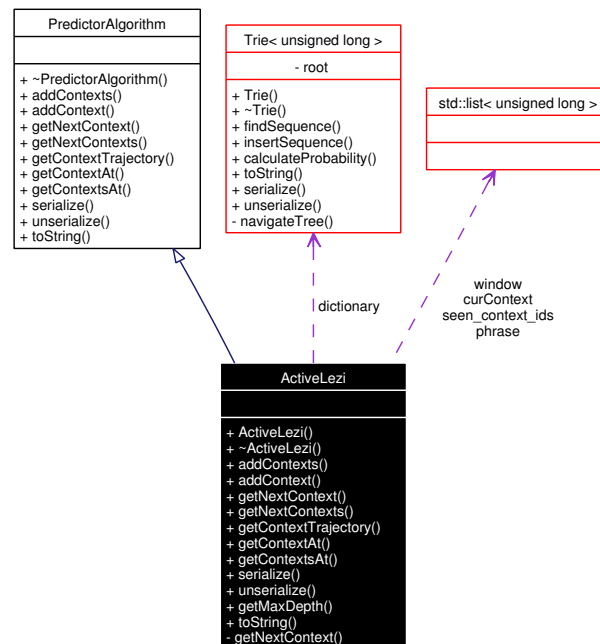
## 6.3 ActiveLezi Class Reference

```
#include <ActiveLezi.h>
```

Inheritance diagram for ActiveLezi:



Collaboration diagram for ActiveLezi:



## Public Member Functions

- [ActiveLezi](#) ([predictorparams](#) &params)
- virtual [~ActiveLezi](#) ()  
*Destructor.*
- virtual void [addContexts](#) (const [membershiplist](#) \*contexts, time\_t time)
- virtual void [addContext](#) (unsigned long contextId, time\_t time)
- virtual unsigned long [getNextContext](#) () const
- virtual [membershiplist](#) [getNextContexts](#) () const
- virtual [contexttrajectory](#) [getContextTrajectory](#) (unsigned int start, unsigned int end) const
- virtual unsigned long [getContextAt](#) (time\_t time) const
- virtual [membershiplist](#) [getContextsAt](#) (time\_t time) const
- virtual string [serialize](#) () const  
*Serialize a predictors data to a string.*
- virtual void [unserialize](#) (string data)  
*Unserialize a predictors data from a string.*
- unsigned int [getMaxDepth](#) () const
- virtual string [toString](#) () const  
*This is only for testing.*

## Private Member Functions

- unsigned long [getNextContext](#) (const [list](#)< unsigned long > \*context) const

## Private Attributes

- [Trie](#)< unsigned long > [dictionary](#)  
*Private property.*
- [list](#)< unsigned long > [phrase](#)  
*Private property.*
- [list](#)< unsigned long > [window](#)  
*Private property.*
- unsigned int [max\\_lz\\_length](#)  
*Private property.*
- unsigned int [maxWindowModelDepth](#)  
*Private property.*
- [list](#)< unsigned long > [curContext](#)  
*Private property.*
- [list](#)< unsigned long > [seen\\_context\\_ids](#)  
*Private property.*

### 6.3.1 Detailed Description

#### [Todo](#)

documentation

Definition at line 33 of file ActiveLezi.h.

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 [ActiveLezi::ActiveLezi](#) ([predictorparams](#) & *params*)

#### [Todo](#)

documentation

Definition at line 33 of file ActiveLezi.cpp.

References [maxWindowModelDepth](#).

### 6.3.3 Member Function Documentation

#### 6.3.3.1 [void ActiveLezi::addContext](#) (unsigned long *contextId*, time\_t *time*) [virtual]

#### [Todo](#)

documentation

Implements [PredictorAlgorithm](#).

Definition at line 56 of file ActiveLezi.cpp.

References curContext, dictionary, max\_lz\_length, maxWindowModelDepth, phrase, and seen\_context\_ids.

Referenced by main().

**6.3.3.2 void ActiveLezi::addContexts (const [membershiplist](#) \* contexts, time\_t time) [virtual]**

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 49 of file ActiveLezi.cpp.

**6.3.3.3 unsigned long ActiveLezi::getContextAt (time\_t time) const [virtual]**

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 151 of file ActiveLezi.cpp.

**6.3.3.4 [membershiplist](#) ActiveLezi::getContextsAt (time\_t time) const [virtual]**

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 158 of file ActiveLezi.cpp.

**6.3.3.5 [contexttrajectory](#) ActiveLezi::getContextTrajectory (unsigned int start, unsigned int end) const [virtual]**

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 116 of file ActiveLezi.cpp.

References curContext, dictionary, Trie< \_type >::findSequence(), and getNextContext().

**6.3.3.6 unsigned int ActiveLezi::getMaxDepth () const**

**Todo**

documentation

Definition at line 177 of file ActiveLezi.cpp.

**6.3.3.7 unsigned long ActiveLezi::getNextContext () const** [virtual]**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 102 of file ActiveLezi.cpp.

References `Trie< _type >::calculateProbability()`, `curContext`, `dictionary`, `membershipList`, and `seen_context_ids`.

Referenced by `getContextTrajectory()`.

**6.3.3.8 unsigned long ActiveLezi::getNextContext (const [list](#)< unsigned long > \* context) const** [private]**Todo**

documentation

Definition at line 86 of file ActiveLezi.cpp.

References `dictionary`, and `seen_context_ids`.

Referenced by `main()`.

**6.3.3.9 [membershipList](#) ActiveLezi::getNextContexts () const** [virtual]**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 107 of file ActiveLezi.cpp.

Referenced by `main()`.

**6.3.3.10 string ActiveLezi::serialize () const** [virtual]

Serialize a predictors data to a string.

**Returns:**

String representation of the predictor data.

Implements [PredictorAlgorithm](#).

Definition at line 165 of file ActiveLezi.cpp.

**6.3.3.11 void ActiveLezi::unserialize (string data)** [virtual]

Unserialize a predictors data from a string.

**Parameters:**

*data* String representation of the predictor data.

Implements [PredictorAlgorithm](#).

Definition at line 170 of file ActiveLezi.cpp.

### 6.3.4 Member Data Documentation

#### 6.3.4.1 `list<unsigned long> ActiveLezi::curContext` [private]

Private property.

##### **Todo**

documentation

Definition at line 47 of file ActiveLezi.h.

Referenced by `addContext()`, `getContextTrajectory()`, and `getNextContext()`.

#### 6.3.4.2 `Trie<unsigned long> ActiveLezi::dictionary` [private]

Private property.

##### **Todo**

documentation

Definition at line 40 of file ActiveLezi.h.

Referenced by `addContext()`, `getContextTrajectory()`, and `getNextContext()`.

#### 6.3.4.3 `unsigned int ActiveLezi::max_lz_length` [private]

Private property.

##### **Todo**

documentation

Definition at line 43 of file ActiveLezi.h.

Referenced by `addContext()`.

#### 6.3.4.4 `unsigned int ActiveLezi::maxWindowModelDepth` [private]

Private property.

##### **Todo**

documentation

Definition at line 44 of file ActiveLezi.h.

Referenced by `ActiveLezi()`, and `addContext()`.

**6.3.4.5** `list<unsigned long> ActiveLezi::phrase` [private]

Private property.

**Todo**

documentation

Definition at line 41 of file ActiveLezi.h.

Referenced by `addContext()`, and `toString()`.

**6.3.4.6** `list<unsigned long> ActiveLezi::seen_context_ids` [private]

Private property.

**Todo**

documentation

Definition at line 50 of file ActiveLezi.h.

Referenced by `addContext()`, and `getNextContext()`.

**6.3.4.7** `list<unsigned long> ActiveLezi::window` [private]

Private property.

**Todo**

documentation

Definition at line 42 of file ActiveLezi.h.

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

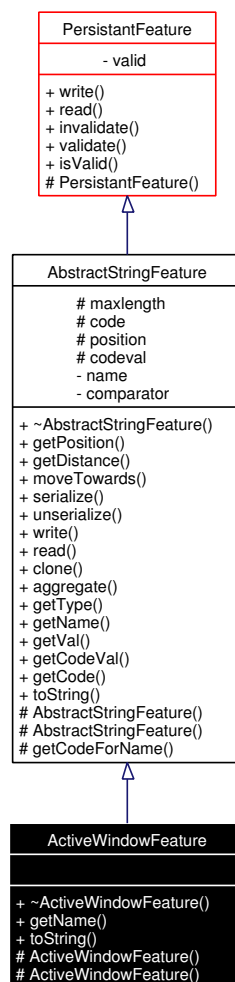
- [ActiveLezi.h](#)
- [ActiveLezi.cpp](#)

## 6.4 ActiveWindowFeature Class Reference

Returns the application with focus.

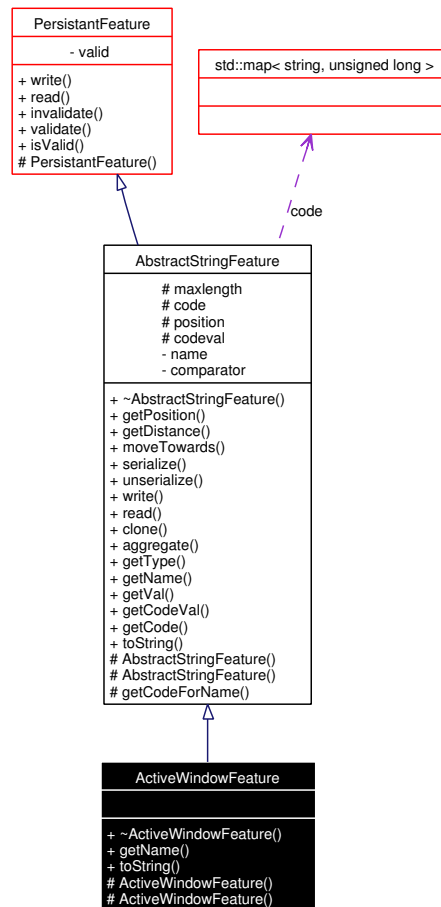
```
#include <ActiveWindow.h>
```

Inheritance diagram for ActiveWindowFeature:



Collaboration diagram for ActiveWindowFeature:





## Public Types

- enum `ComparatorType` { **None**, **Levenshtein** }  
*The distance metric to use.*
- enum `FeatureType` {  
**boolean**, **nominal**, **ordinal**, **numerical\_discrete**,  
**numerical\_continuous** }  
*Possible feature types.*

## Public Member Functions

- virtual const string `getName` () const  
*Query a features name.*
- virtual string `toString` () const  
*Get feature as string.*
- virtual double `getPosition` () const

*Query a features position.*

- virtual double `getDistance` (`Feature *f`) const  
*Calculates the distance between two features.*
- virtual void `moveTowards` (`Feature *f`, double factor)  
*Move feature.*
- virtual string `serialize` () const  
*Serialize a samples data to a string.*
- virtual void `unserialize` (string value)  
*Unserialize a samples data from a string.*
- virtual `featureparams write` () const  
*Externalize feature.*
- virtual void `read` (`featureparams *param`)  
*Load feature from persistant data.*
- virtual `Feature * clone` () const  
*Clone a feature.*
- virtual void `aggregate` (`aggregatelist` samples)  
*Aggregate a sample values from other sources.*
- virtual `FeatureType getType` () const  
*Query a features type.*
- const string & `getVal` () const  
*Query the string values.*
- const unsigned long `getCodeVal` () const
- const `stringcode * getCode` ()
- void `invalidate` ()  
*Invalidate feature.*
- void `validate` ()  
*Set the validation flag to true.*
- bool `isValid` ()  
*Query validation flag.*
- bool `isExternalizable` ()  
*Query externalization flag.*

## Protected Member Functions

- [ActiveWindowFeature](#) ([stringcode](#) \*code, long \*maxlen)  
*Feature constructor*
- [ActiveWindowFeature](#) ([stringcode](#) \*code, long \*maxlen, string window)  
*Sample constructor.*
- unsigned long [getCodeForName](#) (const string &name)  
*Get a code value for a given string.*

## Protected Attributes

- long \* [maxlength](#)  
*A reference to the static, persistant maximum length of strings seen so far.*
- [stringcode](#) \* [code](#)  
*A reference to the static, persistant code table of strings seen so far.*
- char \* [position](#)  
*Only for internal usage in moveTowards and getDistance.*
- unsigned long [codeval](#)  
*The coded value of the feature.*
- const bool [externalize](#)  
*Externalization flag.*

## Related Functions

(Note that these are not member functions.)

- long [levenshtein](#) (char \*\*x, const char \*t, double \*factor)  
*Modified Levenshtein algorithm.*

### 6.4.1 Detailed Description

Returns the application with focus.

This feature tracks the application with focus on the users desktop and returns the name active application. It is based on the [AbstractStringFeature](#) and uses the Levenshtein distance.

Definition at line 36 of file ActiveWindow.h.

## 6.4.2 Constructor & Destructor Documentation

### 6.4.2.1 `ActiveWindowFeature::ActiveWindowFeature (stringcode * code, long * maxlen)` `[inline, protected]`

Feature constructor

This constructor initializes the feature as prototypes value. For creating a specific sample of a feature, the sample constructor should be used.

#### Parameters:

**code** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the code table.

**maxlen** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum string length.

Definition at line 55 of file `ActiveWindow.h`.

References `stringcode`.

### 6.4.2.2 `ActiveWindowFeature::ActiveWindowFeature (stringcode * code, long * maxlen, string window)` `[inline, protected]`

Sample constructor.

This constructor initializes the feature with a specific value and should thus be used for samples.

#### Parameters:

**code** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the code table.

**maxlen** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum string length.

**window** The string representation of the sample value.

Definition at line 71 of file `ActiveWindow.h`.

References `stringcode`.

## 6.4.3 Member Function Documentation

### 6.4.3.1 `void AbstractStringFeature::aggregate (aggregatelist samples)` `[virtual, inherited]`

Aggregate a sample values from other sources.

#### Parameters:

**samples** A list of `<timestamp, sample>` tuples.

Implements [Feature](#).

Definition at line 421 of file `AbstractString.cpp`.

References `aggregatelist`.

**6.4.3.2** [Feature](#) \* **AbstractStringFeature::clone () const** [virtual, inherited]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 403 of file `AbstractString.cpp`.

References `AbstractStringFeature::AbstractStringFeature()`, `AbstractStringFeature::code`, `AbstractStringFeature::codeval`, `AbstractStringFeature::comparator`, `AbstractStringFeature::maxlength`, `AbstractStringFeature::name`, and `AbstractStringFeature::position`.

**6.4.3.3** **unsigned long AbstractStringFeature::getCodeForName (const string & name)**  
[protected, inherited]

Get a code value for a given string.

A helper function to return the respective code for a given string. If the string is encountered the first time a new code is allocated.

**Parameters:**

*name* The string to code

**Returns:**

A code value

Definition at line 303 of file `AbstractString.cpp`.

References `AbstractStringFeature::code`, `PersistentFeature::invalidate()`, and `AbstractStringFeature::maxlength`.

Referenced by `AbstractStringFeature::AbstractStringFeature()`, and `AbstractStringListFeature::getCodeForList()`.

**6.4.3.4** **double AbstractStringFeature::getDistance ([Feature](#) \* f) const** [virtual, inherited]

Calculates the distance between two features.

**Parameters:**

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 337 of file `AbstractString.cpp`.

References `AbstractStringFeature::code`, `AbstractStringFeature::codeval`, `AbstractStringFeature::comparator`, `AbstractStringFeature::levenshtein()`, `AbstractStringFeature::maxlength`, and `AbstractStringFeature::position`.

**6.4.3.5 virtual const string ActiveWindowFeature::getName () const** [inline, virtual]

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Reimplemented from [AbstractStringFeature](#).

Definition at line 76 of file ActiveWindow.h.

**6.4.3.6 double AbstractStringFeature::getPosition () const** [virtual, inherited]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 323 of file AbstractString.cpp.

References AbstractStringFeature::maxlength, and AbstractStringFeature::position.

**6.4.3.7 virtual [FeatureType](#) AbstractStringFeature::getType () const** [inline, virtual, inherited]

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 100 of file AbstractString.h.

**6.4.3.8 const string& AbstractStringFeature::getVal () const** [inline, inherited]

Query the string values.

**Returns:**

Values list

Definition at line 108 of file AbstractString.h.

References AbstractStringFeature::name.

Referenced by Java\_at\_jku\_intelligence\_samples\_StringSample\_nativeGetVal().

**6.4.3.9 void PersistentFeature::invalidate () [inline, inherited]**

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by AbstractStringFeature::getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature::NumericalDiscreteFeature(), TimeFeature::TimeFeature(), NumericalContinuousFeature::unserialize(), and NumericalDiscreteFeature::unserialize().

**6.4.3.10 bool Feature::isExternalizable () [inline, inherited]**

Query externalization flag.

**Returns:**

*true* if the feature is externalizable ( i.e. it has persistent data that should be stored across restarts), *false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.4.3.11 bool PersistentFeature::isValid () [inline, inherited]**

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

**6.4.3.12 void AbstractStringFeature::moveTowards (Feature \**f*, double *factor*) [virtual, inherited]**

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

**Parameters:**

*f* [Feature](#) used for distance measurement.

*factor* Distance weight.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 371 of file AbstractString.cpp.

References AbstractStringFeature::code, AbstractStringFeature::codeval, AbstractStringFeature::comparator, AbstractStringFeature::levenshtein(), AbstractStringFeature::position, and rand\_double.

**6.4.3.13 void AbstractStringFeature::read ([featureparams](#) \* *param*)** [virtual, inherited]

Load feature from persistent data.

Initializes the persistent feature data from the given representation.

**Parameters:**

*param* Persistent feature data.

**See also:**

[write](#)

Implements [PersistentFeature](#).

Definition at line 296 of file AbstractString.cpp.

References AbstractStringFeature::code, and featureparams.

**6.4.3.14 string AbstractStringFeature::serialize () const** [virtual, inherited]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 228 of file AbstractString.cpp.

References AbstractStringFeature::codeval, AbstractStringFeature::comparator, and AbstractStringFeature::position.

Referenced by WlanActiveMacAddressFeature::toString(), WlanActiveEssidFeature::toString(), AbstractStringFeature::toString(), and toupperstr().

**6.4.3.15 string ActiveWindowFeature::toString () const** [virtual]

Get feature as string.

**Note:**

This is only for testing.

**Returns:**

[Feature](#) as string.

Reimplemented from [AbstractStringFeature](#).

Definition at line 36 of file ActiveWindow.cpp.

**6.4.3.16 void AbstractStringFeature::unserialize (string *value*)** [virtual, inherited]

Unserialize a samples data from a string.



**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 251 of file AbstractString.cpp.

References [AbstractStringFeature::code](#), [AbstractStringFeature::codeval](#), [AbstractStringFeature::comparator](#), [AbstractStringFeature::maxlength](#), and [AbstractStringFeature::position](#).

**6.4.3.17 [featureparams](#) AbstractStringFeature::write () const** [virtual, inherited]

Externalize feature.

**Returns:**

Persistant feature data.

**See also:**

[read](#)

Implements [PersistantFeature](#).

Definition at line 283 of file AbstractString.cpp.

References [AbstractStringFeature::code](#), and [featureparams](#).

**6.4.4 Friends And Related Function Documentation****6.4.4.1 [long levenshtein](#) (char \*\* *x*, const char \* *t*, double \* *factor*)** [related, inherited]

Modified Levenshtein algorithm.

This function implements a slightly modified version of the Levenshtein algorithm to not only calculate the distance of two strings but also modify the string *x* towards the string *t* with the propability *factor*. In other words, every transformation operation found by the algorithm is performed on the string *x* with the propability *factor*.

**Parameters:**

*x* Input string.

*t* String to compare the input string to.

*factor* Propability with witch transformations are performed. Has to be a value out of the intervall  $[0; 1]$ .

**Returns:**

The levenshtein distance between *x* and *t*.

**Todo**

alloc once

Definition at line 46 of file AbstractString.cpp.

References [rand\\_double](#).

Referenced by [AbstractStringFeature::getDistance\(\)](#), and [AbstractStringFeature::moveTowards\(\)](#).

## 6.4.5 Member Data Documentation

### 6.4.5.1 unsigned long [AbstractStringFeature::codeval](#) [protected, inherited]

The coded value of the feature.

See also:

[name](#)  
[code](#)

Definition at line 144 of file [AbstractString.h](#).

Referenced by [AbstractStringFeature::AbstractStringFeature\(\)](#), [AbstractStringFeature::clone\(\)](#), [AbstractStringFeature::getCodeVal\(\)](#), [AbstractStringFeature::getDistance\(\)](#), [AbstractStringFeature::moveTowards\(\)](#), [AbstractStringFeature::serialize\(\)](#), and [AbstractStringFeature::unserialize\(\)](#).

### 6.4.5.2 const bool [Feature::externalize](#) [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to [PersistentFeature](#), because only subclasses of this type are (by policy) allowed to set this variable to true.

See also:

[PersistentFeature](#)

Definition at line 187 of file [Feature.h](#).

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

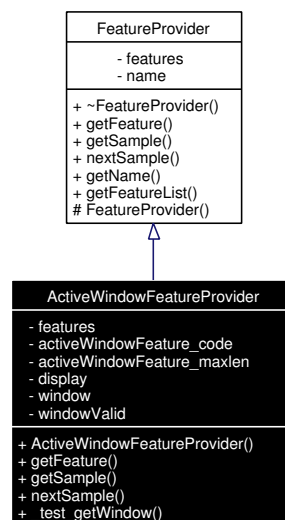
- [ActiveWindow.h](#)
- [ActiveWindow.cpp](#)

## 6.5 ActiveWindowFeatureProvider Class Reference

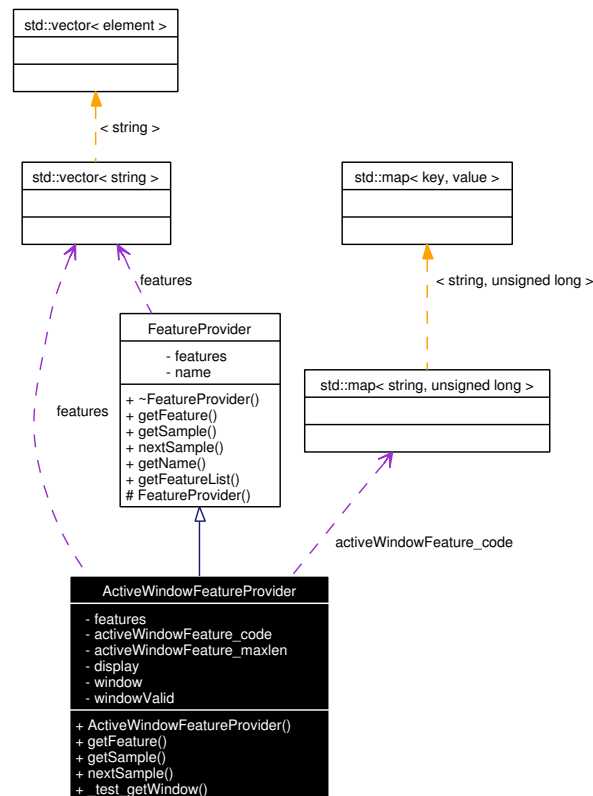
Provider for the [ActiveWindowFeature](#).

```
#include <ActiveWindow.h>
```

Inheritance diagram for ActiveWindowFeatureProvider:



Collaboration diagram for ActiveWindowFeatureProvider:



## Public Member Functions

- [ActiveWindowFeatureProvider](#) ([providerparams](#) &params)

*Constructor.*

- virtual [Feature](#) \* [getFeature](#) (string [name](#)) const

*Query feature instance.*

- virtual [Feature](#) \* [getSample](#) (string [name](#)) const

*Query sample instance.*

- virtual void [nextSample](#) (clock\_t checkpoint)

*Prepare sample.*

- string [\\_test\\_getWindow](#) ()

- string [getName](#) () const

*Query provider name.*

- [stringvector](#) \* [getFeatureList](#) () const

*Query list of provided features.*

## Private Attributes

- [stringvector features](#)  
*List of features.*
- [stringcode activeWindowFeature\\_code](#)  
*Map of code values.*
- [long activeWindowFeature\\_maxlen](#)  
*Length of longest string.*
- [string display](#)  
*Name of the X display.*
- [string window](#)  
*Window name.*
- [bool windowValid](#)  
*Indicator whether a sample is valid or not.*

### 6.5.1 Detailed Description

Provider for the [ActiveWindowFeature](#).

Definition at line 86 of file ActiveWindow.h.

### 6.5.2 Constructor & Destructor Documentation

#### 6.5.2.1 [ActiveWindowFeatureProvider::ActiveWindowFeatureProvider](#) ([providerparams](#) & [params](#))

Constructor.

**Parameters:**

*params* Map of initialization parameters

Definition at line 44 of file ActiveWindow.cpp.

### 6.5.3 Member Function Documentation

#### 6.5.3.1 [Feature](#) \* [ActiveWindowFeatureProvider::getFeature](#) ([string name](#)) const [virtual]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the requested feature.

Implements [FeatureProvider](#).

Definition at line 58 of file `ActiveWindow.cpp`.

**6.5.3.2 `stringvector* FeatureProvider::getFeatureList () const` [inline, inherited]**

Query list of provided features.

**Returns:**

A list of feature names provided by that feature provider.

Definition at line 497 of file `Feature.h`.

Referenced by `FeatureContainer::loadFeature()`.

**6.5.3.3 `string FeatureProvider::getName () const` [inline, inherited]**

Query provider name.

**Returns:**

A global unique name for the feature provider.

Definition at line 487 of file `Feature.h`.

**6.5.3.4 `Feature * ActiveWindowFeatureProvider::getSample (string name) const` [virtual]**

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the sample.

**See also:**

[nextSample](#)

Implements [FeatureProvider](#).

Definition at line 66 of file `ActiveWindow.cpp`.

**6.5.3.5 `void ActiveWindowFeatureProvider::nextSample (clock_t checkpoint)` [virtual]**

Prepare sample.

Prepare the feature provider to return the next sample from the sensor when [getSample\(\)](#) is called.

**Parameters:**

*checkpoint* This parameter tells the feature extractor that it must not return sensordata sampled earlier than the given clock value.

**See also:**

[getSample](#)

Implements [FeatureProvider](#).

Definition at line 27 of file ActiveWindowLinux.cpp.

References [display](#), [window](#), and [windowValid](#).

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

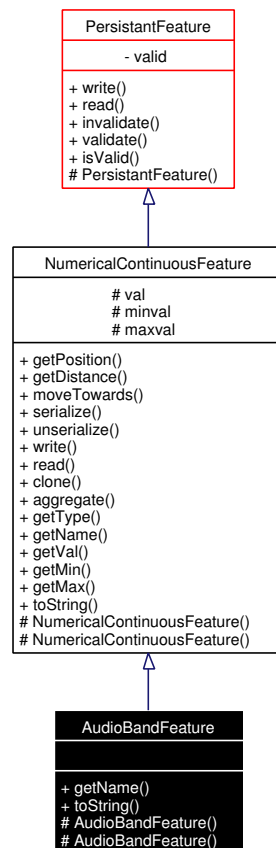
- [ActiveWindow.h](#)
- [ActiveWindow.cpp](#)
- [ActiveWindowLinux.cpp](#)
- [ActiveWindowWindows.cpp](#)

## 6.6 AudioBandFeature Class Reference

Represents a subband.

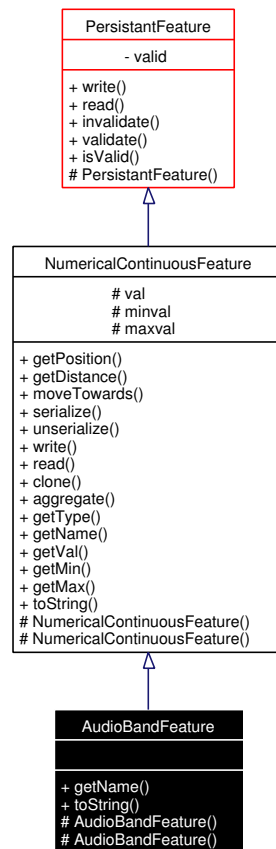
```
#include <Audio.h>
```

Inheritance diagram for AudioBandFeature:



Collaboration diagram for AudioBandFeature:





## Public Types

- enum `FeatureType` {  
**boolean, nominal, ordinal, numerical\_discrete,**  
**numerical\_continuous** }  
*Possible feature types.*

## Public Member Functions

- virtual const string `getName` () const  
*Query a features name.*
- virtual string `toString` () const  
*This is only for testing.*
- virtual double `getPosition` () const  
*Query a features position.*
- virtual double `getDistance` (Feature \*f) const  
*Calculates the distance between two features.*

- virtual void [moveTowards](#) ([Feature](#) \*f, double factor)

*Move feature.*

- virtual string [serialize](#) () const

*Serialize a samples data to a string.*

- virtual void [unserialize](#) (string value)

*Unserialize a samples data from a string.*

- virtual [featureparams](#) [write](#) () const

*Externalize feature.*

- virtual void [read](#) ([featureparams](#) \*param)

*Load feature from persistant data.*

- virtual [Feature](#) \* [clone](#) () const

*Clone a feature.*

- virtual void [aggregate](#) ([aggregatelist](#) samples)

*Aggregate a sample values from other sources.*

- virtual [FeatureType](#) [getType](#) () const

*Query a features type.*

- const double [getVal](#) () const

- const double [getMin](#) () const

- const double [getMax](#) () const

- void [invalidate](#) ()

*Invalidate feature.*

- void [validate](#) ()

*Set the validation flag to true.*

- bool [isValid](#) ()

*Query validation flag.*

- bool [isExternalizable](#) ()

*Query externalization flag.*

## Protected Member Functions

- [AudioBandFeature](#) (double \*minval, double \*maxval)

*Feature constructor*

- [AudioBandFeature](#) (double \*minval, double \*maxval, double val)

*Sample constructor.*

## Protected Attributes

- double [val](#)  
*The feature value.*
- double \* [minval](#)  
*A reference to the static, persistent minimum value seen so far.*
- double \* [maxval](#)  
*A reference to the static, persistent maximum value seen so far.*
- const bool [externalize](#)  
*Externalization flag.*

### 6.6.1 Detailed Description

Represents a subband.

This feature represents a subband of the frequency spectrum. The calculation is performed using the FFT algorithm. The feature has to be referenced as "Audio.Band.[n]" where n is a number in the interval [0; bands]. The maximum number of bands is a global variable defined in the [AudioFeatureProvider](#).

Definition at line 39 of file Audio.h.

### 6.6.2 Constructor & Destructor Documentation

#### 6.6.2.1 AudioBandFeature::AudioBandFeature (double \* *minval*, double \* *maxval*) [inline, protected]

Feature constructor

This constructor initializes the feature with a random value and should thus only be used for creating e.g. prototypes of points in a clustering space. For creating a specific sample of a feature, the second constructor should be used.

##### Parameters:

***minval*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the minimum value.

***maxval*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the maximum value.

Definition at line 59 of file Audio.h.

#### 6.6.2.2 AudioBandFeature::AudioBandFeature (double \* *minval*, double \* *maxval*, double *val*) [inline, protected]

Sample constructor.

This constructor initializes the feature with a specific value and should thus be used for samples.

**Parameters:**

*minval* A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the minimum value.

*maxval* A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum value.

*val* The sample value.

Definition at line 75 of file Audio.h.

### 6.6.3 Member Function Documentation

#### 6.6.3.1 `void NumericalContinuousFeature::aggregate (aggregatelist samples) [virtual, inherited]`

Aggregate a sample values from other sources.

**Parameters:**

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 309 of file Numerical.cpp.

References [aggregatelist](#).

#### 6.6.3.2 `Feature * NumericalContinuousFeature::clone () const [virtual, inherited]`

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Definition at line 304 of file Numerical.cpp.

References [NumericalContinuousFeature::maxval](#), [NumericalContinuousFeature::minval](#), [NumericalContinuousFeature::NumericalContinuousFeature\(\)](#), and [NumericalContinuousFeature::val](#).

#### 6.6.3.3 `double NumericalContinuousFeature::getDistance (Feature * f) const [virtual, inherited]`

Calculates the distance between two features.

**Parameters:**

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Definition at line 220 of file Numerical.cpp.

References [NumericalContinuousFeature::getPosition\(\)](#), [NumericalContinuousFeature::maxval](#), and [NumericalContinuousFeature::minval](#).

**6.6.3.4 virtual const string AudioBandFeature::getName () const** [inline, virtual]

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Reimplemented from [NumericalContinuousFeature](#).

Definition at line 78 of file Audio.h.

**6.6.3.5 double NumericalContinuousFeature::getPosition () const** [virtual, inherited]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Definition at line 204 of file Numerical.cpp.

References [NumericalContinuousFeature::maxval](#), [NumericalContinuousFeature::minval](#), and [NumericalContinuousFeature::val](#).

Referenced by [NumericalContinuousFeature::getDistance\(\)](#).

**6.6.3.6 virtual FeatureType NumericalContinuousFeature::getType () const** [inline, virtual, inherited]

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 183 of file Numerical.h.

**6.6.3.7 const double NumericalContinuousFeature::getVal () const** [inline, inherited]**Returns:**

The value of the feature.

Definition at line 187 of file Numerical.h.

Referenced by [getProvider\(\)](#), [Java\\_at\\_jku\\_intelligence\\_samples\\_NumericalContinuousSample\\_nativeGetVal\(\)](#), [WlanActiveSignalLevelFeature::toString\(\)](#), and [NumericalContinuousFeature::toString\(\)](#).

#### 6.6.3.8 void PersistentFeature::invalidate () [inline, inherited]

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by AbstractStringFeature::getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature::NumericalDiscreteFeature(), TimeFeature::TimeFeature(), NumericalContinuousFeature::unserialize(), and NumericalDiscreteFeature::unserialize().

#### 6.6.3.9 bool Feature::isExternalizable () [inline, inherited]

Query externalization flag.

##### Returns:

*true* if the feature is externalizable ( i.e. it has persistent data that should be stored across restarts),  
*false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

#### 6.6.3.10 bool PersistentFeature::isValid () [inline, inherited]

Query validation flag.

##### Returns:

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

#### 6.6.3.11 void NumericalContinuousFeature::moveTowards (Feature \* *f*, double *factor*) [virtual, inherited]

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

##### Parameters:

*f* Feature used for distance measurement.

*factor* Distance weight.

Implements Feature.

Definition at line 227 of file Numerical.cpp.

References NumericalContinuousFeature::maxval, NumericalContinuousFeature::minval, and NumericalContinuousFeature::val.

**6.6.3.12 void NumericalContinuousFeature::read (featureparams \* *param*)** [virtual, inherited]

Load feature from persistant data.

Initializes the persistant feature data from the given representation.

**Parameters:**

*param* Persistant feature data.

**See also:**

[write](#)

Implements [PersistantFeature](#).

Definition at line 291 of file Numerical.cpp.

References [featureparams](#), [NumericalContinuousFeature::maxval](#), and [NumericalContinuousFeature::minval](#).

**6.6.3.13 string NumericalContinuousFeature::serialize () const** [virtual, inherited]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [Feature](#).

Definition at line 246 of file Numerical.cpp.

References [NumericalContinuousFeature::val](#).

**6.6.3.14 void NumericalContinuousFeature::unserialize (string *value*)** [virtual, inherited]

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Definition at line 254 of file Numerical.cpp.

References [PersistantFeature::invalidate\(\)](#), [NumericalContinuousFeature::maxval](#), [NumericalContinuousFeature::minval](#), and [NumericalContinuousFeature::val](#).

**6.6.3.15 featureparams NumericalContinuousFeature::write () const** [virtual, inherited]

Externalize feature.

**Returns:**

Persistant feature data.

**See also:**[read](#)

Implements [PersistantFeature](#).

Definition at line 277 of file Numerical.cpp.

References [featureparams](#), [NumericalContinuousFeature::maxval](#), and [NumericalContinuousFeature::minval](#).

## 6.6.4 Member Data Documentation

### 6.6.4.1 `const bool Feature::externalize` [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistant features, the object should be cast to `PersistentFeature`, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**[PersistantFeature](#)

Definition at line 187 of file Feature.h.

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

- [Audio.h](#)
- [Audio.cpp](#)

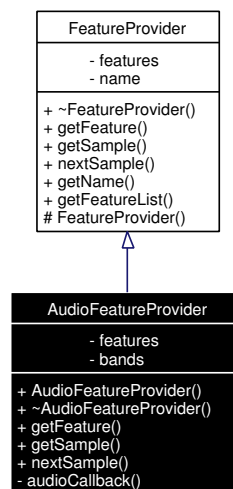


## 6.7 AudioFeatureProvider Class Reference

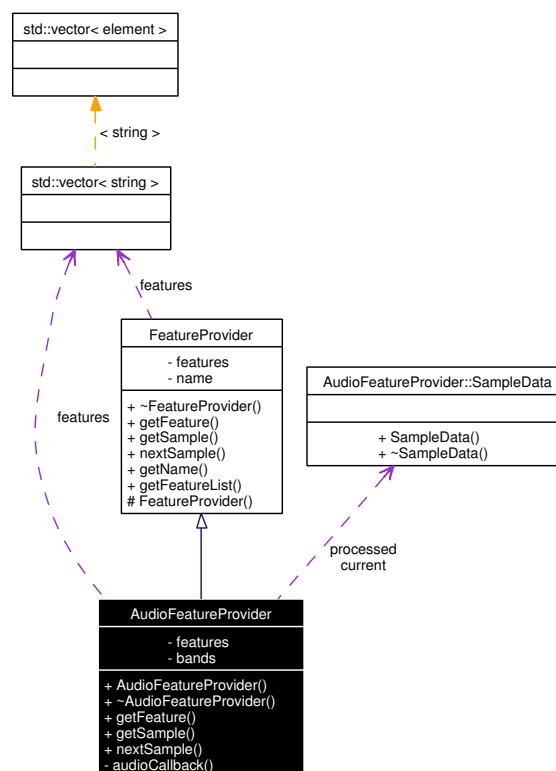
Provider for the audio features.

```
#include <Audio.h>
```

Inheritance diagram for AudioFeatureProvider:



Collaboration diagram for AudioFeatureProvider:



## Public Member Functions

- [AudioFeatureProvider](#) ([providerparams](#) &params)
- virtual [Feature](#) \* [getFeature](#) (string [name](#)) const  
*Query feature instance.*
- virtual [Feature](#) \* [getSample](#) (string [name](#)) const  
*Query sample instance.*
- virtual void [nextSample](#) (clock\_t checkpoint)  
*Prepare sample.*
- string [getName](#) () const  
*Query provider name.*
- [stringvector](#) \* [getFeatureList](#) () const  
*Query list of provided features.*

## Static Private Member Functions

- int [audioCallback](#) (void \*inputBuffer, void \*outputBuffer, unsigned long framesPerBuffer, Pa-Timestamp outTime, void \*userData)  
*Portaudio callback function.*

## Private Attributes

- [stringvector](#) [features](#)  
*List of features.*
- unsigned [bands](#)  
*Number of subbands to calculate.*
- double [audioBandFeature\\_minval](#)  
*Feature properties.*
- double [audioBandFeature\\_maxval](#)  
*Feature properties.*
- double [audioMeanFeature\\_minval](#)  
*Feature properties.*
- double [audioMeanFeature\\_maxval](#)  
*Feature properties.*
- long [audioPeaksFeature\\_minval](#)  
*Feature properties.*

- long [audioPeaksFeature\\_maxval](#)  
*Feature properties.*
- PaDeviceID [device](#)  
*Feature properties.*
- PortAudioStream \* [stream](#)  
*Feature properties.*
- [SampleData](#) [current](#)  
*Feature properties.*
- [SampleData](#) [processed](#)  
*Feature properties.*

### 6.7.1 Detailed Description

Provider for the audio features.

Definition at line 196 of file Audio.h.

### 6.7.2 Constructor & Destructor Documentation

#### 6.7.2.1 AudioFeatureProvider::AudioFeatureProvider ([providerparams](#) & [params](#))

Definition at line 63 of file Audio.cpp.

### 6.7.3 Member Function Documentation

#### 6.7.3.1 [Feature](#) \* AudioFeatureProvider::getFeature (string [name](#)) const [virtual]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

##### Parameters:

***name*** Name of the requested feature.

##### Returns:

An instance of the requested feature.

Implements [FeatureProvider](#).

Definition at line 171 of file Audio.cpp.

### 6.7.3.2 [stringvector](#)\* FeatureProvider::getFeatureList () const [inline, inherited]

Query list of provided features.

#### Returns:

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by FeatureContainer::loadFeature().

### 6.7.3.3 [string](#) FeatureProvider::getName () const [inline, inherited]

Query provider name.

#### Returns:

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

### 6.7.3.4 [Feature](#) \* AudioFeatureProvider::getSample (string *name*) const [virtual]

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

#### Parameters:

*name* Name of the requested feature.

#### Returns:

An instance of the sample.

#### See also:

[nextSample](#)

Implements [FeatureProvider](#).

Definition at line 188 of file Audio.cpp.

References [audioBandFeature\\_maxval](#), [audioBandFeature\\_minval](#), [processed](#), and [AudioFeatureProvider::SampleData::spectrum](#).

### 6.7.3.5 void AudioFeatureProvider::nextSample (clock\_t *checkpoint*) [virtual]

Prepare sample.

Prepare the feature provider to return the next sample from the sensor when [getSample\(\)](#) is called.

#### Parameters:

*checkpoint* This parameter tells the feature extractor that it must not return sensordata sampled earlier than the given clock value.

See also:

[getSample](#)

Implements [FeatureProvider](#).

Definition at line 209 of file Audio.cpp.

References [current](#), and [processed](#).

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

- [Audio.h](#)
- [Audio.cpp](#)

## 6.8 AudioFeatureProvider::SampleData Struct Reference

### Public Member Functions

- [SampleData](#) (unsigned [bands](#)=0)  
*Constructor.*
- [~SampleData](#) ()  
*Destructor.*

### Public Attributes

- double \* [spectrum](#)  
*Sample property.*
- unsigned long [spectrum\\_counter](#)  
*Sample property.*
- double [average](#)  
*Sample property.*
- unsigned long [average\\_counter](#)  
*Sample property.*
- double [average2](#)  
*Sample property.*
- double [average2\\_old](#)  
*Sample property.*
- unsigned long [average2\\_counter](#)  
*Sample property.*
- double [treshold](#)  
*Sample property.*
- unsigned long [peaks](#)  
*Sample property.*
- bool [peak](#)  
*Sample property.*

### 6.8.1 Detailed Description

#### [Todo](#)

documentation

Definition at line 202 of file Audio.h.

## 6.8.2 Constructor & Destructor Documentation

### 6.8.2.1 AudioFeatureProvider::SampleData::SampleData (unsigned *bands* = 0) [inline]

Constructor.

**Parameters:**

*bands* Number of subbands

Definition at line 226 of file Audio.h.

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

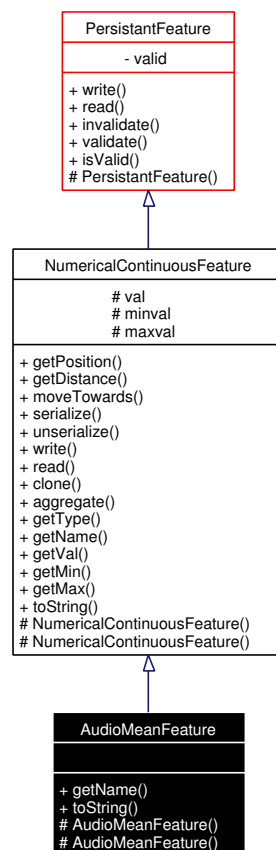
- [Audio.h](#)

## 6.9 AudioMeanFeature Class Reference

Calculates the mean level of the input signal.

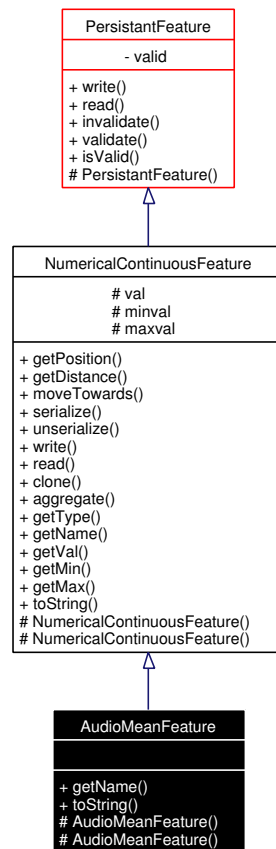
```
#include <Audio.h>
```

Inheritance diagram for AudioMeanFeature:



Collaboration diagram for AudioMeanFeature:





## Public Types

- enum `FeatureType` {  
**boolean, nominal, ordinal, numerical\_discrete,**  
**numerical\_continuous** }  
*Possible feature types.*

## Public Member Functions

- virtual const string `getName ()` const  
*Query a features name.*
- virtual string `toString ()` const  
*This is only for testing.*
- virtual double `getPosition ()` const  
*Query a features position.*
- virtual double `getDistance (Feature *f)` const  
*Calculates the distance between two features.*

- virtual void [moveTowards](#) ([Feature](#) \*f, double factor)

*Move feature.*

- virtual string [serialize](#) () const

*Serialize a samples data to a string.*

- virtual void [unserialize](#) (string value)

*Unserialize a samples data from a string.*

- virtual [featureparams](#) [write](#) () const

*Externalize feature.*

- virtual void [read](#) ([featureparams](#) \*param)

*Load feature from persistant data.*

- virtual [Feature](#) \* [clone](#) () const

*Clone a feature.*

- virtual void [aggregate](#) ([aggregatelist](#) samples)

*Aggregate a sample values from other sources.*

- virtual [FeatureType](#) [getType](#) () const

*Query a features type.*

- const double [getVal](#) () const

- const double [getMin](#) () const

- const double [getMax](#) () const

- void [invalidate](#) ()

*Invalidate feature.*

- void [validate](#) ()

*Set the validation flag to true.*

- bool [isValid](#) ()

*Query validation flag.*

- bool [isExternalizable](#) ()

*Query externalization flag.*

## Protected Member Functions

- [AudioMeanFeature](#) (double \*[minval](#), double \*[maxval](#))

*Feature constructor*

- [AudioMeanFeature](#) (double \*[minval](#), double \*[maxval](#), double mean)

*Sample constructor.*

## Protected Attributes

- double `val`  
*The feature value.*
- double \* `minval`  
*A reference to the static, persistent minimum value seen so far.*
- double \* `maxval`  
*A reference to the static, persistent maximum value seen so far.*
- const bool `externalize`  
*Externalization flag.*

### 6.9.1 Detailed Description

Calculates the mean level of the input signal.

This feature extracts the mean signal level out of the input signal. After each call to `nextSample` the time window taken for calculating the mean value is cleared.

Definition at line 93 of file `Audio.h`.

### 6.9.2 Constructor & Destructor Documentation

#### 6.9.2.1 `AudioMeanFeature::AudioMeanFeature (double * minval, double * maxval)` [`inline`, `protected`]

Feature constructor

This constructor initializes the feature with a random value and should thus only be used for creating e.g. prototypes of points in a clustering space. For creating a specific sample of a feature, the second constructor should be used.

##### Parameters:

***minval*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the minimum value.

***maxval*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the maximum value.

Definition at line 113 of file `Audio.h`.

#### 6.9.2.2 `AudioMeanFeature::AudioMeanFeature (double * minval, double * maxval, double mean)` [`inline`, `protected`]

Sample constructor.

This constructor initializes the feature with a specific value and should thus be used for samples.

##### Parameters:

***minval*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the minimum value.

*maxval* A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the maximum value.

*mean* The sample value.

Definition at line 129 of file Audio.h.

## 6.9.3 Member Function Documentation

### 6.9.3.1 `void NumericalContinuousFeature::aggregate (aggregatelist samples) [virtual, inherited]`

Aggregate a sample values from other sources.

#### Parameters:

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 309 of file Numerical.cpp.

References [aggregatelist](#).

### 6.9.3.2 `Feature * NumericalContinuousFeature::clone () const [virtual, inherited]`

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Definition at line 304 of file Numerical.cpp.

References [NumericalContinuousFeature::maxval](#), [NumericalContinuousFeature::minval](#), [NumericalContinuousFeature::NumericalContinuousFeature\(\)](#), and [NumericalContinuousFeature::val](#).

### 6.9.3.3 `double NumericalContinuousFeature::getDistance (Feature *f) const [virtual, inherited]`

Calculates the distance between two features.

#### Parameters:

*f* [Feature](#) used for distance measurement.

#### Returns:

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Definition at line 220 of file Numerical.cpp.

References [NumericalContinuousFeature::getPosition\(\)](#), [NumericalContinuousFeature::maxval](#), and [NumericalContinuousFeature::minval](#).

**6.9.3.4 virtual const string AudioMeanFeature::getName () const** [inline, virtual]

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Reimplemented from [NumericalContinuousFeature](#).

Definition at line 132 of file Audio.h.

**6.9.3.5 double NumericalContinuousFeature::getPosition () const** [virtual, inherited]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Definition at line 204 of file Numerical.cpp.

References [NumericalContinuousFeature::maxval](#), [NumericalContinuousFeature::minval](#), and [NumericalContinuousFeature::val](#).

Referenced by [NumericalContinuousFeature::getDistance\(\)](#).

**6.9.3.6 virtual [FeatureType](#) NumericalContinuousFeature::getType () const** [inline, virtual, inherited]

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 183 of file Numerical.h.

**6.9.3.7 const double NumericalContinuousFeature::getVal () const** [inline, inherited]**Returns:**

The value of the feature.

Definition at line 187 of file Numerical.h.

Referenced by [getProvider\(\)](#), [Java\\_at\\_jku\\_intelligence\\_samples\\_NumericalContinuousSample\\_nativeGetVal\(\)](#), [WlanActiveSignalLevelFeature::toString\(\)](#), and [NumericalContinuousFeature::toString\(\)](#).

#### 6.9.3.8 void PersistentFeature::invalidate () [inline, inherited]

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by AbstractStringFeature::getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature::NumericalDiscreteFeature(), TimeFeature::TimeFeature(), NumericalContinuousFeature::unserialize(), and NumericalDiscreteFeature::unserialize().

#### 6.9.3.9 bool Feature::isExternalizable () [inline, inherited]

Query externalization flag.

##### Returns:

*true* if the feature is externalizable ( i.e. it has persistent data that should be stored across restarts),  
*false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

#### 6.9.3.10 bool PersistentFeature::isValid () [inline, inherited]

Query validation flag.

##### Returns:

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

#### 6.9.3.11 void NumericalContinuousFeature::moveTowards (Feature \* *f*, double *factor*) [virtual, inherited]

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

##### Parameters:

*f* Feature used for distance measurement.

*factor* Distance weight.

Implements Feature.

Definition at line 227 of file Numerical.cpp.

References NumericalContinuousFeature::maxval, NumericalContinuousFeature::minval, and NumericalContinuousFeature::val.

**6.9.3.12 void NumericalContinuousFeature::read (featureparams \* param) [virtual, inherited]**

Load feature from persistant data.

Initializes the persistant feature data from the given representation.

**Parameters:**

*param* Persistant feature data.

**See also:**

[write](#)

Implements [PersistantFeature](#).

Definition at line 291 of file Numerical.cpp.

References [featureparams](#), [NumericalContinuousFeature::maxval](#), and [NumericalContinuousFeature::minval](#).

**6.9.3.13 string NumericalContinuousFeature::serialize () const [virtual, inherited]**

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [Feature](#).

Definition at line 246 of file Numerical.cpp.

References [NumericalContinuousFeature::val](#).

**6.9.3.14 void NumericalContinuousFeature::unserialize (string value) [virtual, inherited]**

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Definition at line 254 of file Numerical.cpp.

References [PersistantFeature::invalidate\(\)](#), [NumericalContinuousFeature::maxval](#), [NumericalContinuousFeature::minval](#), and [NumericalContinuousFeature::val](#).

**6.9.3.15 featureparams NumericalContinuousFeature::write () const [virtual, inherited]**

Externalize feature.

**Returns:**

Persistant feature data.

**See also:**[read](#)

Implements [PersistantFeature](#).

Definition at line 277 of file Numerical.cpp.

References [featureparams](#), [NumericalContinuousFeature::maxval](#), and [NumericalContinuousFeature::minval](#).

## 6.9.4 Member Data Documentation

### 6.9.4.1 `const bool Feature::externalize` [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistant features, the object should be cast to `PersistentFeature`, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**[PersistantFeature](#)

Definition at line 187 of file Feature.h.

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

- [Audio.h](#)
- [Audio.cpp](#)

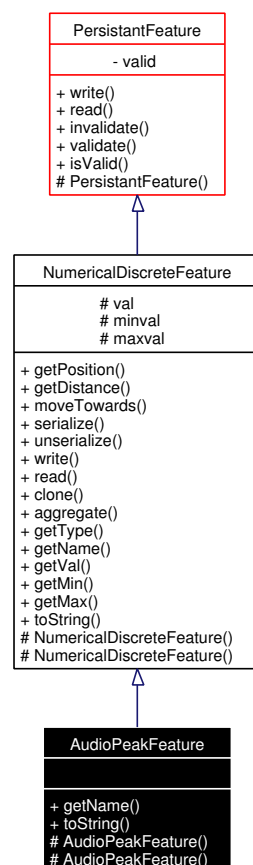


## 6.10 AudioPeakFeature Class Reference

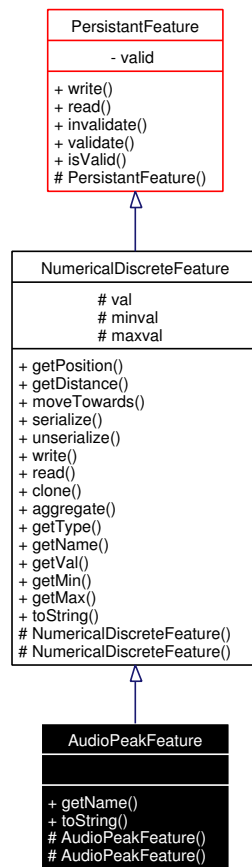
Counts the peaks in the input signal.

```
#include <Audio.h>
```

Inheritance diagram for AudioPeakFeature:



Collaboration diagram for AudioPeakFeature:



## Public Types

- enum `FeatureType` {  
**boolean, nominal, ordinal, numerical\_discrete,**  
**numerical\_continuous** }  
*Possible feature types.*

## Public Member Functions

- virtual const string `getName ()` const  
*Query a features name.*
- virtual string `toString ()` const  
*This is only for testing.*
- virtual double `getPosition ()` const  
*Query a features position.*
- virtual double `getDistance (Feature *f)` const  
*Calculates the distance between two features.*

- virtual void [moveTowards](#) ([Feature](#) \*f, double factor)  
*Move feature.*
- virtual string [serialize](#) () const  
*Serialize a samples data to a string.*
- virtual void [unserialize](#) (string value)  
*Unserialize a samples data from a string.*
- virtual [featureparams](#) [write](#) () const  
*Externalize feature.*
- virtual void [read](#) ([featureparams](#) \*param)  
*Load feature from persistant data.*
- virtual [Feature](#) \* [clone](#) () const  
*Clone a feature.*
- virtual void [aggregate](#) ([aggregatelist](#) samples)  
*Aggregate a sample values from other sources.*
- virtual [FeatureType](#) [getType](#) () const  
*Query a features type.*
- const long [getVal](#) () const
- const long [getMin](#) () const
- const long [getMax](#) () const
- void [invalidate](#) ()  
*Invalidate feature.*
- void [validate](#) ()  
*Set the validation flag to true.*
- bool [isValid](#) ()  
*Query validation flag.*
- bool [isExternalizable](#) ()  
*Query externalization flag.*

## Protected Member Functions

- [AudioPeakFeature](#) (long \*minval, long \*maxval)  
*Feature constructor*
- [AudioPeakFeature](#) (long \*minval, long \*maxval, unsigned peaks)  
*Sample constructor.*

## Protected Attributes

- double `val`  
*The feature value.*
- long \* `minval`  
*A reference to the static, persistent minimum value seen so far.*
- long \* `maxval`  
*A reference to the static, persistent maximum value seen so far.*
- const bool `externalize`  
*Externalization flag.*

### 6.10.1 Detailed Description

Counts the peaks in the input signal.

This feature counts the peaks in the input signal. If there are high peaks the extractor becomes less sensible to lower peaks, if there are only a few low peaks the extractor becomes more sensible to them. When `nextSample` is called, the peak counter is reset to 0.

Definition at line 147 of file `Audio.h`.

### 6.10.2 Constructor & Destructor Documentation

#### 6.10.2.1 `AudioPeakFeature::AudioPeakFeature (long * minval, long * maxval)` [`inline`, `protected`]

Feature constructor

This constructor initializes the feature with a random value and should thus only be used for creating e.g. prototypes of points in a clustering space. For creating a specific sample of a feature, the second constructor should be used.

##### Parameters:

- minval*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the minimum value.
- maxval*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the maximum value.

Definition at line 167 of file `Audio.h`.

#### 6.10.2.2 `AudioPeakFeature::AudioPeakFeature (long * minval, long * maxval, unsigned peaks)` [`inline`, `protected`]

Sample constructor.

This constructor initializes the feature with a specific value and should thus be used for samples.

**Parameters:**

*minval* A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the minimum value.

*maxval* A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum value.

*peaks* The sample value.

Definition at line 183 of file Audio.h.

### 6.10.3 Member Function Documentation

#### 6.10.3.1 `void NumericalDiscreteFeature::aggregate (aggregatelist samples) [virtual, inherited]`

Aggregate a sample values from other sources.

**Parameters:**

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 159 of file Numerical.cpp.

References [aggregatelist](#).

#### 6.10.3.2 `Feature * NumericalDiscreteFeature::clone () const [virtual, inherited]`

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Definition at line 154 of file Numerical.cpp.

References [NumericalDiscreteFeature::maxval](#), [NumericalDiscreteFeature::minval](#), [NumericalDiscreteFeature::NumericalDiscreteFeature\(\)](#), and [NumericalDiscreteFeature::val](#).

#### 6.10.3.3 `double NumericalDiscreteFeature::getDistance (Feature * f) const [virtual, inherited]`

Calculates the distance between two features.

**Parameters:**

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Definition at line 73 of file Numerical.cpp.

References [NumericalDiscreteFeature::getPosition\(\)](#), [NumericalDiscreteFeature::maxval](#), and [NumericalDiscreteFeature::minval](#).

**6.10.3.4 virtual const string AudioPeakFeature::getName () const** [inline, virtual]

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Reimplemented from [NumericalDiscreteFeature](#).

Definition at line 186 of file Audio.h.

**6.10.3.5 double NumericalDiscreteFeature::getPosition () const** [virtual, inherited]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Definition at line 57 of file Numerical.cpp.

References [NumericalDiscreteFeature::maxval](#), [NumericalDiscreteFeature::minval](#), and [NumericalDiscreteFeature::val](#).

Referenced by [NumericalDiscreteFeature::getDistance\(\)](#).

**6.10.3.6 virtual [FeatureType](#) NumericalDiscreteFeature::getType () const** [inline, virtual, inherited]

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 103 of file Numerical.h.

**6.10.3.7 const long NumericalDiscreteFeature::getVal () const** [inline, inherited]**Returns:**

The value of the feature.

Definition at line 107 of file Numerical.h.

References [NumericalDiscreteFeature::val](#).

Referenced by [Java\\_at\\_jku\\_intelligence\\_samples\\_NumericalDiscreteSample\\_nativeGetVal\(\)](#), [WlanNumPeersFeature::toString\(\)](#), [NumericalDiscreteFeature::toString\(\)](#), and [BluetoothNumPeersFeature::toString\(\)](#).

**6.10.3.8 void PersistentFeature::invalidate ()** [inline, inherited]

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by AbstractStringFeature::getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature::NumericalDiscreteFeature(), TimeFeature::TimeFeature(), NumericalContinuousFeature::unserialize(), and NumericalDiscreteFeature::unserialize().

**6.10.3.9 bool Feature::isExternalizable ()** [inline, inherited]

Query externalization flag.

**Returns:**

*true* if the feature is externalizable ( i.e. it has persistent data that should be stored across restarts), *false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.10.3.10 bool PersistentFeature::isValid ()** [inline, inherited]

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

**6.10.3.11 void NumericalDiscreteFeature::moveTowards (Feature \**f*, double *factor*)** [virtual, inherited]

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

**Parameters:**

*f* Feature used for distance measurement.

*factor* Distance weight.

Implements Feature.

Definition at line 80 of file Numerical.cpp.

References NumericalDiscreteFeature::maxval, NumericalDiscreteFeature::minval, and NumericalDiscreteFeature::val.

**6.10.3.12** `void NumericalDiscreteFeature::read (featureparams * param)` [virtual, inherited]

Load feature from persistant data.

Initializes the persistant feature data from the given representation.

**Parameters:**

*param* Persistant feature data.

**See also:**

[write](#)

Implements [PersistantFeature](#).

Definition at line 141 of file Numerical.cpp.

References featureparams, NumericalDiscreteFeature::maxval, and NumericalDiscreteFeature::minval.

**6.10.3.13** `string NumericalDiscreteFeature::serialize () const` [virtual, inherited]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [Feature](#).

Definition at line 96 of file Numerical.cpp.

References NumericalDiscreteFeature::val.

**6.10.3.14** `void NumericalDiscreteFeature::unserialize (string value)` [virtual, inherited]

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Definition at line 104 of file Numerical.cpp.

References [PersistantFeature::invalidate\(\)](#), [NumericalDiscreteFeature::maxval](#), [NumericalDiscreteFeature::minval](#), and [NumericalDiscreteFeature::val](#).

**6.10.3.15** `featureparams NumericalDiscreteFeature::write () const` [virtual, inherited]

Externalize feature.

**Returns:**

Persistant feature data.

**See also:**

[read](#)



Implements [PersistantFeature](#).

Definition at line 127 of file Numerical.cpp.

References [featureparams](#), [NumericalDiscreteFeature::maxval](#), and [NumericalDiscreteFeature::minval](#).

## 6.10.4 Member Data Documentation

### 6.10.4.1 `const bool Feature::externalize` [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistant features, the object should be cast to [PersistentFeature](#), because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**

[PersistantFeature](#)

Definition at line 187 of file Feature.h.

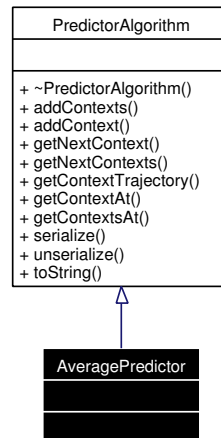
The documentation for this class was generated from the following files:

- [Audio.h](#)
- [Audio.cpp](#)

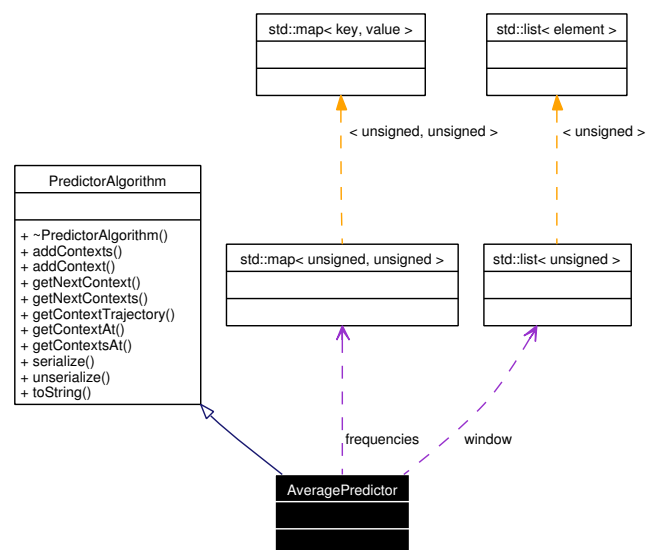
## 6.11 AveragePredictor Class Reference

```
#include <AveragePredictor.h>
```

Inheritance diagram for AveragePredictor:



Collaboration diagram for AveragePredictor:



### [NOHEADER]

- [AveragePredictor](#) ([predictorparams](#) &[params](#))
- virtual void [addContexts](#) (const [membership](#)[list](#) \*contexts, time\_t time)
- virtual void [addContext](#) (unsigned long contextId, time\_t time)
- virtual unsigned long [getNextContext](#) () const
- virtual [membership](#)[list](#) [getNextContexts](#) () const
- virtual [context](#)[trajectory](#) [getContextTrajectory](#) (unsigned int start, unsigned int end) const

- virtual unsigned long [getContextAt](#) (time\_t time) const
- virtual [membershiplist](#) [getContextsAt](#) (time\_t time) const
- virtual string [serialize](#) () const
- virtual void [unserialize](#) (string data)
- virtual string [toString](#) () const

*This is only for testing.*

- unsigned [windowSize](#)

*Private property.*

- [list](#)< unsigned > [window](#)
- [map](#)< unsigned, unsigned > [frequencies](#)

### 6.11.1 Detailed Description

**Todo**

documentation

Definition at line 32 of file AveragePredictor.h.

### 6.11.2 Constructor & Destructor Documentation

#### 6.11.2.1 AveragePredictor::AveragePredictor ([predictorparams](#) & *params*)

**Todo**

documentation

Definition at line 34 of file AveragePredictor.cpp.

References [windowSize](#).

### 6.11.3 Member Function Documentation

#### 6.11.3.1 void AveragePredictor::addContext (unsigned long *contextId*, time\_t *time*) [virtual]

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 50 of file AveragePredictor.cpp.

References [frequencies](#), [window](#), and [windowSize](#).

#### 6.11.3.2 void AveragePredictor::addContexts (const [membershiplist](#) \* *contexts*, time\_t *time*) [virtual]

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 43 of file AveragePredictor.cpp.

**6.11.3.3   unsigned long AveragePredictor::getContextAt (time\_t *time*) const   [virtual]****Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 94 of file AveragePredictor.cpp.

References membership-list.

**6.11.3.4   membership-list AveragePredictor::getContextsAt (time\_t *time*) const   [virtual]****Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 100 of file AveragePredictor.cpp.

**6.11.3.5   context-trajectory AveragePredictor::getContextTrajectory (unsigned int *start*, unsigned int *end*) const   [virtual]****Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 85 of file AveragePredictor.cpp.

**6.11.3.6   unsigned long AveragePredictor::getNextContext () const   [virtual]****Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 61 of file AveragePredictor.cpp.

**6.11.3.7   membership-list AveragePredictor::getNextContexts () const   [virtual]****Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 74 of file AveragePredictor.cpp.

**6.11.3.8   string AveragePredictor::serialize () const   [virtual]****Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 106 of file AveragePredictor.cpp.

**6.11.3.9 void AveragePredictor::unserialize (string *data*) [virtual]****Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 111 of file AveragePredictor.cpp.

References [windowSize](#).

**6.11.4 Member Data Documentation****6.11.4.1 map<unsigned, unsigned> AveragePredictor::frequencies [private]****Todo**

documentation

Definition at line 42 of file AveragePredictor.h.

Referenced by [addContext\(\)](#).

**6.11.4.2 list<unsigned> AveragePredictor::window [private]****Todo**

documentation

Definition at line 41 of file AveragePredictor.h.

Referenced by [addContext\(\)](#), and [toString\(\)](#).

**6.11.4.3 unsigned AveragePredictor::windowSize [private]**

Private property.

**Todo**

documentation

Definition at line 40 of file AveragePredictor.h.

Referenced by [addContext\(\)](#), [AveragePredictor\(\)](#), and [unserialize\(\)](#).

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

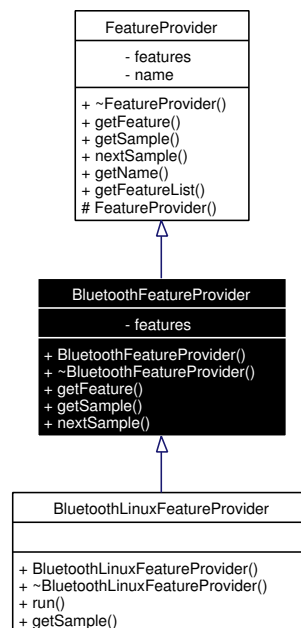
- [AveragePredictor.h](#)
- [AveragePredictor.cpp](#)

## 6.12 BluetoothFeatureProvider Class Reference

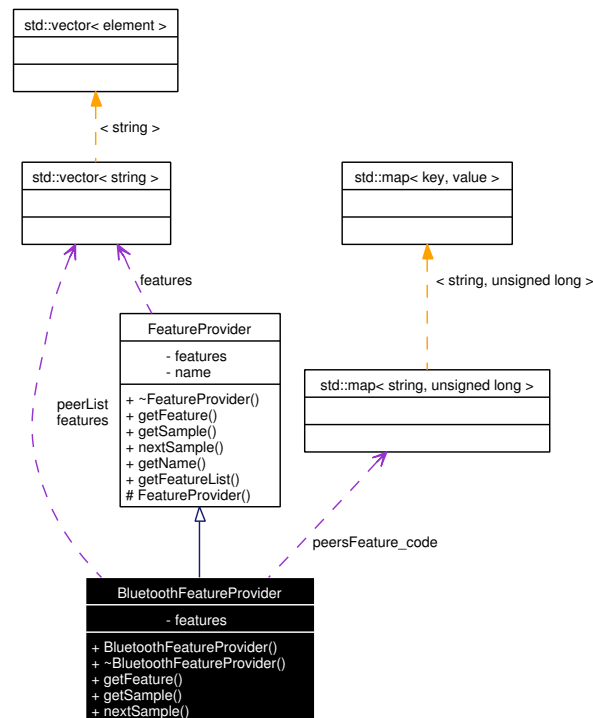
Provider for the Bluetooth features.

```
#include <Bluetooth.h>
```

Inheritance diagram for BluetoothFeatureProvider:



Collaboration diagram for BluetoothFeatureProvider:



## Public Member Functions

- [BluetoothFeatureProvider](#) ([providerparams](#) &[params](#))

*Constructor.*

- virtual [~BluetoothFeatureProvider](#) ()

*Destructor.*

- virtual [Feature](#) \* [getFeature](#) (string [name](#)) const

*Query feature instance.*

- virtual [Feature](#) \* [getSample](#) (string [name](#)) const

*Query sample instance.*

- virtual void [nextSample](#) (clock\_t checkpoint)

*Prepare sample.*

- string [getName](#) () const

*Query provider name.*

- [stringvector](#) \* [getFeatureList](#) () const

*Query list of provided features.*

## Protected Attributes

- [stringvector peerList](#)  
*Feature properties.*
- [bool peerListValid](#)  
*Feature properties.*

## Private Attributes

- [stringvector features](#)  
*List of features.*
- [stringcode peersFeature\\_code](#)  
*Feature properties.*
- [long peersFeature\\_maxlen](#)  
*Feature properties.*
- [long numPeersFeature\\_minval](#)  
*Feature properties.*
- [long numPeersFeature\\_maxval](#)  
*Feature properties.*

### 6.12.1 Detailed Description

Provider for the Bluetooth features.

Definition at line 135 of file Bluetooth.h.

### 6.12.2 Constructor & Destructor Documentation

#### 6.12.2.1 BluetoothFeatureProvider::BluetoothFeatureProvider ([providerparams](#) & *params*)

Constructor.

##### Parameters:

*params* Map of initialization parameters

Definition at line 50 of file Bluetooth.cpp.

References [features](#), [numPeersFeature\\_maxval](#), [numPeersFeature\\_minval](#), [peerListValid](#), [peersFeature\\_maxlen](#), and [providerparams](#).



### 6.12.3 Member Function Documentation

#### 6.12.3.1 **Feature** \* BluetoothFeatureProvider::getFeature (string *name*) const [virtual]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the requested feature.

Implements [FeatureProvider](#).

Definition at line 84 of file Bluetooth.cpp.

References numPeersFeature\_maxval, numPeersFeature\_minval, peersFeature\_code, and peersFeature\_maxlen.

#### 6.12.3.2 **stringvector**\* FeatureProvider::getFeatureList () const [inline, inherited]

Query list of provided features.

**Returns:**

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by FeatureContainer::loadFeature().

#### 6.12.3.3 **string** FeatureProvider::getName () const [inline, inherited]

Query provider name.

**Returns:**

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

#### 6.12.3.4 **Feature** \* BluetoothFeatureProvider::getSample (string *name*) const [virtual]

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the sample.

See also:

[nextSample](#)

Implements [FeatureProvider](#).

Reimplemented in [BluetoothLinuxFeatureProvider](#).

Definition at line 73 of file Bluetooth.cpp.

References numPeersFeature\_maxval, numPeersFeature\_minval, peerList, peerListValid, peersFeature\_code, and peersFeature\_maxlen.

#### 6.12.3.5 void BluetoothFeatureProvider::nextSample (clock\_t *checkpoint*) [virtual]

Prepare sample.

Prepare the feature provider to return the next sample from the sensor when [getSample\(\)](#) is called.

**Parameters:**

*checkpoint* This parameter tells the feature extractor that it must not return sensordata sampled earlier than the given clock value.

See also:

[getSample](#)

Implements [FeatureProvider](#).

Definition at line 92 of file BluetoothLinux.cpp.

References BLUETOOTH\_FLUSH\_CACHE, BLUETOOTH\_INQUIRYLENGTH, BLUETOOTH\_L2\_PINGDELAY, BLUETOOTH\_MAX\_PING\_TIME, BLUETOOTH\_MIN\_LINK\_QUALITY, BLUETOOTH\_SCANDelay, BLUETOOTH\_USE\_L2\_CONNECTIONS, featureparams, and peerListValid.

Referenced by Java\_at\_jku\_bluetooth\_LinuxBluetoothAdapter\_nativeGetAddressesInRange(), and Java\_at\_jku\_bluetooth\_LinuxBluetoothAdapter\_nativeIsAddressValid().

## 6.12.4 Member Data Documentation

### 6.12.4.1 long BluetoothFeatureProvider::numPeersFeature\_maxval [private]

[Feature](#) properties.

**Todo**

move to features

Definition at line 149 of file Bluetooth.h.

Referenced by BluetoothFeatureProvider(), getFeature(), and getSample().

### 6.12.4.2 long BluetoothFeatureProvider::numPeersFeature\_minval [private]

[Feature](#) properties.

**Todo**

move to features

Definition at line 148 of file Bluetooth.h.

Referenced by BluetoothFeatureProvider(), getFeature(), and getSample().

#### 6.12.4.3 [stringvector BluetoothFeatureProvider::peerList](#) [protected]

[Feature](#) properties.

##### **Todo**

move to features

Definition at line 158 of file Bluetooth.h.

Referenced by getSample().

#### 6.12.4.4 [bool BluetoothFeatureProvider::peerListValid](#) [protected]

[Feature](#) properties.

##### **Todo**

move to features

Definition at line 159 of file Bluetooth.h.

Referenced by BluetoothFeatureProvider(), getSample(), and nextSample().

#### 6.12.4.5 [stringcode BluetoothFeatureProvider::peersFeature\\_code](#) [private]

[Feature](#) properties.

##### **Todo**

move to features

Definition at line 146 of file Bluetooth.h.

Referenced by getFeature(), and getSample().

#### 6.12.4.6 [long BluetoothFeatureProvider::peersFeature\\_maxlen](#) [private]

[Feature](#) properties.

##### **Todo**

move to features

Definition at line 147 of file Bluetooth.h.

Referenced by BluetoothFeatureProvider(), getFeature(), and getSample().

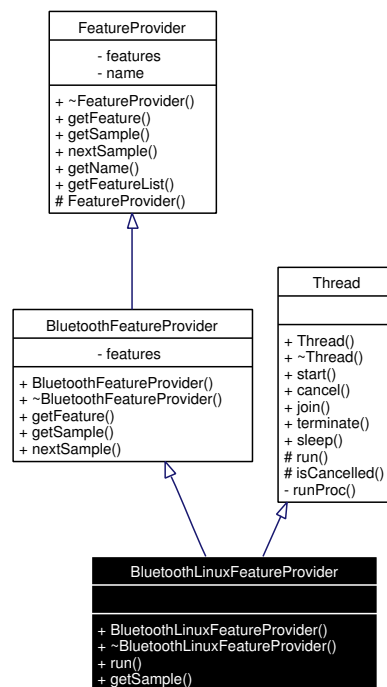
The documentation for this class was generated from the following files:

- [Bluetooth.h](#)
- [Bluetooth.cpp](#)
- [BluetoothLinux.cpp](#)
- [BluetoothWindows.cpp](#)

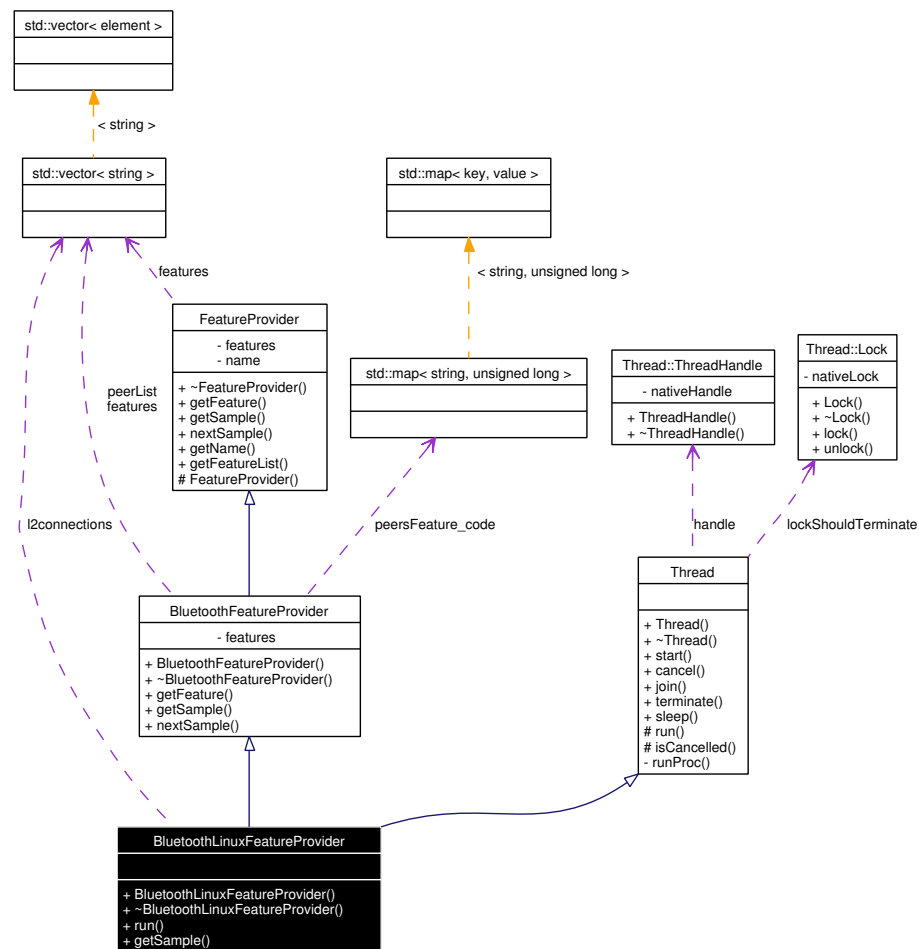
## 6.13 BluetoothLinuxFeatureProvider Class Reference

```
#include <BluetoothLinux.h>
```

Inheritance diagram for BluetoothLinuxFeatureProvider:



Collaboration diagram for BluetoothLinuxFeatureProvider:



## Public Member Functions

- [BluetoothLinuxFeatureProvider](#) ([featureparams](#) &[params](#))

*Constructor.*

- virtual [~BluetoothLinuxFeatureProvider](#) () throw ()

*Destructor.*

- virtual void [run](#) () throw ()

*Thread main worker function.*

- virtual [Feature](#) \* [getSample](#) (string [name](#)) const

*Query sample instance.*

- virtual [Feature](#) \* [getFeature](#) (string [name](#)) const

*Query feature instance.*

- virtual void [nextSample](#) (clock\_t checkpoint)

*Prepare sample.*

- string [getName](#) () const  
*Query provider name.*
- [stringvector](#) \* [getFeatureList](#) () const  
*Query list of provided features.*
- void [start](#) ()  
*Start thread execution.*
- void [cancel](#) ()  
*End thread execution.*
- void [join](#) ()  
*Join thread.*
- void [terminate](#) ()  
*Terminate thread execution.*

## Static Public Member Functions

- void [sleep](#) (unsigned milliseconds)  
*sleep function*

## Protected Member Functions

- bool [isCancelled](#) ()

## Protected Attributes

- [stringvector](#) [peerList](#)  
*Feature properties.*
- bool [peerListValid](#)  
*Feature properties.*

## Private Attributes

- unsigned int [delay](#)  
*Provider properties.*
- unsigned int [inquiryLength](#)  
*Provider properties.*
- bool [flushInquiryCache](#)

*Provider properties.*

- unsigned int [maxpingtime](#)

*Provider properties.*

- unsigned short [minlinkquality](#)

*Provider properties.*

- unsigned int [l2\\_pingdelay](#)

*Provider properties.*

- bool [use\\_l2\\_connections](#)

*Provider properties.*

- Lock [lockPeerList](#)

*Provider properties.*

- [stringvector](#) [l2connections](#)

*Provider properties.*

### 6.13.1 Detailed Description

**Todo**

documentation

Definition at line 33 of file BluetoothLinux.h.

### 6.13.2 Constructor & Destructor Documentation

#### 6.13.2.1 BluetoothLinuxFeatureProvider::BluetoothLinuxFeatureProvider ([featureparams](#) & [params](#))

Constructor.

**Parameters:**

*params* Map of initialization parameters

**Todo**

check parameter !!  
check parameter !!  
check parameter !!  
check parameter !!  
check parameter !!

Definition at line 98 of file BluetoothLinux.cpp.

### 6.13.3 Member Function Documentation

#### 6.13.3.1 **Feature** \* BluetoothFeatureProvider::getFeature (string *name*) const [virtual, inherited]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the requested feature.

Implements [FeatureProvider](#).

Definition at line 84 of file Bluetooth.cpp.

References [BluetoothFeatureProvider::numPeersFeature\\_maxval](#), [BluetoothFeatureProvider::numPeersFeature\\_minval](#), [BluetoothFeatureProvider::peersFeature\\_code](#), and [BluetoothFeatureProvider::peersFeature\\_maxlen](#).

#### 6.13.3.2 **stringvector**\* FeatureProvider::getFeatureList () const [inline, inherited]

Query list of provided features.

**Returns:**

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by [FeatureContainer::loadFeature\(\)](#).

#### 6.13.3.3 **string** FeatureProvider::getName () const [inline, inherited]

Query provider name.

**Returns:**

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

#### 6.13.3.4 **Feature** \* BluetoothLinuxFeatureProvider::getSample (string *name*) const [virtual]

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

**Parameters:**

*name* Name of the requested feature.



**Returns:**

An instance of the sample.

**See also:**

[nextSample](#)

Reimplemented from [BluetoothFeatureProvider](#).

Definition at line 219 of file BluetoothLinux.cpp.

Referenced by Java\_at\_jku\_bluetooth\_LinuxBluetoothAdapter\_nativeGetAddressesInRange(), and Java\_at\_jku\_bluetooth\_LinuxBluetoothAdapter\_nativeIsValidAddress().

**6.13.3.5 bool Thread::isCancelled () [protected, inherited]****Returns:**

*true* if the thread is cancelled, *false* if the thread is still running

Definition at line 43 of file Thread.cpp.

References Thread::cancelled.

Referenced by GSMFeatureProvider::nextSample(), WlanLinuxFeatureProvider::run(), SystemCommandStringListFeatureProvider::run(), and run().

**6.13.3.6 void BluetoothFeatureProvider::nextSample (clock\_t *checkpoint*) [virtual, inherited]**

Prepare sample.

Prepare the feature provider to return the next sample from the sensor when [getSample\(\)](#) is called.

**Parameters:**

***checkpoint*** This parameter tells the feature extractor that it must not return sensor data sampled earlier than the given clock value.

**See also:**

[getSample](#)

Implements [FeatureProvider](#).

Definition at line 92 of file BluetoothLinux.cpp.

References BLUETOOTH\_FLUSH\_CACHE, BLUETOOTH\_INQUIRYLENGTH, BLUETOOTH\_L2\_PINGDELAY, BLUETOOTH\_MAX\_PING\_TIME, BLUETOOTH\_MIN\_LINK\_QUALITY, BLUETOOTH\_SCANDelay, BLUETOOTH\_USE\_L2\_CONNECTIONS, featureparams, and BluetoothFeatureProvider::peerListValid.

Referenced by Java\_at\_jku\_bluetooth\_LinuxBluetoothAdapter\_nativeGetAddressesInRange(), and Java\_at\_jku\_bluetooth\_LinuxBluetoothAdapter\_nativeIsValidAddress().

**6.13.3.7 void Thread::sleep (unsigned *milliseconds*) [static, inherited]**

sleep function

**Parameters:**

*milliseconds* Timeout in milliseconds

Definition at line 80 of file ThreadPosix.cpp.

Referenced by `Main()`, `GSMFeatureProvider::readLine()`, `WlanLinuxFeatureProvider::run()`, `System-CommandStringListFeatureProvider::run()`, `Scanner::run()`, and `run()`.

## 6.13.4 Member Data Documentation

### 6.13.4.1 [stringvector BluetoothFeatureProvider::peerList](#) [protected, inherited]

[Feature](#) properties.

#### [Todo](#)

move to features

Definition at line 158 of file Bluetooth.h.

Referenced by `BluetoothFeatureProvider::getSample()`.

### 6.13.4.2 [bool BluetoothFeatureProvider::peerListValid](#) [protected, inherited]

[Feature](#) properties.

#### [Todo](#)

move to features

Definition at line 159 of file Bluetooth.h.

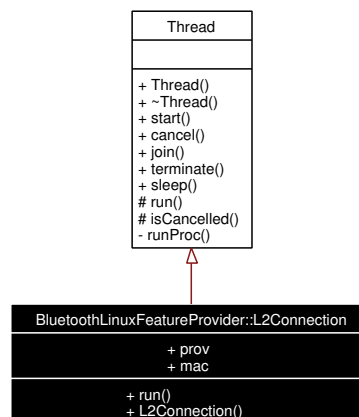
Referenced by `BluetoothFeatureProvider::BluetoothFeatureProvider()`, `BluetoothFeatureProvider::getSample()`, and `BluetoothFeatureProvider::nextSample()`.

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

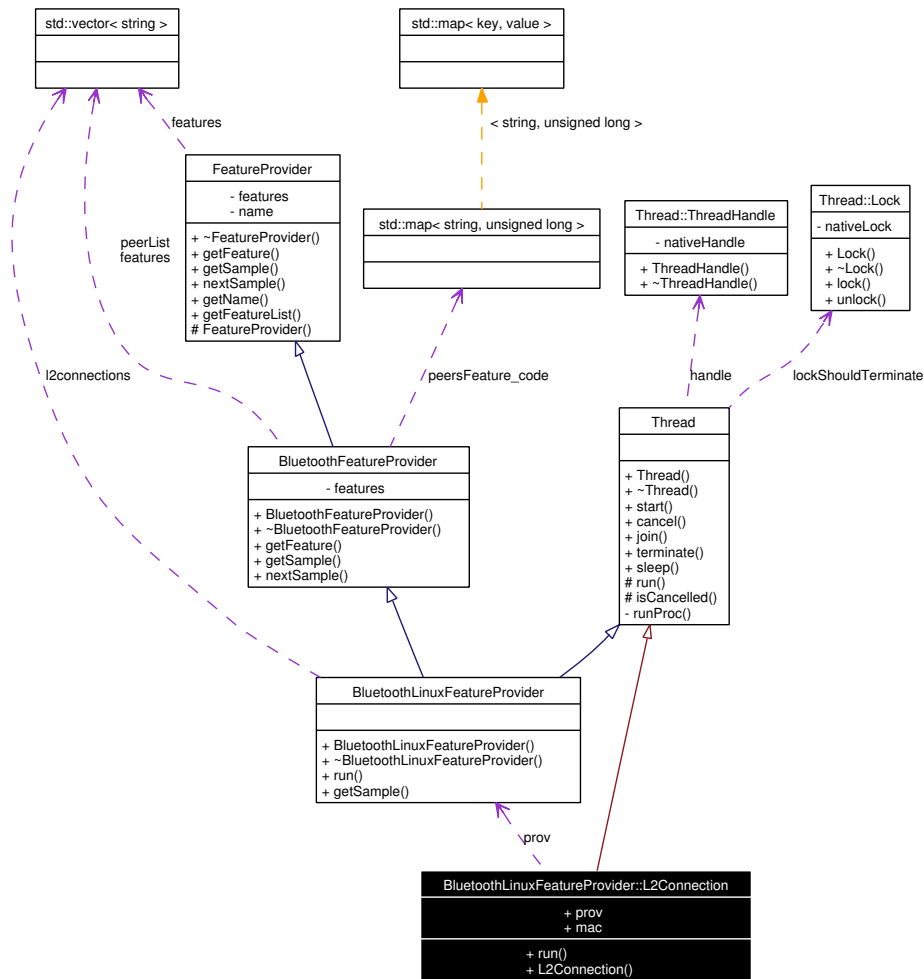
- [BluetoothLinux.h](#)
- [BluetoothLinux.cpp](#)

## 6.14 BluetoothLinuxFeatureProvider::L2Connection Class Reference

Inheritance diagram for BluetoothLinuxFeatureProvider::L2Connection:



Collaboration diagram for BluetoothLinuxFeatureProvider::L2Connection:



## Public Member Functions

- virtual void [run](#) () throw ()  
*Thread* main worker function.
- [L2Connection](#) ([BluetoothLinuxFeatureProvider](#) \*prov, string mac)  
*Constructor.*

## Public Attributes

- [BluetoothLinuxFeatureProvider](#) \* prov  
*FeatureProvider.*
- const string mac  
*Target MAC address.*

## Static Public Attributes

- const int `ident` = 200  
*magic values from Linux l2ping ....*
- const int `size` = 20  
*magic values from Linux l2ping ....*

## Private Member Functions

- void `start` ()  
*Start thread execution.*
- void `cancel` ()  
*End thread execution.*
- void `join` ()  
*Join thread.*
- void `terminate` ()  
*Terminate thread execution.*
- bool `isCancelled` ()

## Static Private Member Functions

- void `sleep` (unsigned milliseconds)  
*sleep function*

### 6.14.1 Detailed Description

#### **Todo**

documentation

Definition at line 39 of file BluetoothLinux.h.

### 6.14.2 Constructor & Destructor Documentation

#### 6.14.2.1 BluetoothLinuxFeatureProvider::L2Connection::L2Connection (BluetoothLinuxFeatureProvider \* *prov*, string *mac*)

Constructor.

#### **Parameters:**

- prov* **Feature** provider  
*mac* Hardware address of the target

Definition at line 259 of file BluetoothLinux.cpp.

### 6.14.3 Member Function Documentation

#### 6.14.3.1 `bool Thread::isCancelled ()` [protected, inherited]

**Returns:**

*true* if the thread is cancelled, *false* if the thread is still running

Definition at line 43 of file Thread.cpp.

References Thread::cancelled.

Referenced by GSMFeatureProvider::nextSample(), WlanLinuxFeatureProvider::run(), SystemCommandStringListFeatureProvider::run(), and BluetoothLinuxFeatureProvider::run().

#### 6.14.3.2 `void Thread::sleep (unsigned milliseconds)` [static, inherited]

sleep function

**Parameters:**

*milliseconds* Timeout in milliseconds

Definition at line 80 of file ThreadPosix.cpp.

Referenced by Main(), GSMFeatureProvider::readLine(), WlanLinuxFeatureProvider::run(), SystemCommandStringListFeatureProvider::run(), Scanner::run(), and BluetoothLinuxFeatureProvider::run().

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

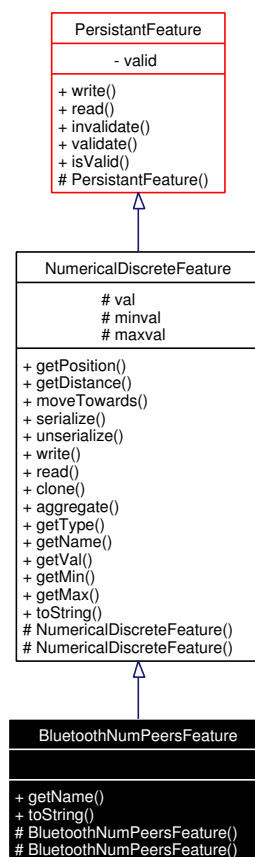
- [BluetoothLinux.h](#)
- [BluetoothLinux.cpp](#)

## 6.15 BluetoothNumPeersFeature Class Reference

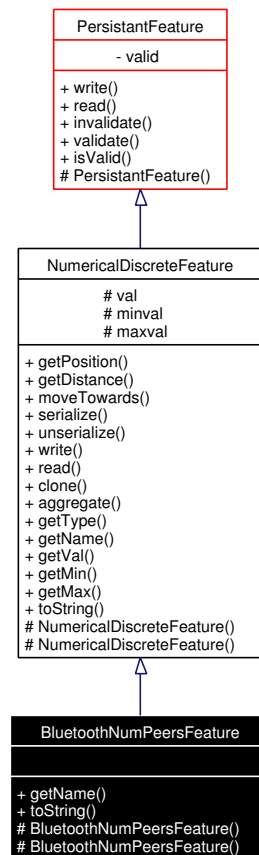
Number of peers in range.

```
#include <Bluetooth.h>
```

Inheritance diagram for BluetoothNumPeersFeature:



Collaboration diagram for BluetoothNumPeersFeature:



## Public Types

- enum [FeatureType](#) {  
**boolean, nominal, ordinal, numerical\_discrete,**  
**numerical\_continuous** }  
*Possible feature types.*

## Public Member Functions

- virtual const string [getName](#) () const  
*Query a features name.*
- virtual string [toString](#) () const  
*This is only for testing.*
- virtual double [getPosition](#) () const  
*Query a features position.*
- virtual double [getDistance](#) ([Feature](#) \*f) const  
*Calculates the distance between two features.*



- virtual void [moveTowards](#) ([Feature](#) \*f, double factor)

*Move feature.*

- virtual string [serialize](#) () const

*Serialize a samples data to a string.*

- virtual void [unserialize](#) (string value)

*Unserialize a samples data from a string.*

- virtual [featureparams](#) [write](#) () const

*Externalize feature.*

- virtual void [read](#) ([featureparams](#) \*param)

*Load feature from persistant data.*

- virtual [Feature](#) \* [clone](#) () const

*Clone a feature.*

- virtual void [aggregate](#) ([aggregatelist](#) samples)

*Aggregate a sample values from other sources.*

- virtual [FeatureType](#) [getType](#) () const

*Query a features type.*

- const long [getVal](#) () const

- const long [getMin](#) () const

- const long [getMax](#) () const

- void [invalidate](#) ()

*Invalidate feature.*

- void [validate](#) ()

*Set the validation flag to true.*

- bool [isValid](#) ()

*Query validation flag.*

- bool [isExternalizable](#) ()

*Query externalization flag.*

## Protected Member Functions

- [BluetoothNumPeersFeature](#) (long \*minval, long \*maxval)

*Feature constructor*

- [BluetoothNumPeersFeature](#) (long \*minval, long \*maxval, unsigned numPeers)

*Sample constructor.*

## Protected Attributes

- double `val`  
*The feature value.*
- long \* `minval`  
*A reference to the static, persistent minimum value seen so far.*
- long \* `maxval`  
*A reference to the static, persistent maximum value seen so far.*
- const bool `externalize`  
*Externalization flag.*

### 6.15.1 Detailed Description

Number of peers in range.

This feature returns the number of Bluetooth peers reachable by the calling device.

Definition at line 86 of file Bluetooth.h.

### 6.15.2 Constructor & Destructor Documentation

#### 6.15.2.1 `BluetoothNumPeersFeature::BluetoothNumPeersFeature (long * minval, long * maxval)` [inline, protected]

Feature constructor

This constructor initializes the feature with a random value and should thus only be used for creating e.g. prototypes of points in a clustering space. For creating a specific sample of a feature, the second constructor should be used.

##### Parameters:

***minval*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the minimum value.

***maxval*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the maximum value.

Definition at line 106 of file Bluetooth.h.

#### 6.15.2.2 `BluetoothNumPeersFeature::BluetoothNumPeersFeature (long * minval, long * maxval, unsigned numPeers)` [inline, protected]

Sample constructor.

This constructor initializes the feature with a specific value and should thus be used for samples.

##### Parameters:

***minval*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the minimum value.

*maxval* A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the maximum value.

*numPeers* The sample value.

Definition at line 122 of file Bluetooth.h.

### 6.15.3 Member Function Documentation

#### 6.15.3.1 `void NumericalDiscreteFeature::aggregate (aggregatelist samples) [virtual, inherited]`

Aggregate a sample values from other sources.

**Parameters:**

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 159 of file Numerical.cpp.

References [aggregatelist](#).

#### 6.15.3.2 `Feature * NumericalDiscreteFeature::clone () const [virtual, inherited]`

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Definition at line 154 of file Numerical.cpp.

References [NumericalDiscreteFeature::maxval](#), [NumericalDiscreteFeature::minval](#), [NumericalDiscreteFeature::NumericalDiscreteFeature\(\)](#), and [NumericalDiscreteFeature::val](#).

#### 6.15.3.3 `double NumericalDiscreteFeature::getDistance (Feature * f) const [virtual, inherited]`

Calculates the distance between two features.

**Parameters:**

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Definition at line 73 of file Numerical.cpp.

References [NumericalDiscreteFeature::getPosition\(\)](#), [NumericalDiscreteFeature::maxval](#), and [NumericalDiscreteFeature::minval](#).

#### 6.15.3.4 **virtual const string BluetoothNumPeersFeature::getName () const** [inline, virtual]

Query a features name.

##### Returns:

Name of the [Feature](#) in the format "Featureprovider.Feature"

Reimplemented from [NumericalDiscreteFeature](#).

Definition at line 125 of file Bluetooth.h.

#### 6.15.3.5 **double NumericalDiscreteFeature::getPosition () const** [virtual, inherited]

Query a features position.

##### Returns:

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

##### Remarks:

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Definition at line 57 of file Numerical.cpp.

References [NumericalDiscreteFeature::maxval](#), [NumericalDiscreteFeature::minval](#), and [NumericalDiscreteFeature::val](#).

Referenced by [NumericalDiscreteFeature::getDistance\(\)](#).

#### 6.15.3.6 **virtual FeatureType NumericalDiscreteFeature::getType () const** [inline, virtual, inherited]

Query a features type.

##### Returns:

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 103 of file Numerical.h.

#### 6.15.3.7 **const long NumericalDiscreteFeature::getVal () const** [inline, inherited]

##### Returns:

The value of the feature.

Definition at line 107 of file Numerical.h.

References [NumericalDiscreteFeature::val](#).

Referenced by [Java\\_at\\_jku\\_intelligence\\_samples\\_NumericalDiscreteSample\\_nativeGetVal\(\)](#), [WlanNumPeersFeature::toString\(\)](#), [NumericalDiscreteFeature::toString\(\)](#), and [toString\(\)](#).

**6.15.3.8 void PersistentFeature::invalidate () [inline, inherited]**

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by AbstractStringFeature::getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature::NumericalDiscreteFeature(), TimeFeature::TimeFeature(), NumericalContinuousFeature::unserialize(), and NumericalDiscreteFeature::unserialize().

**6.15.3.9 bool Feature::isExternalizable () [inline, inherited]**

Query externalization flag.

**Returns:**

*true* if the feature is externalizable ( i.e. it has persistent data that should be stored across restarts), *false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.15.3.10 bool PersistentFeature::isValid () [inline, inherited]**

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

**6.15.3.11 void NumericalDiscreteFeature::moveTowards (Feature \**f*, double *factor*) [virtual, inherited]**

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

**Parameters:**

*f* Feature used for distance measurement.

*factor* Distance weight.

Implements Feature.

Definition at line 80 of file Numerical.cpp.

References NumericalDiscreteFeature::maxval, NumericalDiscreteFeature::minval, and NumericalDiscreteFeature::val.

**6.15.3.12** `void NumericalDiscreteFeature::read (featureparams * param)` [virtual, inherited]

Load feature from persistant data.

Initializes the persistant feature data from the given representation.

**Parameters:**

*param* Persistant feature data.

**See also:**

[write](#)

Implements [PersistantFeature](#).

Definition at line 141 of file Numerical.cpp.

References featureparams, NumericalDiscreteFeature::maxval, and NumericalDiscreteFeature::minval.

**6.15.3.13** `string NumericalDiscreteFeature::serialize () const` [virtual, inherited]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [Feature](#).

Definition at line 96 of file Numerical.cpp.

References NumericalDiscreteFeature::val.

**6.15.3.14** `void NumericalDiscreteFeature::unserialize (string value)` [virtual, inherited]

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Definition at line 104 of file Numerical.cpp.

References [PersistantFeature::invalidate\(\)](#), [NumericalDiscreteFeature::maxval](#), [NumericalDiscreteFeature::minval](#), and [NumericalDiscreteFeature::val](#).

**6.15.3.15** `featureparams NumericalDiscreteFeature::write () const` [virtual, inherited]

Externalize feature.

**Returns:**

Persistant feature data.

**See also:**

[read](#)

Implements [PersistantFeature](#).

Definition at line 127 of file Numerical.cpp.

References [featureparams](#), [NumericalDiscreteFeature::maxval](#), and [NumericalDiscreteFeature::minval](#).

## 6.15.4 Member Data Documentation

### 6.15.4.1 `const bool Feature::externalize` [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistant features, the object should be cast to [PersistentFeature](#), because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**

[PersistantFeature](#)

Definition at line 187 of file Feature.h.

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

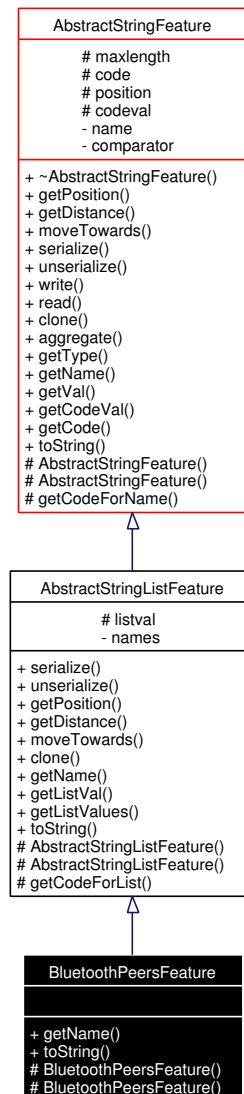
- [Bluetooth.h](#)
- [Bluetooth.cpp](#)

## 6.16 BluetoothPeersFeature Class Reference

List of peers in range.

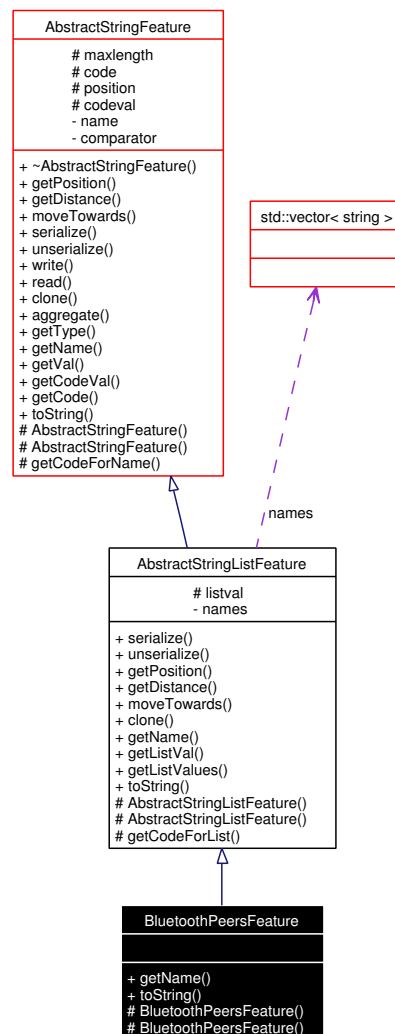
```
#include <Bluetooth.h>
```

Inheritance diagram for BluetoothPeersFeature:



Collaboration diagram for BluetoothPeersFeature:





## Public Types

- enum **ComparatorType** { **None**, **Levenshtein** }

*The distance metric to use.*

- enum **FeatureType** {  
**boolean**, **nominal**, **ordinal**, **numerical\_discrete**,  
**numerical\_continuous** }

*Possible feature types.*

## Public Member Functions

- virtual const string **getName** () const

*Query a features name.*

- virtual string [toString](#) () const  
*Get feature as string.*
- virtual string [serialize](#) () const  
*Serialize a samples data to a string.*
- virtual void [unserialize](#) (string value)  
*Unserialize a samples data from a string.*
- virtual double [getPosition](#) () const  
*Query a features position.*
- virtual double [getDistance](#) (Feature \*f) const  
*Calculates the distance between two features.*
- virtual void [moveTowards](#) (Feature \*f, double factor)  
*Move feature.*
- virtual Feature \* [clone](#) () const  
*Clone a feature.*
- const bit\_vector & [getListVal](#) () const  
*Query the list of values.*
- const stringvector & [getListValues](#) () const  
*Query the list of values.*
- virtual [featureparams write](#) () const  
*Externalize feature.*
- virtual void [read](#) (featureparams \*param)  
*Load feature from persistant data.*
- virtual void [aggregate](#) (aggregatelist samples)  
*Aggregate a sample values from other sources.*
- virtual FeatureType [getType](#) () const  
*Query a features type.*
- const string & [getVal](#) () const  
*Query the string values.*
- const unsigned long [getCodeVal](#) () const
- const [stringcode](#) \* [getCode](#) ()
- void [invalidate](#) ()  
*Invalidate feature.*

- void `validate` ()  
*Set the validation flag to true.*
- bool `isValid` ()  
*Query validation flag.*
- bool `isExternalizable` ()  
*Query externalization flag.*

## Protected Member Functions

- `BluetoothPeersFeature` (`stringcode` \*code, long \*maxlen)  
*Feature constructor*
- `BluetoothPeersFeature` (`stringcode` \*code, long \*maxlen, const `stringvector` \*strList)  
*Sample constructor.*
- bit\_vector `getCodeForList` (const `stringvector` \*names)  
*Get a code value for a given stringvector.*
- unsigned long `getCodeForName` (const string &name)  
*Get a code value for a given string.*

## Protected Attributes

- bit\_vector `listval`  
*The coded value of the feature.*
- long \* `maxlength`  
*A reference to the static, persistent maximum length of strings seen so far.*
- `stringcode` \* `code`  
*A reference to the static, persistent code table of strings seen so far.*
- char \* `position`  
*Only for internal usage in `moveTowards` and `getDistance`.*
- unsigned long `codeval`  
*The coded value of the feature.*
- const bool `externalize`  
*Externalization flag.*

## Related Functions

(Note that these are not member functions.)

- long [levenshtein](#) (char \*\*x, const char \*t, double \*factor)  
*Modified Levenshtein algorithm.*

### 6.16.1 Detailed Description

List of peers in range.

This feature returns a list of hardware addresses representing the Bluetooth peers in reachable by the calling device.

Definition at line 36 of file Bluetooth.h.

### 6.16.2 Constructor & Destructor Documentation

#### 6.16.2.1 BluetoothPeersFeature::BluetoothPeersFeature ([stringcode](#) \* *code*, long \* *maxlen*) [inline, protected]

Feature constructor

This constructor initializes the feature as prototypes value. For creating a specific sample of a feature, the sample constructor should be used.

##### Parameters:

- code*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the code table.
- maxlen*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum string length.

Definition at line 55 of file Bluetooth.h.

References `stringcode`.

#### 6.16.2.2 BluetoothPeersFeature::BluetoothPeersFeature ([stringcode](#) \* *code*, long \* *maxlen*, const [stringvector](#) \* *strList*) [inline, protected]

Sample constructor.

This constructor initializes the feature with a specific value and should thus be used for samples.

##### Parameters:

- code*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the code table.
- maxlen*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum string length.
- strList*** The string representation of the sample value.

Definition at line 71 of file Bluetooth.h.

References `stringcode`, and `stringvector`.

### 6.16.3 Member Function Documentation

#### 6.16.3.1 void AbstractStringFeature::aggregate ([aggregatelist](#) *samples*) [virtual, inherited]

Aggregate a sample values from other sources.

**Parameters:**

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 421 of file AbstractString.cpp.

References [aggregatelist](#).

#### 6.16.3.2 [Feature](#) \* AbstractStringListFeature::clone () const [virtual, inherited]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Reimplemented from [AbstractStringFeature](#).

Definition at line 581 of file AbstractString.cpp.

References [AbstractStringListFeature::AbstractStringListFeature\(\)](#), [AbstractStringListFeature::listval](#), and [AbstractStringListFeature::names](#).

#### 6.16.3.3 bit\_vector AbstractStringListFeature::getCodeForList (const [stringvector](#) \* *names*) [protected, inherited]

Get a code value for a given stringvector.

The function composes a bit\_vector wherein for each string in the stringvector the bit on the position of the strings code value is set to *true*. The remaining bits are initialized to *false*.

**Parameters:**

*names* The stringvector to code

**Returns:**

A code value as bit\_vector

Definition at line 450 of file AbstractString.cpp.

References [AbstractStringFeature::getCodeForName\(\)](#), and [stringvector](#).

Referenced by [AbstractStringListFeature::AbstractStringListFeature\(\)](#), and [WlanPeersFeature::WlanPeersFeature\(\)](#).

#### 6.16.3.4 unsigned long AbstractStringFeature::getCodeForName (const string & *name*) [protected, inherited]

Get a code value for a given string.

A helper function to return the respective code for a given string. If the string is encountered the first time a new code is allocated.

**Parameters:**

*name* The string to code

**Returns:**

A code value

Definition at line 303 of file AbstractString.cpp.

References AbstractStringFeature::code, PersistentFeature::invalidate(), and AbstractStringFeature::maxlength.

Referenced by AbstractStringFeature::AbstractStringFeature(), and AbstractStringListFeature::getCodeForList().

### 6.16.3.5 double AbstractStringListFeature::getDistance (Feature \*f) const [virtual, inherited]

Calculates the distance between two features.

**Parameters:**

*f* Feature used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Reimplemented from AbstractStringFeature.

Definition at line 518 of file AbstractString.cpp.

References AbstractStringListFeature::listval.

### 6.16.3.6 const bit\_vector& AbstractStringListFeature::getListVal () const [inline, inherited]

Query the list of values.

**Returns:**

Values list

Definition at line 244 of file AbstractString.h.

### 6.16.3.7 const stringvector& AbstractStringListFeature::getListValues () const [inline, inherited]

Query the list of values.

**Returns:**

Values list

Definition at line 253 of file AbstractString.h.

References [stringvector](#).

Referenced by [Java\\_at\\_jku\\_intelligence\\_samples\\_StringListSample\\_nativeGetValues\(\)](#), [WlanPeersFeature::toString\(\)](#), [SystemCommandStringListFeature::toString\(\)](#), [toString\(\)](#), and [AbstractStringListFeature::toString\(\)](#).

#### 6.16.3.8 virtual const string BluetoothPeersFeature::getName () const [inline, virtual]

Query a features name.

##### Returns:

Name of the [Feature](#) in the format "Featureprovider.Feature"

Reimplemented from [AbstractStringListFeature](#).

Definition at line 74 of file Bluetooth.h.

#### 6.16.3.9 double AbstractStringListFeature::getPosition () const [virtual, inherited]

Query a features position.

##### Returns:

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

##### Remarks:

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Reimplemented from [AbstractStringFeature](#).

Definition at line 498 of file AbstractString.cpp.

References [AbstractStringListFeature::listval](#).

#### 6.16.3.10 virtual [FeatureType](#) AbstractStringFeature::getType () const [inline, virtual, inherited]

Query a features type.

##### Returns:

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 100 of file AbstractString.h.

#### 6.16.3.11 const string& AbstractStringFeature::getVal () const [inline, inherited]

Query the string values.

**Returns:**

Values list

Definition at line 108 of file AbstractString.h.

References AbstractStringFeature::name.

Referenced by Java\_at\_jku\_intelligence\_samples\_StringSample\_nativeGetVal().

**6.16.3.12 void PersistentFeature::invalidate () [inline, inherited]**

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by AbstractStringFeature::getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature::NumericalDiscreteFeature(), TimeFeature::TimeFeature(), NumericalContinuousFeature::unserialize(), and NumericalDiscreteFeature::unserialize().

**6.16.3.13 bool Feature::isExternalizable () [inline, inherited]**

Query externalization flag.

**Returns:**

*true* if the feature is externalizable ( i.e. it has persistent data that should be stored across restarts),  
*false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.16.3.14 bool PersistentFeature::isValid () [inline, inherited]**

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

**6.16.3.15 void AbstractStringListFeature::moveTowards (Feature \*f, double factor) [virtual, inherited]**

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample s.

**Parameters:**

*f* Feature used for distance measurement.



*factor* Distance weight.

Reimplemented from [AbstractStringFeature](#).

Definition at line 546 of file AbstractString.cpp.

References AbstractStringListFeature::listval, and rand\_double.

**6.16.3.16 void AbstractStringFeature::read (featureparams \* param) [virtual, inherited]**

Load feature from persistant data.

Initializes the persistant feature data from the given representation.

**Parameters:**

*param* Persistant feature data.

**See also:**

[write](#)

Implements [PersistantFeature](#).

Definition at line 296 of file AbstractString.cpp.

References AbstractStringFeature::code, and featureparams.

**6.16.3.17 string AbstractStringListFeature::serialize () const [virtual, inherited]**

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Reimplemented from [AbstractStringFeature](#).

Definition at line 470 of file AbstractString.cpp.

References AbstractStringListFeature::listval.

Referenced by WlanPeersFeature::toString(), toString(), and AbstractStringListFeature::toString().

**6.16.3.18 string BluetoothPeersFeature::toString () const [virtual]**

Get feature as string.

**Note:**

This is only for testing.

**Returns:**

[Feature](#) as string.

Reimplemented from [AbstractStringListFeature](#).

Definition at line 29 of file Bluetooth.cpp.

References AbstractStringListFeature::getListValues(), and AbstractStringListFeature::serialize().

**6.16.3.19 void AbstractStringListFeature::unserialize (string *value*)** [virtual, inherited]

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Reimplemented from [AbstractStringFeature](#).

Definition at line 480 of file AbstractString.cpp.

References AbstractStringListFeature::listval.

**6.16.3.20 [featureparams](#) AbstractStringFeature::write () const** [virtual, inherited]

Externalize feature.

**Returns:**

Persistent feature data.

**See also:**

[read](#)

Implements [PersistentFeature](#).

Definition at line 283 of file AbstractString.cpp.

References AbstractStringFeature::code, and [featureparams](#).

**6.16.4 Friends And Related Function Documentation****6.16.4.1 long levenshtein (char \*\* *x*, const char \* *t*, double \* *factor*)** [related, inherited]

Modified Levenshtein algorithm.

This function implements a slightly modified version of the Levenshtein algorithm to not only calculate the distance of two strings but also modify the string *x* towards the string *t* with the probability *factor*. In other words, every transformation operation found by the algorithm is performed on the string *x* with the probability *factor*.

**Parameters:**

*x* Input string.

*t* String to compare the input string to.

*factor* Probability with witch transformations are performed. Has to be a value out of the intervall  $[0; 1]$ .

**Returns:**

The levenshtein distance between *x* and *t*.

**Todo**

alloc once

Definition at line 46 of file AbstractString.cpp.

References rand\_double.

Referenced by AbstractStringFeature::getDistance(), and AbstractStringFeature::moveTowards().

## 6.16.5 Member Data Documentation

### 6.16.5.1 unsigned long [AbstractStringFeature::codeval](#) [protected, inherited]

The coded value of the feature.

See also:

[name](#)  
[code](#)

Definition at line 144 of file [AbstractString.h](#).

Referenced by [AbstractStringFeature::AbstractStringFeature\(\)](#), [AbstractStringFeature::clone\(\)](#), [AbstractStringFeature::getCodeVal\(\)](#), [AbstractStringFeature::getDistance\(\)](#), [AbstractStringFeature::moveTowards\(\)](#), [AbstractStringFeature::serialize\(\)](#), and [AbstractStringFeature::unserialize\(\)](#).

### 6.16.5.2 const bool [Feature::externalize](#) [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to [PersistentFeature](#), because only subclasses of this type are (by policy) allowed to set this variable to true.

See also:

[PersistentFeature](#)

Definition at line 187 of file [Feature.h](#).

### 6.16.5.3 bit\_vector [AbstractStringListFeature::listval](#) [protected, inherited]

The coded value of the feature.

See also:

[names](#)  
[code](#)

Definition at line 180 of file [AbstractString.h](#).

Referenced by [AbstractStringListFeature::AbstractStringListFeature\(\)](#), [AbstractStringListFeature::clone\(\)](#), [AbstractStringListFeature::getDistance\(\)](#), [AbstractStringListFeature::getPosition\(\)](#), [AbstractStringListFeature::moveTowards\(\)](#), [AbstractStringListFeature::serialize\(\)](#), and [AbstractStringListFeature::unserialize\(\)](#).

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

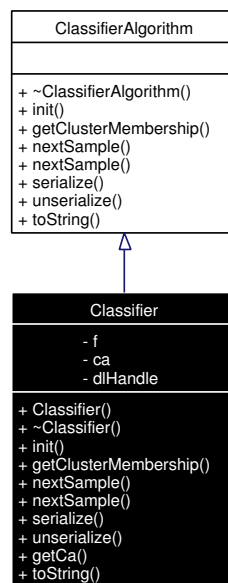
- [Bluetooth.h](#)
- [Bluetooth.cpp](#)

## 6.17 Classifier Class Reference

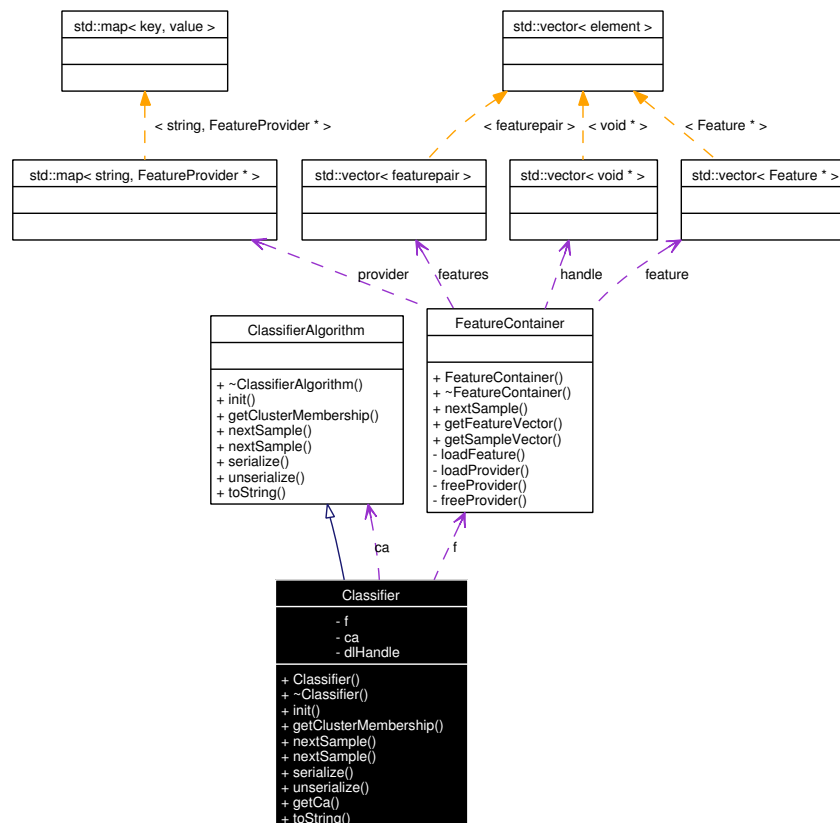
Classifier Wrapper.

```
#include <Classifier.h>
```

Inheritance diagram for Classifier:



Collaboration diagram for Classifier:



## Public Member Functions

- **Classifier** (const string &name, const **classifierparams** &params)  
*Creates the classifier.*
- virtual **~Classifier** ()  
*Destructor.*
- virtual void **init** (FeatureContainer \*fc)  
*Initializer.*
- virtual **membershiplist** **getClusterMembership** (const **featurevector** \*sample)  
*Get membership vector.*
- virtual unsigned long **nextSample** ()  
*Fetch next sample vector.*
- virtual unsigned long **nextSample** (const **featurevector** \*sample)  
*Fetch next sample vector Gets the next sample vector passed as parameter.*
- virtual string **serialize** () const  
*Serialize a samples data to a string.*

- virtual void [unserialize](#) (string data)  
*Unserialize a samples data from a string.*
- [ClassifierAlgorithm](#) \* [getCa](#) () const  
*Return actual [ClassifierAlgorithm](#) instance.*
- virtual string [toString](#) () const  
*This is only for testing.*

## Private Attributes

- [FeatureContainer](#) \* [f](#)  
*Instance.*
- [ClassifierAlgorithm](#) \* [ca](#)  
*Implementation.*
- void \* [dlHandle](#)  
*Library handle.*

## 6.17.1 Detailed Description

Classifier Wrapper.

The Classifier Class is a wrapper class that loads a dll containing the actual implementation during runtime.

Definition at line 167 of file Classifier.h.

## 6.17.2 Constructor & Destructor Documentation

### 6.17.2.1 Classifier::Classifier (const string & *name*, const [classifierparams](#) & *params*)

Creates the classifier.

#### Parameters:

*name* The name of the requested classifier implementation.

*params* A map of parameters to initialize the classifier.

Definition at line 27 of file LinuxClassifier.cpp.

References [ca](#), [classifierparams](#), [dlHandle](#), [getClassifier\(\)](#), and [getClassifier\\_t](#).

## 6.17.3 Member Function Documentation

### 6.17.3.1 virtual [membershiplist](#) Classifier::getClusterMembership (const [featurevector](#) \* *sample*) [inline, virtual]

Get membership vector.

This method returns the data for the next steps, which is a list of context classes found by the algorithm with associated membership values. These values specify the membership of the current feature values to each of the context classes in the interval [0; 1]. They can thus be seen as the probabilities that the context classes are currently active.

**Parameters:**

*sample* The sample vector for which the membership list should be calculated

Implements [ClassifierAlgorithm](#).

Definition at line 193 of file Classifier.h.

Referenced by `evaluate()`, and `Scanner::run()`.

### 6.17.3.2 `virtual void Classifier::init (FeatureContainer *fc) [inline, virtual]`

Initializer.

Initializes internal data structures of the classification/clustering algorithm. This usually involves constructing initial prototypes for cluster centers; thus, randomized feature vectors are necessary for the initialization. The `getFeatureVector` method of the passed [FeatureContainer](#) object may be called multiple times by the initialization. The given [FeatureContainer](#) is also used when `nextSample()` is called.

**Parameters:**

*fc* The [FeatureContainer](#) from which the randomized feature vectors should be retrieved.

**See also:**

[FeatureContainer::getFeatureVector](#)  
[nextSample\(\)](#)

Implements [ClassifierAlgorithm](#).

Definition at line 192 of file Classifier.h.

Referenced by `evaluate()`, and `Scanner::run()`.

### 6.17.3.3 `virtual unsigned long Classifier::nextSample (const featurevector * sample) [inline, virtual]`

Fetch next sample vector Gets the next sample vector passed as parameter.

**Parameters:**

*sample* samplevector

Implements [ClassifierAlgorithm](#).

Definition at line 195 of file Classifier.h.

### 6.17.3.4 `virtual unsigned long Classifier::nextSample () [inline, virtual]`

Fetch next sample vector.

Gets the next sample vector from the configured feature source. This method calls `nextSample` and `getSampleVector` on the [FeatureContainer](#) given to the `init` method.

**See also:**

[FeatureContainer::nextSample](#)  
[FeatureContainer::getSampleVector](#)

Implements [ClassifierAlgorithm](#).

Definition at line 194 of file Classifier.h.

Referenced by `evaluate()`, and `Scanner::run()`.

**6.17.3.5 virtual string Classifier::serialize () const [inline, virtual]**

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [ClassifierAlgorithm](#).

Definition at line 196 of file Classifier.h.

Referenced by `evaluate()`.

**6.17.3.6 virtual void Classifier::unserialize (string data) [inline, virtual]**

Unserialize a samples data from a string.

**Parameters:**

*data* String representation of the samples data.

Implements [ClassifierAlgorithm](#).

Definition at line 197 of file Classifier.h.

References `ca`, `f`, `fc`, and `ClassifierAlgorithm::init()`.

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

- [Classifier.h](#)
- [LinuxClassifier.cpp](#)
- [WindowsClassifier.cpp](#)

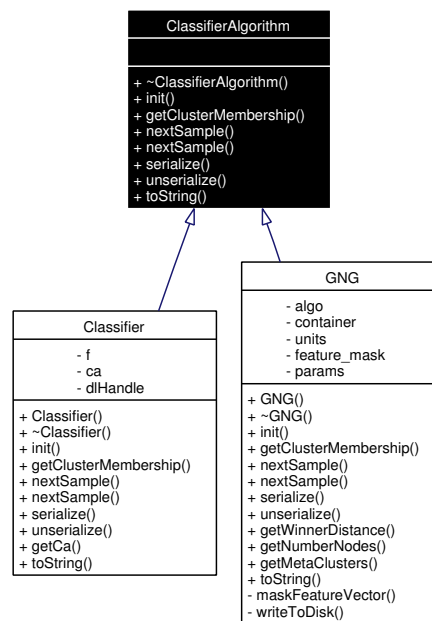


## 6.18 ClassifierAlgorithm Class Reference

ClassifierAlgorithm Interface.

```
#include <Classifier.h>
```

Inheritance diagram for ClassifierAlgorithm:



### Public Member Functions

- virtual `~ClassifierAlgorithm ()`  
*Destructor.*
- virtual void `init (FeatureContainer *fc)=0`  
*Initializer.*
- virtual `membershiplist getClusterMembership (const featurevector *sample)=0`  
*Get membership vector.*
- virtual unsigned long `nextSample ()=0`  
*Fetch next sample vector.*
- virtual unsigned long `nextSample (const featurevector *sample)=0`  
*Fetch next sample vector Gets the next sample vector passed as parameter.*
- virtual string `serialize () const =0`  
*Serialize a samples data to a string.*
- virtual void `unserialize (string data)=0`  
*Unserialize a samples data from a string.*

- virtual string [toString](#) () const =0

*This is only for testing.*

## 6.18.1 Detailed Description

ClassifierAlgorithm Interface.

This class defines the abstract interface to classification/clustering algorithms. Any algorithm which is to be used for the classification step in this framework has to implement these methods and be derived from this class.

Definition at line 82 of file Classifier.h.

## 6.18.2 Member Function Documentation

### 6.18.2.1 virtual [membershiplist](#) ClassifierAlgorithm::getClusterMembership (const [featurevector](#) \* *sample*) [pure virtual]

Get membership vector.

This method returns the data for the next steps, which is a list of context classes found by the algorithm with associated membership values. These values specify the membership of the current feature values to each of the context classes in the interval [0; 1]. They can thus be seen as the probabilities that the context classes are currently active.

#### Parameters:

*sample* The sample vector for which the membership list should be calculated

Implemented in [Classifier](#), and [GNG](#).

Referenced by [Java\\_at\\_jku\\_intelligence\\_context\\_Context\\_nativeGetClusterMembership\(\)](#), and [Java\\_at\\_jku\\_intelligence\\_context\\_Context\\_nativeGetNumClusters\(\)](#).

### 6.18.2.2 virtual void ClassifierAlgorithm::init ([FeatureContainer](#) \* *fc*) [pure virtual]

Initializer.

Initializes internal data structures of the classification/clustering algorithm. This usually involves constructing initial prototypes for cluster centers; thus, randomized feature vectors are necessary for the initialization. The [getFeatureVector](#) method of the passed [FeatureContainer](#) object may be called multiple times by the initialization. The given [FeatureContainer](#) is also used when [nextSample\(\)](#) is called.

#### Parameters:

*fc* The [FeatureContainer](#) from which the randomized feature vectors should be retrieved.

#### See also:

[FeatureContainer::getFeatureVector](#)  
[nextSample\(\)](#)

Implemented in [Classifier](#), and [GNG](#).

Referenced by [Java\\_at\\_jku\\_intelligence\\_context\\_Context\\_nativeInit\(\)](#), and [Classifier::unserialize\(\)](#).

**6.18.2.3 virtual unsigned long ClassifierAlgorithm::nextSample (const [featurevector](#) \* *sample*)**  
[pure virtual]

Fetch next sample vector Gets the next sample vector passed as parameter.

**Parameters:**

*sample* samplevector

Implemented in [Classifier](#), and [GNG](#).

**6.18.2.4 virtual unsigned long ClassifierAlgorithm::nextSample ()** [pure virtual]

Fetch next sample vector.

Gets the next sample vector from the configured feature source. This method calls nextSample and getSampleVector on the [FeatureContainer](#) given to the init method.

**See also:**

[FeatureContainer::nextSample](#)

[FeatureContainer::getSampleVector](#)

Implemented in [Classifier](#), and [GNG](#).

Referenced by Classifier::getCa(), and Java\_at\_jku\_intelligence\_context\_Context\_nativeNextSample().

**6.18.2.5 virtual string ClassifierAlgorithm::serialize () const** [pure virtual]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implemented in [Classifier](#), and [GNG](#).

**6.18.2.6 virtual void ClassifierAlgorithm::unserialize (string *data*)** [pure virtual]

Unserialize a samples data from a string.

**Parameters:**

*data* String representation of the samples data.

Implemented in [Classifier](#), and [GNG](#).

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

- [Classifier.h](#)

## 6.19 config\_param Struct Reference

description for parameters in the config file

```
#include <conf.h>
```

### Public Attributes

- char \* **param\_name**
- char \* **param\_help**
- int **conf\_value**
- conf\_copy\_func **copy**
- conf\_print\_func **print**

### 6.19.1 Detailed Description

description for parameters in the config file

Definition at line 115 of file conf.h.

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

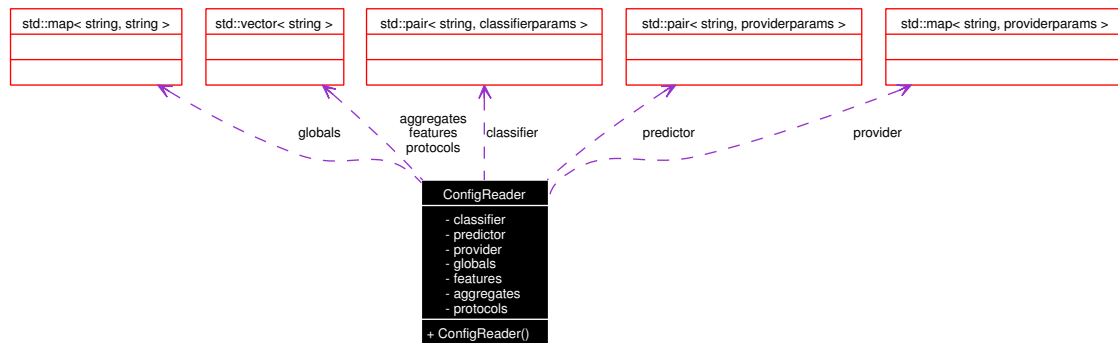
- conf.h

## 6.20 ConfigReader Class Reference

Configuration parser.

```
#include <ConfigReader.h>
```

Collaboration diagram for ConfigReader:



### Public Member Functions

- [ConfigReader](#) (const char \*file)  
*Constructor.*
- const [pair](#)< string, [map](#)< string, string > > \* [getClassifier](#) ()
- const [pair](#)< string, [map](#)< string, string > > \* [getPredictor](#) ()
- const [map](#)< string, [map](#)< string, string > > \* [getFeatureProvider](#) ()
- const [vector](#)< string > \* [getFeatures](#) ()
- const [vector](#)< string > \* [getAggregates](#) ()
- const [vector](#)< string > \* [getProtocols](#) ()
- const [map](#)< string, string > \* [getGlobals](#) ()

### Private Attributes

- [pair](#)< string, [classifierparams](#) > [classifier](#)  
*Classifier settings.*
- [pair](#)< string, [providerparams](#) > [predictor](#)  
*Predictor settings.*
- [map](#)< string, [providerparams](#) > [provider](#)  
*Provider settings.*
- [parametermap](#) [globals](#)  
*Global settings.*
- [stringvector](#) [features](#)  
*Features.*

- [stringvector aggregates](#)

*Aggregates.*

- [stringvector protocols](#)

*Protocols.*

## 6.20.1 Detailed Description

Configuration parser.

This class parses the configuration file and provides the settings as hashtables.

Definition at line 37 of file ConfigReader.h.

## 6.20.2 Constructor & Destructor Documentation

### 6.20.2.1 ConfigReader::ConfigReader (const char \* *file*)

Constructor.

#### Parameters:

*file* Path to the configuration file

Definition at line 27 of file ConfigReader.cpp.

References aggregates, classifier, features, globals, predictor, protocols, and provider.

## 6.20.3 Member Function Documentation

### 6.20.3.1 const [vector](#)< string > \* ConfigReader::getAggregates ()

#### Todo

documentation  
source cleanup (don't copy maps)

Definition at line 157 of file ConfigReader.cpp.

References aggregates.

### 6.20.3.2 const [pair](#)< string, [map](#)< string, string > > \* ConfigReader::getClassifier ()

#### Todo

documentation  
source cleanup (don't copy maps)

Definition at line 147 of file ConfigReader.cpp.

References classifier.

Referenced by Java\_at\_jku\_intelligence\_context\_Context\_nativeInit(), main(), and Scanner::run().

**6.20.3.3** `const map< string, map< string, string > > * ConfigReader::getFeatureProvider ()`**Todo**

- documentation
- source cleanup (don't copy maps)

Definition at line 167 of file ConfigReader.cpp.

References provider.

Referenced by FeatureContainer::FeatureContainer().

**6.20.3.4** `const vector< string > * ConfigReader::getFeatures ()`**Todo**

- documentation
- source cleanup (don't copy maps)

Definition at line 152 of file ConfigReader.cpp.

References features.

Referenced by FeatureContainer::FeatureContainer().

**6.20.3.5** `const map< string, string > * ConfigReader::getGlobals ()`**Todo**

- documentation
- source cleanup (don't copy maps)

Definition at line 172 of file ConfigReader.cpp.

References globals.

Referenced by FeatureContainer::FeatureContainer(), LoggingFeatureContainer::LoggingFeatureContainer(), Main(), main(), ReplayFeatureContainer::ReplayFeatureContainer(), and Scanner::run().

**6.20.3.6** `const pair<string, map<string, string> >* ConfigReader::getPredictor ()`**Todo**

- documentation
- source cleanup (don't copy maps)

**6.20.3.7** `const vector< string > * ConfigReader::getProtocols ()`**Todo**

- documentation
- source cleanup (don't copy maps)

Definition at line 162 of file ConfigReader.cpp.

References protocols.

## 6.20.4 Member Data Documentation

### 6.20.4.1 `pair<string, classifierparams> ConfigReader::classifier` [private]

[Classifier](#) settings.

This variable stores the name of the classifier algorithm and a map of parameters for the classifier. The data structure represents the `<classifier>` section of the configuration file. Definition at line 48 of file `ConfigReader.h`.

Referenced by `ConfigReader()`, and `getClassifier()`.

### 6.20.4.2 `parametermap ConfigReader::globals` [private]

Global settings.

This variable stores global settings to control the framework. The data structure represents the `<globals>` section of the configuration file. Definition at line 80 of file `ConfigReader.h`.

Referenced by `ConfigReader()`, and `getGlobals()`.

### 6.20.4.3 `pair<string, providerparams> ConfigReader::predictor` [private]

[Predictor](#) settings.

This variable stores the name of the predictor algorithm and a map of parameters for the predictor. The data structure represents the `<predictor>` section of the configuration file. Definition at line 59 of file `ConfigReader.h`.

Referenced by `ConfigReader()`.

### 6.20.4.4 `map<string, providerparams> ConfigReader::provider` [private]

Provider settings.

This variable stores additional configuration parameters for various feature providers. The data structure represents the `<featureproviders>` section of the configuration file. Definition at line 70 of file `ConfigReader.h`.

Referenced by `ConfigReader()`, and `getFeatureProvider()`.

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

- [ConfigReader.h](#)
- [ConfigReader.cpp](#)

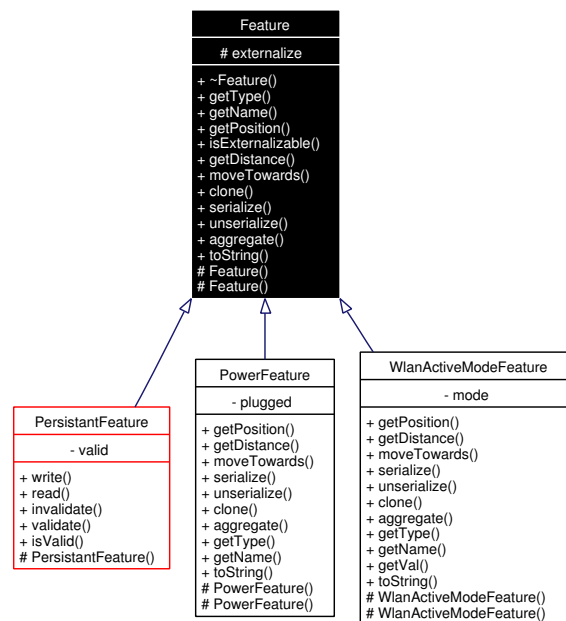


## 6.21 Feature Class Reference

Feature interface.

```
#include <Feature.h>
```

Inheritance diagram for Feature:



### Public Types

- enum `FeatureType` {  
     **boolean**, **nominal**, **ordinal**, **numerical\_discrete**,  
     **numerical\_continuous** }

*Possible feature types.*

### Public Member Functions

- virtual `~Feature ()`  
*Destructor.*
- virtual `FeatureType getType () const =0`  
*Query a features type.*
- virtual const string `getName () const =0`  
*Query a features name.*
- virtual double `getPosition () const =0`  
*Query a features position.*

- bool `isExternalizable ()`  
*Query externalization flag.*
- virtual double `getDistance (Feature *f) const =0`  
*Calculates the distance between two features.*
- virtual void `moveTowards (Feature *f, double factor)=0`  
*Move feature.*
- virtual `Feature * clone () const =0`  
*Clone a feature.*
- virtual string `serialize () const =0`  
*Serialize a samples data to a string.*
- virtual void `unserialize (string value)=0`  
*Unserialize a samples data from a string.*
- virtual void `aggregate (aggregatelist samples)=0`  
*Aggregate a sample values from other sources.*
- virtual string `toString () const =0`  
*This is only for testing.*

## Protected Member Functions

- `Feature (bool ext)`  
*Constructor for usage in PerstantFeature.*
- `Feature ()`  
*Default constructor.*

## Protected Attributes

- const bool `externalize`  
*Externalization flag.*

### 6.21.1 Detailed Description

Feature interface.

A feature represents a single dimension in the cluster space and implements the necessary functionality to compare features of the same kind and measure distances between them. A feature is identified by a name which has to be unique in the scope of its feature provider. It abstracts the type and structure of the underlying sensor data and can also apply arbitrary data processing and transformation to the raw sensor

data. Typically, a sensor (i.e. a [FeatureProvider](#)) will provide multiple features which are extracted using domain-specific knowledge.

A feature will also be referenced as sample. Samples are a more generic form of a feature only containing information about their position and without information about corresponding neighbours, age, error, etc.

**See also:**

[FeatureProvider](#)

Definition at line 167 of file Feature.h.

## 6.21.2 Constructor & Destructor Documentation

### 6.21.2.1 **Feature::Feature** (bool *ext*) [inline, protected]

Constructor for usage in PerstantFeature.

Sets the externalization flag to the given value.

**See also:**

`extnlze`

Definition at line 196 of file Feature.h.

### 6.21.2.2 **Feature::Feature** () [inline, protected]

Default constructor.

Sets the externalization flag to *false*.

**See also:**

`extnlze`

Definition at line 205 of file Feature.h.

### 6.21.2.3 **virtual Feature::~~Feature** () [inline, virtual]

Destructor.

Empty virtual destructor to have an entry in the implementing classes method table. This ensures that derived classes get destructed correctly. Definition at line 215 of file Feature.h.

## 6.21.3 Member Function Documentation

### 6.21.3.1 **virtual void Feature::aggregate** ([aggregatelist](#) *samples*) [pure virtual]

Aggregate a sample values from other sources.

**Parameters:**

*samples* A list of <timestamp, sample> tuples.

Implemented in [AbstractStringFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

**6.21.3.2 virtual [Feature\\*](#) Feature::clone () const** [pure virtual]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

**6.21.3.3 virtual double Feature::getDistance ([Feature](#) \**f*) const** [pure virtual]

Calculates the distance between two features.

**Parameters:**

*f* Feature used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

**6.21.3.4 virtual const string Feature::getName () const** [pure virtual]

Query a features name.

**Returns:**

Name of the Feature in the format "Featureprovider.Feature"

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [ActiveWindowFeature](#), [AudioBandFeature](#), [AudioMeanFeature](#), [AudioPeakFeature](#), [BluetoothPeersFeature](#), [BluetoothNumPeersFeature](#), [GSMCellFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [SystemCommandStringListFeature](#), [TimeFeature](#), [WlanActiveEssidFeature](#), [WlanActiveMacAddressFeature](#), [WlanActiveModeFeature](#), [WlanActiveSignalLevelFeature](#), [WlanPeersFeature](#), and [WlanNumPeersFeature](#).

Referenced by `Java_at_jku_intelligence_samples_Sample_nativeGetName()`.

**6.21.3.5 virtual double Feature::getPosition () const** [pure virtual]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

Referenced by `Java_at_jku_intelligence_samples_Sample_nativeGetPosition()`.

**6.21.3.6 virtual [FeatureType](#) Feature::getType () const** [pure virtual]

Query a features type.

**Returns:**

The type of the Feature.

Implemented in [AbstractStringFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

Referenced by `Java_at_jku_intelligence_samples_Sample_nativeGetType()`.

**6.21.3.7 bool Feature::isExternalizable ()** [inline]

Query externalization flag.

**Returns:**

*true* if the feature is externalizeable ( i.e. it has persistend data that should be stored across restarts),  
*false* otherwise.

Definition at line 250 of file `Feature.h`.

Referenced by `FeatureContainer::FeatureContainer()`, and `FeatureContainer::nextSample()`.

**6.21.3.8 virtual void Feature::moveTowards ([Feature](#) \**f*, double *factor*)** [pure virtual]

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

**Parameters:**

*f* Feature used for distance measurement.

*factor* Distance weight.

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

Referenced by `Unit::splitUnit()`.

**6.21.3.9 virtual string Feature::serialize () const** [pure virtual]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

**6.21.3.10 virtual void Feature::unserialize (string *value*)** [pure virtual]

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [NumericalDiscreteFeature](#), [Numerical-ContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

Referenced by [unserializeFeatureVector\(\)](#).

**6.21.4 Member Data Documentation****6.21.4.1 const bool [Feature::externalize](#)** [protected]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to [PersistentFeature](#), because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**

[PersistentFeature](#)

Definition at line 187 of file [Feature.h](#).

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

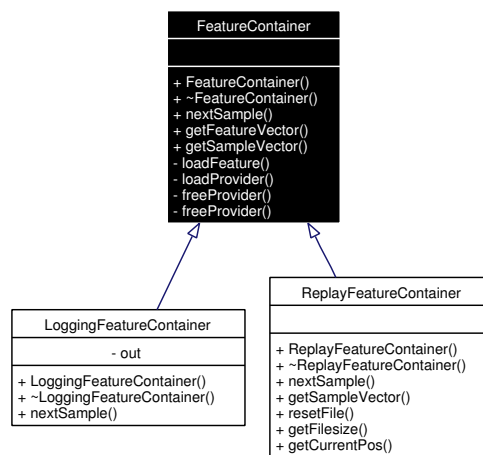
- [Feature.h](#)

## 6.22 FeatureContainer Class Reference

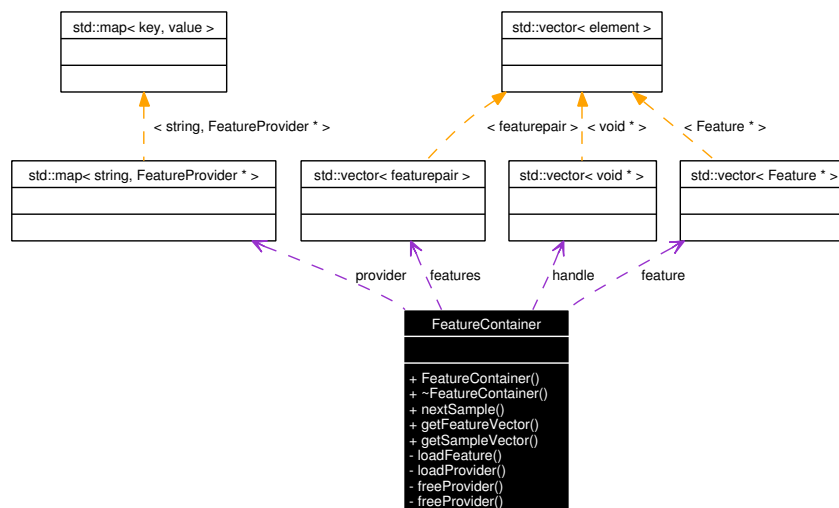
Container class.

```
#include <FeatureContainer.h>
```

Inheritance diagram for FeatureContainer:



Collaboration diagram for FeatureContainer:



### [NOHEADER]

- typedef `pair< string, FeatureProvider * >` `featurepair`

*Protected property.*

- typedef `vector< featurepair >` `featuremap`

*Protected property.*

- [featuremap](#) features

*Protected property.*

## Public Member Functions

- [FeatureContainer](#) ([ConfigReader](#) \*config)

*Constructor.*

- virtual [~FeatureContainer](#) ()

*Destructor.*

- virtual void [nextSample](#) ()

*Retrieve next sample.*

- virtual [featurevector](#) [getFeatureVector](#) () const

*Get random feature vector.*

- virtual const [featurevector](#) \* [getSampleVector](#) () const

*Get sample vector.*

## Private Types

- typedef [map](#)< string, [FeatureProvider](#) \* > [providermap](#)

*Map of [FeatureProvider](#) handles.*

## Private Member Functions

- bool [loadFeature](#) (const char \*id)

*Load a feature.*

- [FeatureProvider](#) \* [loadProvider](#) (const char \*provider, const [providerparams](#) &args)

*Load a provider.*

- void [freeProvider](#) (void \*provider)

*Unload a provider.*

- void [freeProvider](#) ()

*Cleanup container.*



## Private Attributes

- string `persist`  
*Private property.*
- `vector< void * >` `handle`  
*Private property.*
- `providermap` `provider`  
*Private property.*
- `featurevector` `feature`  
*Private property.*
- FILE \* `out`  
*Private property.*

### 6.22.1 Detailed Description

Container class.

Loads and initializes the feature providers as defined in the configuration file. It provides methods to commonly operate on all loaded providers and their requested features.

Definition at line 39 of file FeatureContainer.h.

### 6.22.2 Member Typedef Documentation

#### 6.22.2.1 `typedef vector<featurepair> FeatureContainer::featuremap` [protected]

Protected property.

**Todo**

documentation

Definition at line 110 of file FeatureContainer.h.

#### 6.22.2.2 `typedef pair<string, FeatureProvider*> FeatureContainer::featurepair` [protected]

Protected property.

**Todo**

documentation

Definition at line 109 of file FeatureContainer.h.

Referenced by `loadFeature()`.

### 6.22.3 Constructor & Destructor Documentation

#### 6.22.3.1 FeatureContainer::FeatureContainer (ConfigReader \* config)

Constructor.

Loads all feature providers listed in the configuration file and initializes them with the according parameters. It then loads all requested features and restores persistent information for them.

**Parameters:**

*config* A ConfigReader instance

Definition at line 29 of file FeatureContainer.cpp.

References featureparams, features, ConfigReader::getFeatureProvider(), ConfigReader::getFeatures(), ConfigReader::getGlobals(), PersistentReader::getPersistentData(), Feature::isExternalizable(), loadFeature(), loadProvider(), persist, and provider.

### 6.22.4 Member Function Documentation

#### 6.22.4.1 void FeatureContainer::freeProvider () [private]

Cleanup container.

This function calls freeProvider for every loaded provider in the container. It is intended to be called on shutdown of the framework. Definition at line 198 of file FeatureContainer.cpp.

References handle.

Referenced by loadFeature(), and ~FeatureContainer().

#### 6.22.4.2 void FeatureContainer::freeProvider (void \* provider) [private]

Unload a provider.

Cleans up all used memory and unloads the library.

**Parameters:**

*provider* Handle of the provider library

Definition at line 64 of file LinuxFeatureContainer.cpp.

#### 6.22.4.3 featurevector FeatureContainer::getFeatureVector () const [virtual]

Get random feature vector.

A feature vector contains pointers to implementations of the specific features, but initialized with random values. These random positions in the feature space are usually used for initializing classification algorithms. The feature objects returned by this method are allocated automatically for the caller (since the caller can not know the specific implementations, this method acts as a factory), but **must** be freed by the caller. The specific feature objects are created by the FeatureContainer implementations, as this method calls getFeature for every dimension.

**Returns:**

A vector of Feature implementations. Every element of the returned vector must be freed after use.

**See also:**[Feature](#)

FeatureContainer::getFeature

Definition at line 179 of file FeatureContainer.cpp.

References features, and featurevector.

Referenced by ReplayFeatureContainer::nextSample(), and Scanner::run().

**6.22.4.4** `const featurevector * FeatureContainer::getSampleVector () const` [virtual]

Get sample vector.

A sample vector contains pointers to implementations of the specific features, with values representing the current sensor values. For performance reasons (since this method can be called by multiple callers in the same time step for retrieving the current feature values), this method returns a pointer to a globally allocated object. Callers **must not** free or modify either the returned vector reference or the feature implementations contained therein.

**Returns:**

A reference to a globally allocated vector containing features that represent the current sensor values.

Reimplemented in [ReplayFeatureContainer](#).

Definition at line 193 of file FeatureContainer.cpp.

References feature, and featurevector.

Referenced by evaluate(), Java\_at\_jku\_intelligence\_context\_Context\_nativeGetClusterMembership(), Java\_at\_jku\_intelligence\_context\_Context\_nativeGetNumClusters(), Java\_at\_jku\_intelligence\_samples\_SampleContainer\_nativeGetSampleVector(), LoggingFeatureContainer::nextSample(), and Scanner::run().

**6.22.4.5** `bool FeatureContainer::loadFeature (const char * id)` [private]

Load a feature.

A feature is identified by a unique name of the form <providername>.<featurename>. loadFeature calls loadProvider with the <providername> part to load the correct feature provider and requests the feature <featurename> from it.

**Parameters:**

*id* Unique name of the feature

**Returns:**

*true* on success, otherwise *false*

Definition at line 95 of file FeatureContainer.cpp.

References featurepair, features, freeProvider(), FeatureProvider::getFeatureList(), handle, loadProvider(), provider, and stringvector.

Referenced by FeatureContainer().

#### 6.22.4.6 [FeatureProvider](#) \* [FeatureContainer::loadProvider](#) (const char \* *provider*, const [providerparams](#) & *args*) [private]

Load a provider.

Like a feature a provider must also have a unique name (this constraint is also given by the fact that a feature provider is a shared library). `loadProvider` locates the feature provider in the file system and loads it into the framework passing a `providerparams` datastructure to it for initialization.

##### Parameters:

*provider* Unique name of the feature provider

*args* A `providerparams` datastructure containing additional parameters from the configuration file.

##### Returns:

An instance of a [FeatureProvider](#) class

Definition at line 28 of file `LinuxFeatureContainer.cpp`.

References `getProvider()`, `getProvider_t`, `handle`, and `providerparams`.

Referenced by `FeatureContainer()`, and `loadFeature()`.

#### 6.22.4.7 `void FeatureContainer::nextSample ()` [virtual]

Retrieve next sample.

Retrieve next samples from all features at once. This method should be called at the beginning of each time step. This method calls `nextSample` for each `FeatureContainer` and then allocates the global feature vector which can be returned by successive calls to `getSampleVector`. The specific feature objects are created by the `FeatureContainer` implementations, as this method calls `getSample` for every dimension.

##### See also:

[FeatureContainer::nextSample](#)

[FeatureContainer::getSample](#)

[getSampleVector](#)

Reimplemented in [LoggingFeatureContainer](#), and [ReplayFeatureContainer](#).

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

References `feature`, `features`, `Feature::isExternalizable()`, `PersistentFeature::isValid()`, `persist`, `provider`, `PersistentFeature::validate()`, `PersistentWriter::write()`, and `PersistentFeature::write()`.

Referenced by `evaluate()`, `Java_at_jku_intelligence_samples_SampleContainer_nativeNextSamples()`, `LoggingFeatureContainer::nextSample()`, and `Scanner::run()`.

## 6.22.5 Member Data Documentation

#### 6.22.5.1 [featurevector](#) [FeatureContainer::feature](#) [private]

Private property.

##### Todo

documentation

Definition at line 52 of file FeatureContainer.h.

Referenced by `getSampleVector()`, `nextSample()`, and `~FeatureContainer()`.

#### 6.22.5.2 `featuremap` `FeatureContainer::features` [protected]

Protected property.

##### Todo

documentation

Definition at line 112 of file FeatureContainer.h.

Referenced by `FeatureContainer()`, `getFeatureVector()`, `loadFeature()`, and `nextSample()`.

#### 6.22.5.3 `vector<void*>` `FeatureContainer::handle` [private]

Private property.

##### Todo

documentation

Definition at line 50 of file FeatureContainer.h.

Referenced by `freeProvider()`, `loadFeature()`, and `loadProvider()`.

#### 6.22.5.4 `FILE*` `FeatureContainer::out` [private]

Private property.

##### Todo

documentation

Reimplemented in [LoggingFeatureContainer](#).

Definition at line 53 of file FeatureContainer.h.

#### 6.22.5.5 `string` `FeatureContainer::persist` [private]

Private property.

##### Todo

documentation

Definition at line 49 of file FeatureContainer.h.

Referenced by `FeatureContainer()`, and `nextSample()`.

**6.22.5.6** [providermap FeatureContainer::provider](#) [private]

Private property.

**Todo**

documentation

Definition at line 51 of file FeatureContainer.h.

Referenced by FeatureContainer(), loadFeature(), and nextSample().

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

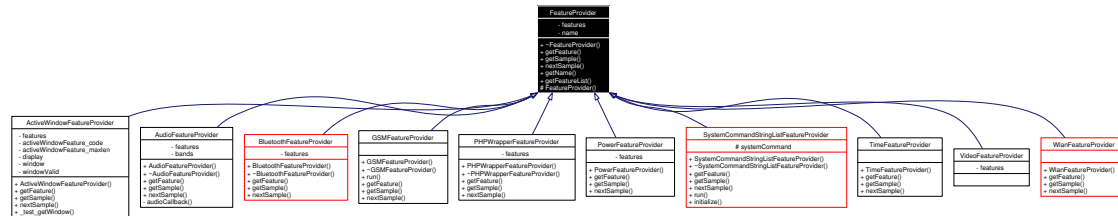
- [FeatureContainer.h](#)
- [FeatureContainer.cpp](#)
- [LinuxFeatureContainer.cpp](#)
- [WindowsFeatureContainer.cpp](#)

## 6.23 FeatureProvider Class Reference

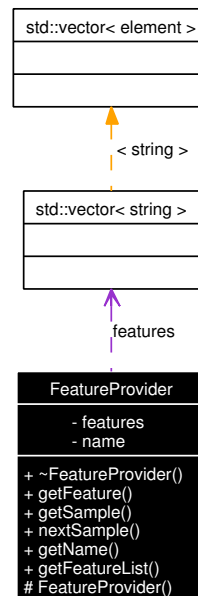
FeatureProvider interface.

```
#include <Feature.h>
```

Inheritance diagram for FeatureProvider:



Collaboration diagram for FeatureProvider:



### Public Member Functions

- virtual [~FeatureProvider](#) ()  
*Destructor.*
- virtual [Feature](#) \* [getFeature](#) (string [name](#)) const =0  
*Query feature instance.*
- virtual [Feature](#) \* [getSample](#) (string [name](#)) const =0  
*Query sample instance.*
- virtual void [nextSample](#) (clock\_t checkpoint)=0

*Prepare sample.*

- string `getName ()` const  
*Query provider name.*
- `stringvector * getFeatureList ()` const  
*Query list of provided features.*

## Protected Member Functions

- `FeatureProvider (string name, stringvector *features)`  
*Constructor.*

## Private Attributes

- `stringvector * features`  
*The list of feature names provided.*
- string `name`  
*The name of the feature provider itself.*

### 6.23.1 Detailed Description

FeatureProvider interface.

Each feature that is used in the cluster space definition is provided by a feature provider. A feature provider can provide more than one feature and will be referred to by a unique name. A feature can then be unambiguously identified by a combination of its own name and the name of its feature provider. Thus the actual cluster space can be defined in a configuration file and the clustering infrastructure can fetch the according features based on that definition.

See also:

[Feature](#)  
[PersistantFeature](#)

Definition at line 413 of file Feature.h.

### 6.23.2 Constructor & Destructor Documentation

#### 6.23.2.1 `FeatureProvider::FeatureProvider (string name, stringvector * features) [inline, protected]`

Constructor.

Initializes the names of the provider and its features.

Parameters:

*name* Name of the provider.



*features* List of featurenames provided.

Definition at line 429 of file Feature.h.

#### 6.23.2.2 virtual FeatureProvider::~~FeatureProvider () [inline, virtual]

Destructor.

Empty virtual destructor to have an entry in the implementing classes method table. This ensures that derived classes get destructed correctly. Definition at line 442 of file Feature.h.

Referenced by SystemCommandStringListFeatureProvider::~~SystemCommandStringListFeatureProvider().

### 6.23.3 Member Function Documentation

#### 6.23.3.1 virtual Feature\* FeatureProvider::getFeature (string name) const [pure virtual]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

##### Parameters:

*name* Name of the requested feature.

##### Returns:

An instance of the requested feature.

Implemented in [ActiveWindowFeatureProvider](#), [AudioFeatureProvider](#), [BluetoothFeatureProvider](#), [GSMFeatureProvider](#), [PHPWrapperFeatureProvider](#), [PowerFeatureProvider](#), [SystemCommandStringListFeatureProvider](#), [TimeFeatureProvider](#), [VideoFeatureProvider](#), and [WlanFeatureProvider](#).

#### 6.23.3.2 stringvector\* FeatureProvider::getFeatureList () const [inline]

Query list of provided features.

##### Returns:

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by FeatureContainer::loadFeature().

#### 6.23.3.3 string FeatureProvider::getName () const [inline]

Query provider name.

##### Returns:

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

#### 6.23.3.4 **virtual [Feature\\*](#) FeatureProvider::getSample (string *name*) const** [pure virtual]

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the sample.

**See also:**

[nextSample](#)

Implemented in [ActiveWindowFeatureProvider](#), [AudioFeatureProvider](#), [BluetoothFeatureProvider](#), [BluetoothLinuxFeatureProvider](#), [GSMFeatureProvider](#), [PHPWrapperFeatureProvider](#), [PowerFeatureProvider](#), [SystemCommandStringListFeatureProvider](#), [TimeFeatureProvider](#), [VideoFeatureProvider](#), [WlanFeatureProvider](#), and [WlanLinuxFeatureProvider](#).

#### 6.23.3.5 **virtual void FeatureProvider::nextSample (clock\_t *checkpoint*)** [pure virtual]

Prepare sample.

Prepare the feature provider to return the next sample from the sensor when [getSample\(\)](#) is called.

**Parameters:**

*checkpoint* This parameter tells the feature extractor that it must not return sensordata sampled earlier than the given clock value.

**See also:**

[getSample](#)

Implemented in [ActiveWindowFeatureProvider](#), [AudioFeatureProvider](#), [BluetoothFeatureProvider](#), [GSMFeatureProvider](#), [PHPWrapperFeatureProvider](#), [PowerFeatureProvider](#), [SystemCommandStringListFeatureProvider](#), [TimeFeatureProvider](#), [VideoFeatureProvider](#), [WlanFeatureProvider](#), and [WlanLinuxFeatureProvider](#).

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

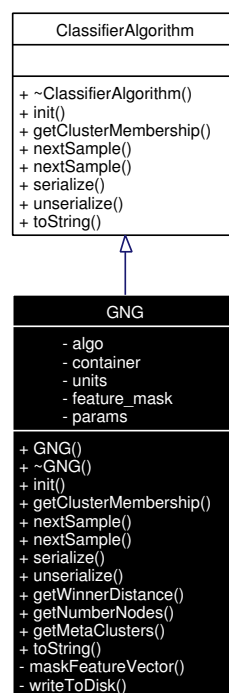
- [Feature.h](#)

## 6.24 GNG Class Reference

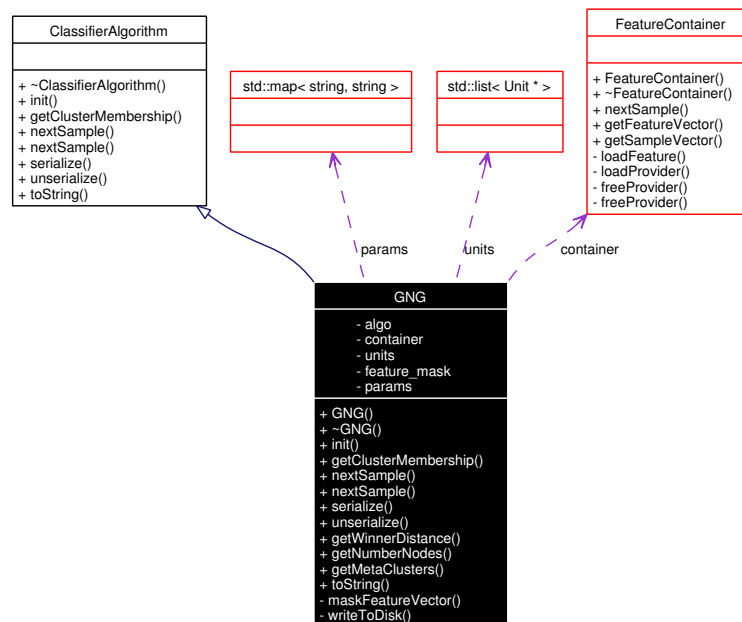
This is lifelong GNG.

```
#include <GNG.h>
```

Inheritance diagram for GNG:



Collaboration diagram for GNG:



## Public Member Functions

- **GNG** (`classifierparams &params`)  
Only stores the parameters for later use in `init()`.
- virtual `~GNG ()`  
Destructor.
- virtual void `init (FeatureContainer *fc)`  
This method initializes the helper variables, creates two initial cluster center prototypes by calling `getFeatureVector` on the given `FeatureContainer` and stored the `FeatureContainer` reference for calls to `nextSample()`.
- virtual `membershiplist getClusterMembership (const featurevector *sample)`  
Get membership vector.
- virtual unsigned long `nextSample ()`  
Fetch next sample vector.
- virtual unsigned long `nextSample (const featurevector *sample)`
- virtual string `serialize () const`  
Serialize a samples data to a string.
- virtual void `unserialize (string data)`  
Unserialize a samples data from a string.
- virtual double `getWinnerDistance (const featurevector *sample, unsigned int &cluster_id, unsigned int &meta_cluster_id) const`
- virtual `unit_list::size_type getNumberNodes () const`

- virtual [meta\\_cluster\\_list](#) getMetaClusters ()
- virtual string [toString](#) () const

*This is only for testing.*

## Static Public Attributes

- unsigned [lambda](#) = 10  
*insert a new unit after ( $\lambda * \text{<number of units>}$ ) samples*
- double [minimal\\_deletion\\_age](#) = 0.5  
*only delete units which are older than that (their youth must be lower than this value)*
- double [deletion\\_threshold](#) = 0.5  
*the threshold for the deletion criterion (it must be lower than this for a node to get deleted)*
- double [deletion\\_threshold\\_lq](#) = 0.15  
*the threshold for the quality measure for learning when deleting a node (it must be lower than this for a node to get deleted)*
- unsigned long [edge\\_age\\_max](#) = 30  
*maximum age of an edge*
- double [T\\_short](#) = 6  
*time constant for exponential decrease of the short term error of a winning unit*
- double [T\\_long](#) = 170  
*time constant for exponential decrease of the long term error of a winning unit*
- double [T\\_youth](#) = 50  
*time constant for exponential decrease of the youth*
- double [T\\_insert\\_threshold](#) = 55  
*insert a new unit after ( $\lambda * \text{<number of units>}$ ) samples*
- double [adaptation\\_threshold](#) = 0.0005  
*insert a new unit after ( $\lambda * \text{<number of units>}$ ) samples*
- double [insertion\\_tolerance](#) = 0.02  
*insert a new unit after ( $\lambda * \text{<number of units>}$ ) samples*
- double [lr\\_winner](#) = 0.15  
*the learning rate of the winning unit*
- double [lr\\_neighbor](#) = 0.01  
*the learning rate of all neighbors*
- double [minimum\\_lr](#) = 0.2  
*the absolute minimum learning rate*

- double `lr_insert_threshold` = 0.2  
*the learning rate for the insertion threshold*
- double `helper_T_short` = -1  
*insert a new unit after ( $\lambda * \text{<number of units>}$ ) samples*
- double `helper_T_long` = -1  
*insert a new unit after ( $\lambda * \text{<number of units>}$ ) samples*
- double `helper_T_youth` = -1  
*insert a new unit after ( $\lambda * \text{<number of units>}$ ) samples*
- double `helper_T_insertion_threshold` = -1  
*insert a new unit after ( $\lambda * \text{<number of units>}$ ) samples*

## Private Member Functions

- `featurevector maskFeatureVector` (const `featurevector` \*sample, bool freeUnused=false) const  
*This is a helper variable to generate a feature vector that only contains those dimensions which are set to 1 in the feature mask.*
- void `writeToDisk` ()

## Private Attributes

- const char \* `algo`
- `FeatureContainer` \* `container`  
*The featurecontainer from which the parameter-less `nextSample()` implementation will fetch samples whenever called.*
- `unit_list` units  
*The list of units in this instance of GNG.*
- bit\_vector `feature_mask`  
*This bitvector has the same length as the feature vectors and defines for each dimension if it is used for classification.*
- const `classifierparams` `params`  
*This is just a copy of the algorithm parameters passed to the constructor and used by init for constructing the feature mask.*

### 6.24.1 Detailed Description

This is lifelong GNG.

It is basically an implementation of Fred H. Hamker: "Life-long learning Cell Structures—continuously learning without catastrophic interference" Neural Networks 12 (2001) 551–573

**Todo**

documentation

Definition at line 54 of file GNG.h.

## 6.24.2 Constructor & Destructor Documentation

### 6.24.2.1 GNG::GNG ([classifierparams](#) & *params*)

Only stores the parameters for later use in [init\(\)](#).

**Parameters:**

*params* Initialization parameters

**See also:**

[init](#)

Definition at line 61 of file GNG.cpp.

References [lr\\_winner](#).

## 6.24.3 Member Function Documentation

### 6.24.3.1 [membershiplist](#) GNG::getClusterMembership (const [featurevector](#) \* *sample*) [virtual]

Get membership vector.

This method returns the data for the next steps, which is a list of context classes found by the algorithm with associated membership values. These values specify the membership of the current feature values to each of the context classes in the interval [0; 1]. They can thus be seen as the probabilities that the context classes are currently active.

**Parameters:**

*sample* The sample vector for which the membership list should be calculated

Implements [ClassifierAlgorithm](#).

Definition at line 450 of file GNG.cpp.

References [units](#).

### 6.24.3.2 [meta\\_cluster\\_list](#) GNG::getMetaClusters () [virtual]

Definition at line 339 of file GNG.cpp.

References [units](#).

Referenced by [Scanner::run\(\)](#).

### 6.24.3.3 [unit\\_list::size\\_type](#) GNG::getNumberNodes () const [virtual]

**Todo**

documentation

Definition at line 334 of file GNG.cpp.

References meta\_cluster\_list.

**6.24.3.4** `double GNG::getWinnerDistance (const featurevector * sample, unsigned int & cluster_id, unsigned int & meta_cluster_id) const` [virtual]

**Todo**

documentation

Definition at line 312 of file GNG.cpp.

References featurevector, and units.

Referenced by evaluate().

**6.24.3.5** `void GNG::init (FeatureContainer *fc)` [virtual]

This method initializes the helper variables, creates two initial cluster center prototypes by calling `getFeatureVector` on the given [FeatureContainer](#) and stored the [FeatureContainer](#) reference for calls to [nextSample\(\)](#).

Additionally, it initializes the bit vector for masking feature dimensions by using the given algorithm parameters and the feature dimensions from the retrieved feature vectors.

**See also:**

[nextSample\(\)](#)

Implements [ClassifierAlgorithm](#).

Definition at line 66 of file GNG.cpp.

References container, fc, feature\_mask, featurevector, helper\_T\_insertion\_threshold, helper\_T\_long, helper\_T\_short, helper\_T\_youth, params, and units.

**6.24.3.6** `featurevector GNG::maskFeatureVector (const featurevector * sample, bool freeUnused = false) const` [private]

This is a helper variable to generate a feature vector that only contains those dimensions which are set to 1 in the feature mask.

The pointers to used feature dimensions are copied to the returned feature vector, the unused feature dimensions (those which are set to 0 in the feature mask) are freed if the second parameter is true.

**Parameters:**

*sample* The input feature vector.

*freeUnused* If true, then unused feature dimensions will be freed. Use with care.

**Returns:**

The reduced feature vector conforming to the mask.

**See also:**

[feature\\_mask](#)

Definition at line 509 of file GNG.cpp.

References feature\_mask.



**6.24.3.7 unsigned long GNG::nextSample (const [featurevector](#) \* *inSample*)** [virtual]**Todo**

cope with increasing dimension of the feature vector ?

**Todo**

this should be done in a better way

Implements [ClassifierAlgorithm](#).

Definition at line 169 of file GNG.cpp.

References [featurevector](#), and [units](#).

**6.24.3.8 unsigned long GNG::nextSample ()** [virtual]

Fetch next sample vector.

Gets the next sample vector from the configured feature source. This method calls [nextSample](#) and [getSampleVector](#) on the [FeatureContainer](#) given to the [init](#) method.

**See also:**

[FeatureContainer::nextSample](#)

[FeatureContainer::getSampleVector](#)

Implements [ClassifierAlgorithm](#).

Definition at line 158 of file GNG.cpp.

**6.24.3.9 string GNG::serialize () const** [virtual]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [ClassifierAlgorithm](#).

Definition at line 365 of file GNG.cpp.

References [units](#).

**6.24.3.10 void GNG::unserialize (string *data*)** [virtual]

Unserialize a samples data from a string.

**Parameters:**

*data* String representation of the samples data.

Implements [ClassifierAlgorithm](#).

Definition at line 397 of file GNG.cpp.

## 6.24.4 Member Data Documentation

### 6.24.4.1 double [GNG::adaptation\\_threshold](#) = 0.0005 [static]

insert a new unit after (lambda \* <number of units>) samples

#### Todo

documentation

Definition at line 48 of file GNG.cpp.

Referenced by evaluate().

### 6.24.4.2 [FeatureContainer\\*](#) [GNG::container](#) [private]

The featurecontainer from which the parameter-less [nextSample\(\)](#) implementation will fetch samples whenever called.

#### See also:

[nextSample\(\)](#)

Definition at line 146 of file GNG.h.

Referenced by init().

### 6.24.4.3 [bit\\_vector](#) [GNG::feature\\_mask](#) [private]

This bitvector has the same length as the feature vectors and defines for each dimension if it is used for classification.

By setting it to 0, those dimensions are effectively ignored by GNG and are not even passed to the methods of the [Unit](#) class. This vector is initialized by the init method (when the feature dimensions are known). The feature mask is honored by all methods which accepts a feature vector as input parameter namely nextSample, getClusterMembership and the helper methods getWinnerDistance and getMetaClusters. Definition at line 170 of file GNG.h.

Referenced by init(), and maskFeatureVector().

### 6.24.4.4 double [GNG::helper\\_T\\_insertion\\_threshold](#) = -1 [static]

insert a new unit after (lambda \* <number of units>) samples

#### Todo

documentation

Definition at line 59 of file GNG.cpp.

Referenced by init().

### 6.24.4.5 double [GNG::helper\\_T\\_long](#) = -1 [static]

insert a new unit after (lambda \* <number of units>) samples

**Todo**

documentation

Definition at line 57 of file GNG.cpp.

Referenced by init().

**6.24.4.6 double GNG::helper\_T\_short = -1 [static]**

insert a new unit after (lambda \* <number of units>) samples

**Todo**

documentation

Definition at line 56 of file GNG.cpp.

Referenced by init().

**6.24.4.7 double GNG::helper\_T\_youth = -1 [static]**

insert a new unit after (lambda \* <number of units>) samples

**Todo**

documentation

Definition at line 58 of file GNG.cpp.

Referenced by init().

**6.24.4.8 double GNG::insertion\_tolerance = 0.02 [static]**

insert a new unit after (lambda \* <number of units>) samples

**Todo**

documentation

Definition at line 49 of file GNG.cpp.

Referenced by evaluate().

**6.24.4.9 unsigned GNG::lambda = 10 [static]**

insert a new unit after (lambda \* <number of units>) samples

**Todo**

documentation

Definition at line 36 of file GNG.cpp.

Referenced by evaluate().

**6.24.4.10** `const classifierparams GNG::params` [private]

This is just a copy of the algorithm parameters passed to the constructor and used by `init` for constructing the feature mask.

**See also:**

`GNG`  
`init`  
`feature_mask`

Definition at line 180 of file `GNG.h`.

Referenced by `init()`.

**6.24.4.11** `double GNG::T_insert_threshold = 55` [static]

insert a new unit after  $(\lambda * \text{<number of units>})$  samples

**Todo**

documentation

Definition at line 47 of file `GNG.cpp`.

Referenced by `evaluate()`.

**6.24.4.12** `unit_list GNG::units` [private]

The list of units in this instance of `GNG`.

This is the authoritative list of all units used by `GNG`, other lists (e.g. the `all_units` pointer in `Unit` or the edge map in `Unit`) may only contain pointers to units stored in this list.

**See also:**

`Unit::edge`  
`Unit::all_units`

Definition at line 157 of file `GNG.h`.

Referenced by `getClusterMembership()`, `getMetaClusters()`, `getWinnerDistance()`, `init()`, `nextSample()`, and `serialize()`.

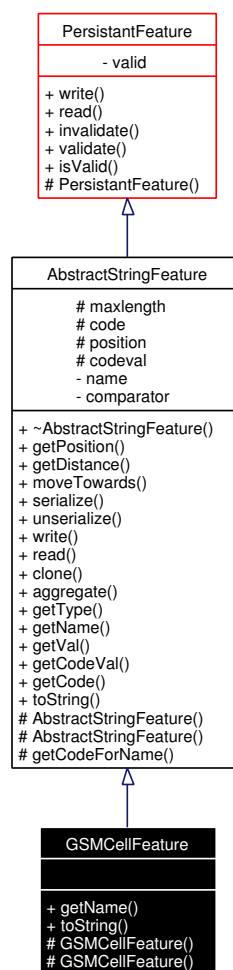
The documentation for this class was generated from the following files:

- `GNG.h`
- `GNG.cpp`

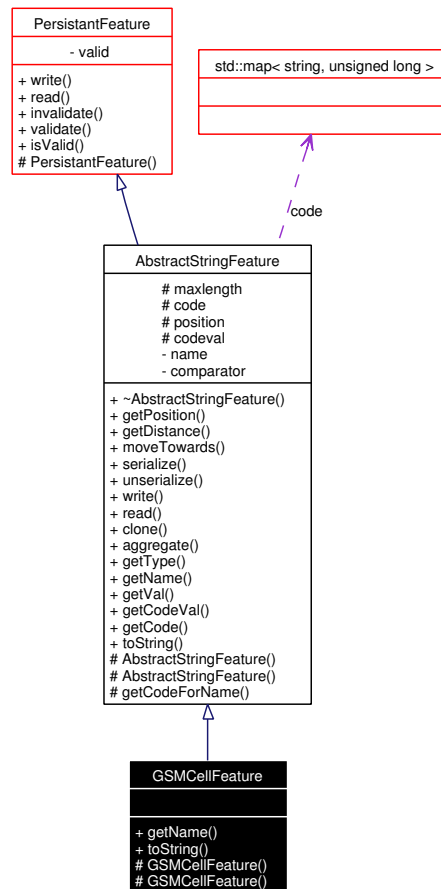
## 6.25 GSMCellFeature Class Reference

```
#include <GSM.h>
```

Inheritance diagram for GSMCellFeature:



Collaboration diagram for GSMCellFeature:



## Public Types

- enum [ComparatorType](#) { **None**, **Levenshtein** }

*The distance metric to use.*

- enum [FeatureType](#) {  
**boolean**, **nominal**, **ordinal**, **numerical\_discrete**,  
**numerical\_continuous** }

*Possible feature types.*

## Public Member Functions

- virtual const string [getName](#) () const  
*Query a features name.*
- virtual string [toString](#) () const  
*Get feature as string.*
- virtual double [getPosition](#) () const

*Query a features position.*

- virtual double `getDistance` (`Feature *f`) const  
*Calculates the distance between two features.*
- virtual void `moveTowards` (`Feature *f`, double factor)  
*Move feature.*
- virtual string `serialize` () const  
*Serialize a samples data to a string.*
- virtual void `unserialize` (string value)  
*Unserialize a samples data from a string.*
- virtual `featureparams write` () const  
*Externalize feature.*
- virtual void `read` (`featureparams *param`)  
*Load feature from persistant data.*
- virtual `Feature * clone` () const  
*Clone a feature.*
- virtual void `aggregate` (`aggregatelist` samples)  
*Aggregate a sample values from other sources.*
- virtual `FeatureType getType` () const  
*Query a features type.*
- const string & `getVal` () const  
*Query the string values.*
- const unsigned long `getCodeVal` () const
- const `stringcode * getCode` ()
- void `invalidate` ()  
*Invalidate feature.*
- void `validate` ()  
*Set the validation flag to true.*
- bool `isValid` ()  
*Query validation flag.*
- bool `isExternalizable` ()  
*Query externalization flag.*

## Protected Member Functions

- [GSMCellFeature](#) ([stringcode](#) \*code, long \*maxlen)  
*Feature constructor*
- [GSMCellFeature](#) ([stringcode](#) \*code, long \*maxlen, string cellid)  
*Sample constructor.*
- unsigned long [getCodeForName](#) (const string &name)  
*Get a code value for a given string.*

## Protected Attributes

- long \* [maxlength](#)  
*A reference to the static, persistant maximum length of strings seen so far.*
- [stringcode](#) \* [code](#)  
*A reference to the static, persistant code table of strings seen so far.*
- char \* [position](#)  
*Only for internal usage in `moveTowards` and `getDistance`.*
- unsigned long [codeval](#)  
*The coded value of the feature.*
- const bool [externalize](#)  
*Externalization flag.*

## Related Functions

(Note that these are not member functions.)

- long [levenshtein](#) (char \*\*x, const char \*t, double \*factor)  
*Modified Levenshtein algorithm.*

### 6.25.1 Detailed Description

#### [Todo](#)

documentation

Definition at line 36 of file GSM.h.



## 6.25.2 Constructor & Destructor Documentation

### 6.25.2.1 GSMCellFeature::GSMCellFeature ([stringcode](#) \* *code*, long \* *maxlen*) [[inline](#), [protected](#)]

Feature constructor

This constructor initializes the feature as prototypes value. For creating a specific sample of a feature, the sample constructor should be used.

#### Parameters:

*code* A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the code table.

*maxlen* A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum string length.

Definition at line 55 of file GSM.h.

References [stringcode](#).

### 6.25.2.2 GSMCellFeature::GSMCellFeature ([stringcode](#) \* *code*, long \* *maxlen*, [string](#) *cellid*) [[inline](#), [protected](#)]

Sample constructor.

This constructor initializes the feature with a specific value and should thus be used for samples.

#### Parameters:

*code* A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the code table.

*maxlen* A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum string length.

*cellid* The string representation of the sample value.

Definition at line 71 of file GSM.h.

References [stringcode](#).

## 6.25.3 Member Function Documentation

### 6.25.3.1 void AbstractStringFeature::aggregate ([aggregatelist](#) *samples*) [[virtual](#), [inherited](#)]

Aggregate a sample values from other sources.

#### Parameters:

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 421 of file AbstractString.cpp.

References [aggregatelist](#).

### 6.25.3.2 **Feature** \* **AbstractStringFeature::clone () const** [virtual, inherited]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 403 of file `AbstractString.cpp`.

References `AbstractStringFeature::AbstractStringFeature()`, `AbstractStringFeature::code`, `AbstractStringFeature::codeval`, `AbstractStringFeature::comparator`, `AbstractStringFeature::maxlength`, `AbstractStringFeature::name`, and `AbstractStringFeature::position`.

### 6.25.3.3 **unsigned long AbstractStringFeature::getCodeForName (const string & name)** [protected, inherited]

Get a code value for a given string.

A helper function to return the respective code for a given string. If the string is encountered the first time a new code is allocated.

#### Parameters:

*name* The string to code

#### Returns:

A code value

Definition at line 303 of file `AbstractString.cpp`.

References `AbstractStringFeature::code`, `PersistentFeature::invalidate()`, and `AbstractStringFeature::maxlength`.

Referenced by `AbstractStringFeature::AbstractStringFeature()`, and `AbstractStringListFeature::getCodeForList()`.

### 6.25.3.4 **double AbstractStringFeature::getDistance (Feature \* f) const** [virtual, inherited]

Calculates the distance between two features.

#### Parameters:

*f* [Feature](#) used for distance measurement.

#### Returns:

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 337 of file `AbstractString.cpp`.

References `AbstractStringFeature::code`, `AbstractStringFeature::codeval`, `AbstractStringFeature::comparator`, `AbstractStringFeature::levenshtein()`, `AbstractStringFeature::maxlength`, and `AbstractStringFeature::position`.

**6.25.3.5 virtual const string GSMCellFeature::getName () const** [inline, virtual]

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Reimplemented from [AbstractStringFeature](#).

Definition at line 74 of file GSM.h.

Referenced by toupperstr().

**6.25.3.6 double AbstractStringFeature::getPosition () const** [virtual, inherited]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 323 of file AbstractString.cpp.

References AbstractStringFeature::maxlength, and AbstractStringFeature::position.

**6.25.3.7 virtual FeatureType AbstractStringFeature::getType () const** [inline, virtual, inherited]

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 100 of file AbstractString.h.

**6.25.3.8 const string& AbstractStringFeature::getVal () const** [inline, inherited]

Query the string values.

**Returns:**

Values list

Definition at line 108 of file AbstractString.h.

References AbstractStringFeature::name.

Referenced by Java\_at\_jku\_intelligence\_samples\_StringSample\_nativeGetVal().

**6.25.3.9 void PersistentFeature::invalidate () [inline, inherited]**

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by AbstractStringFeature::getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature::NumericalDiscreteFeature(), TimeFeature::TimeFeature(), NumericalContinuousFeature::unserialize(), and NumericalDiscreteFeature::unserialize().

**6.25.3.10 bool Feature::isExternalizable () [inline, inherited]**

Query externalization flag.

**Returns:**

*true* if the feature is externalizable ( i.e. it has persistent data that should be stored across restarts), *false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.25.3.11 bool PersistentFeature::isValid () [inline, inherited]**

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

**6.25.3.12 void AbstractStringFeature::moveTowards (Feature \*f, double factor) [virtual, inherited]**

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample s.

**Parameters:**

*f* Feature used for distance measurement.

*factor* Distance weight.

Implements Feature.

Reimplemented in AbstractStringListFeature.

Definition at line 371 of file AbstractString.cpp.

References AbstractStringFeature::code, AbstractStringFeature::codeval, AbstractStringFeature::comparator, AbstractStringFeature::levenshtein(), AbstractStringFeature::position, and rand\_double.

**6.25.3.13** `void AbstractStringFeature::read (featureparams * param)` [virtual, inherited]

Load feature from persistant data.

Initializes the persistant feature data from the given representation.

**Parameters:**

*param* Persistant feature data.

**See also:**

[write](#)

Implements [PersistantFeature](#).

Definition at line 296 of file AbstractString.cpp.

References AbstractStringFeature::code, and featureparams.

**6.25.3.14** `string AbstractStringFeature::serialize () const` [virtual, inherited]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 228 of file AbstractString.cpp.

References AbstractStringFeature::codeval, AbstractStringFeature::comparator, and AbstractStringFeature::position.

Referenced by WlanActiveMacAddressFeature::toString(), WlanActiveEssidFeature::toString(), AbstractStringFeature::toString(), and toupperstr().

**6.25.3.15** `string GSMCellFeature::toString () const` [virtual]

Get feature as string.

**Note:**

This is only for testing.

**Returns:**

[Feature](#) as string.

Reimplemented from [AbstractStringFeature](#).

Definition at line 60 of file GSM.cpp.

**6.25.3.16** `void AbstractStringFeature::unserialize (string value)` [virtual, inherited]

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 251 of file AbstractString.cpp.

References [AbstractStringFeature::code](#), [AbstractStringFeature::codeval](#), [AbstractStringFeature::comparator](#), [AbstractStringFeature::maxlength](#), and [AbstractStringFeature::position](#).

### 6.25.3.17 [featureparams](#) AbstractStringFeature::write () const [virtual, inherited]

Externalize feature.

**Returns:**

Persistent feature data.

**See also:**

[read](#)

Implements [PersistentFeature](#).

Definition at line 283 of file AbstractString.cpp.

References [AbstractStringFeature::code](#), and [featureparams](#).

## 6.25.4 Friends And Related Function Documentation

### 6.25.4.1 [long levenshtein](#) (char \*\* *x*, const char \* *t*, double \* *factor*) [related, inherited]

Modified Levenshtein algorithm.

This function implements a slightly modified version of the Levenshtein algorithm to not only calculate the distance of two strings but also modify the string *x* towards the string *t* with the probability *factor*. In other words, every transformation operation found by the algorithm is performed on the string *x* with the probability *factor*.

**Parameters:**

*x* Input string.

*t* String to compare the input string to.

*factor* Propability with witch transformations are performed. Has to be a value out of the intervall  $[0; 1]$ .

**Returns:**

The levenshtein distance between *x* and *t*.

**Todo**

alloc once

Definition at line 46 of file AbstractString.cpp.

References [rand\\_double](#).

Referenced by [AbstractStringFeature::getDistance\(\)](#), and [AbstractStringFeature::moveTowards\(\)](#).

## 6.25.5 Member Data Documentation

### 6.25.5.1 unsigned long [AbstractStringFeature::codeval](#) [protected, inherited]

The coded value of the feature.

See also:

[name](#)  
[code](#)

Definition at line 144 of file [AbstractString.h](#).

Referenced by [AbstractStringFeature::AbstractStringFeature\(\)](#), [AbstractStringFeature::clone\(\)](#), [AbstractStringFeature::getCodeVal\(\)](#), [AbstractStringFeature::getDistance\(\)](#), [AbstractStringFeature::moveTowards\(\)](#), [AbstractStringFeature::serialize\(\)](#), and [AbstractStringFeature::unserialize\(\)](#).

### 6.25.5.2 const bool [Feature::externalize](#) [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to [PersistentFeature](#), because only subclasses of this type are (by policy) allowed to set this variable to true.

See also:

[PersistentFeature](#)

Definition at line 187 of file [Feature.h](#).

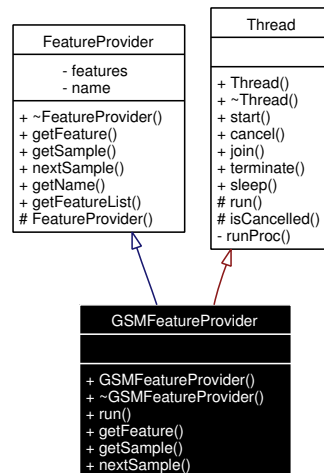
The documentation for this class was generated from the following files:

- [GSM.h](#)
- [GSM.cpp](#)

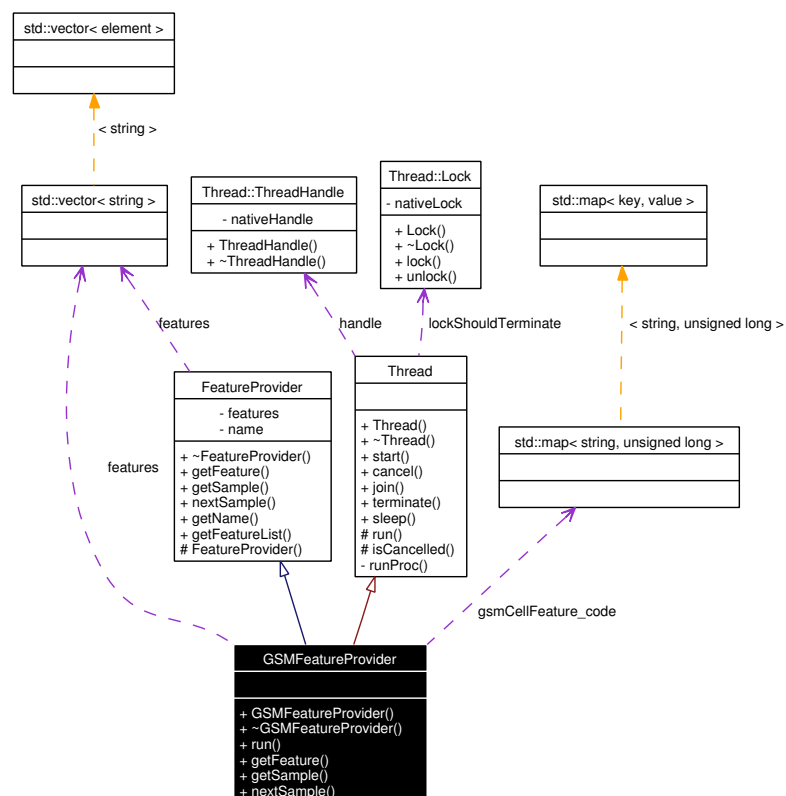
## 6.26 GSMFeatureProvider Class Reference

```
#include <GSM.h>
```

Inheritance diagram for GSMFeatureProvider:



Collaboration diagram for GSMFeatureProvider:





**[NOHEADER]**

- void [init](#) ([featureparams](#) &params)
- virtual bool [openSerialPort](#) (string serialport)
- virtual void [closeSerialPort](#) ()
- virtual bool [writeLine](#) (string line)
- virtual string [readLine](#) ()
- [stringvector](#) [features](#)
- int [delay](#)
- [stringcode](#) [gsmCellFeature\\_code](#)
- long [gsmCellFeature\\_maxlen](#)

**Public Member Functions**

- [GSMFeatureProvider](#) ([providerparams](#) &params)  
*Constructor.*
- virtual [~GSMFeatureProvider](#) ()  
*Destructor.*
- virtual void [run](#) ()  
*Thread main worker function.*
- virtual [Feature](#) \* [getFeature](#) (string [name](#)) const  
*Query feature instance.*
- virtual [Feature](#) \* [getSample](#) (string [name](#)) const  
*Query sample instance.*
- virtual void [nextSample](#) (clock\_t checkpoint)  
*Prepare sample.*
- string [getName](#) () const  
*Query provider name.*
- [stringvector](#) \* [getFeatureList](#) () const  
*Query list of provided features.*

**Protected Attributes**

- string [cellId](#)
- bool [cellIdValid](#)
- Lock [lockCellId](#)
- string [serialPort](#)

## Private Member Functions

- void [start](#) ()  
*Start thread execution.*
- void [cancel](#) ()  
*End thread execution.*
- void [join](#) ()  
*Join thread.*
- void [terminate](#) ()  
*Terminate thread execution.*
- bool [isCancelled](#) ()

## Static Private Member Functions

- void [sleep](#) (unsigned milliseconds)  
*sleep function*

### 6.26.1 Detailed Description

[Todo](#)  
documentation

Definition at line 85 of file GSM.h.

### 6.26.2 Constructor & Destructor Documentation

#### 6.26.2.1 GSMFeatureProvider::GSMFeatureProvider ([providerparams](#) & *params*)

Constructor.

**Parameters:**  
*params* Map of initialization parameters

[Todo](#)  
check parameter !!

Definition at line 69 of file GSM.cpp.

### 6.26.3 Member Function Documentation

#### 6.26.3.1 void GSMFeatureProvider::closeSerialPort () [private, virtual]

Definition at line 87 of file GSMLinux.cpp.

References `portFd`, and `term_old`.

Referenced by `run()`.

**6.26.3.2** [Feature](#) \* GSMFeatureProvider::getFeature (string *name*) const [virtual]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the requested feature.

Implements [FeatureProvider](#).

Definition at line 116 of file GSM.cpp.

**6.26.3.3** [stringvector](#)\* FeatureProvider::getFeatureList () const [inline, inherited]

Query list of provided features.

**Returns:**

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by FeatureContainer::loadFeature().

**6.26.3.4** string FeatureProvider::getName () const [inline, inherited]

Query provider name.

**Returns:**

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

**6.26.3.5** [Feature](#) \* GSMFeatureProvider::getSample (string *name*) const [virtual]

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the sample.

**See also:**

[nextSample](#)

Implements [FeatureProvider](#).

Definition at line 103 of file GSM.cpp.

References cellId, cellIdValid, gsmCellFeature\_code, and gsmCellFeature\_maxlen.

**6.26.3.6 void GSMFeatureProvider::init (featureparams & params) [private]**

Definition at line 38 of file GSMLinux.cpp.

References featureparams, GSM\_SERIALPORT, and serialPort.

**6.26.3.7 bool Thread::isCancelled () [protected, inherited]****Returns:**

*true* if the thread is cancelled, *false* if the thread is still running

Definition at line 43 of file Thread.cpp.

References Thread::cancelled.

Referenced by nextSample(), WlanLinuxFeatureProvider::run(), SystemCommandStringListFeatureProvider::run(), and BluetoothLinuxFeatureProvider::run().

**6.26.3.8 void GSMFeatureProvider::nextSample (clock\_t checkpoint) [virtual]**

Prepare sample.

Prepare the feature provider to return the next sample from the sensor when [getSample\(\)](#) is called.

**Parameters:**

*checkpoint* This parameter tells the feature extractor that it must not return sensordata sampled earlier than the given clock value.

**See also:**

[getSample](#)

Implements [FeatureProvider](#).

Definition at line 125 of file GSM.cpp.

References Thread::isCancelled().

**6.26.3.9 bool GSMFeatureProvider::openSerialPort (string serialport) [private, virtual]**

name of the serial port Definition at line 48 of file GSMLinux.cpp.

References portFd, and term\_old.

**6.26.3.10 string GSMFeatureProvider::readLine () [private, virtual]**

Definition at line 109 of file GSMLinux.cpp.

References portFd, and Thread::sleep().

Referenced by run().

**6.26.3.11 void Thread::sleep (unsigned milliseconds) [static, inherited]**

sleep function

**Parameters:**

*milliseconds* Timeout in milliseconds

Definition at line 80 of file ThreadPosix.cpp.

Referenced by Main(), readLine(), WlanLinuxFeatureProvider::run(), SystemCommandStringListFeatureProvider::run(), Scanner::run(), and BluetoothLinuxFeatureProvider::run().

**6.26.3.12 bool GSMFeatureProvider::writeLine (string line) [private, virtual]**

Definition at line 96 of file GSMLinux.cpp.

References portFd.

Referenced by run().

**6.26.4 Member Data Documentation****6.26.4.1 string GSMFeatureProvider::cellId [protected]****Todo**

documentation

Definition at line 106 of file GSM.h.

Referenced by getSample(), and run().

**6.26.4.2 bool GSMFeatureProvider::cellIdValid [protected]****Todo**

documentation

Definition at line 107 of file GSM.h.

Referenced by getSample(), and run().

**6.26.4.3 int GSMFeatureProvider::delay [private]**

Definition at line 91 of file GSM.h.

**6.26.4.4 stringvector GSMFeatureProvider::features [private]****Todo**

documentation

Reimplemented from [FeatureProvider](#).

Definition at line 90 of file GSM.h.

**6.26.4.5 stringcode GSMFeatureProvider::gsmCellFeature\_code [private]**

Definition at line 92 of file GSM.h.

Referenced by getSample().

**6.26.4.6**   **long** [GSMFeatureProvider::gsmCellFeature\\_maxlen](#)   [private]

Definition at line 93 of file GSM.h.

Referenced by `getSample()`.

**6.26.4.7**   **Lock** [GSMFeatureProvider::lockCellId](#)   [mutable, protected]**Todo**

documentation

Definition at line 108 of file GSM.h.

Referenced by `run()`.

**6.26.4.8**   **string** [GSMFeatureProvider::serialPort](#)   [protected]**Todo**

documentation

Definition at line 110 of file GSM.h.

Referenced by `init()`.

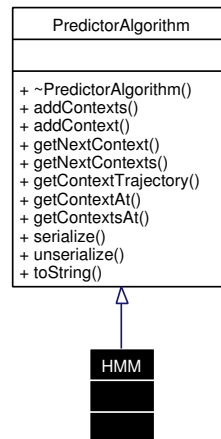
The documentation for this class was generated from the following files:

- [GSM.h](#)
- [GSM.cpp](#)
- [GSMLinux.cpp](#)
- [GSMWindows.cpp](#)

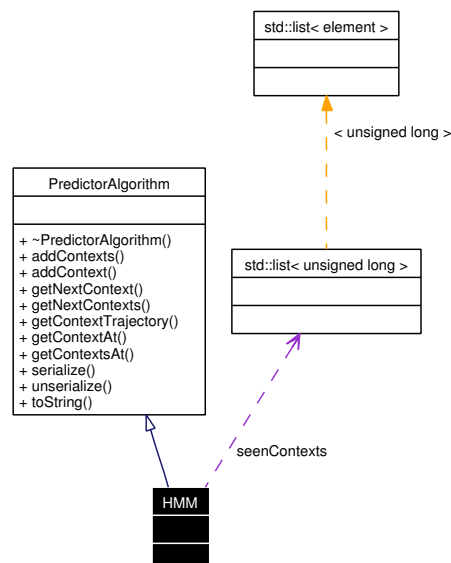
## 6.27 HMM Class Reference

```
#include <HMM.h>
```

Inheritance diagram for HMM:



Collaboration diagram for HMM:



### [NOHEADER]

- **HMM** ([predictorparams](#) &params)
- **HMM** (unsigned int [numHiddenStates](#), unsigned int numAlphabetSymbols)
- virtual `~HMM()`

*Destructor.*

- virtual void [addContexts](#) (const [membershipList](#) \*contexts, time\_t time)
- virtual void [addContext](#) (unsigned long contextId, time\_t time)
- virtual unsigned long [getNextContext](#) () const
- virtual [membershipList](#) [getNextContexts](#) () const
- virtual [contextTrajectory](#) [getContextTrajectory](#) (unsigned int start, unsigned int end) const
- virtual unsigned long [getContextAt](#) (time\_t time) const
- virtual [membershipList](#) [getContextsAt](#) (time\_t time) const
- virtual string [serialize](#) () const  
*Serialize a predictors data to a string.*
- virtual void [unserialize](#) (string data)  
*Unserialize a predictors data from a string.*
- void [addContextTrajectory](#) (const [vector](#)< unsigned int > &traj, unsigned int from, unsigned int to)  
*helper function until I find out how to do a iterative model update....*
- unsigned long [getNextContext](#) (const [vector](#)< unsigned int > &window) const
- virtual [membershipList](#) [getNextContexts](#) (const [vector](#)< unsigned int > &window) const
- virtual string [toString](#) () const  
*This is only for testing.*
- unsigned int [numHiddenStates](#)  
*Private property.*
- unsigned int **numAlphabetSymbols**
- GHMM\_DiscreteModel \* **hmm**
- [list](#)< unsigned long > **seenContexts**

### 6.27.1 Detailed Description

**Todo**

documentation

Definition at line 35 of file HMM.h.

### 6.27.2 Member Function Documentation

#### 6.27.2.1 void HMM::addContext (unsigned long contextId, time\_t time) [virtual]

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 145 of file HMM.cpp.



**6.27.2.2** `void HMM::addContexts (const membershiplist * contexts, time_t time) [virtual]`

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 138 of file HMM.cpp.

**6.27.2.3** `unsigned long HMM::getContextAt (time_t time) const [virtual]`

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 215 of file HMM.cpp.

**6.27.2.4** `membershiplist HMM::getContextsAt (time_t time) const [virtual]`

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 222 of file HMM.cpp.

**6.27.2.5** `contexttrajectory HMM::getContextTrajectory (unsigned int start, unsigned int end) const [virtual]`

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 207 of file HMM.cpp.

**6.27.2.6** `unsigned long HMM::getNextContext () const [virtual]`

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 151 of file HMM.cpp.

References `getNextContexts()`, and `membershiplist`.

Referenced by `main()`.

**6.27.2.7** `membershiplist HMM::getNextContexts () const [virtual]`

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 172 of file HMM.cpp.

References membershipList.

Referenced by getNextContext().

#### **6.27.2.8** `string HMM::serialize () const` `[virtual]`

Serialize a predictors data to a string.

##### **Returns:**

String representation of the predictor data.

Implements [PredictorAlgorithm](#).

Definition at line 229 of file HMM.cpp.

#### **6.27.2.9** `void HMM::unserialize (string data)` `[virtual]`

Unserialize a predictors data from a string.

##### **Parameters:**

*data* String representation of the predictor data.

Implements [PredictorAlgorithm](#).

Definition at line 234 of file HMM.cpp.

### **6.27.3 Member Data Documentation**

#### **6.27.3.1** `unsigned int HMM::numHiddenStates` `[private]`

Private property.

##### **Todo**

documentation

Definition at line 44 of file HMM.h.

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

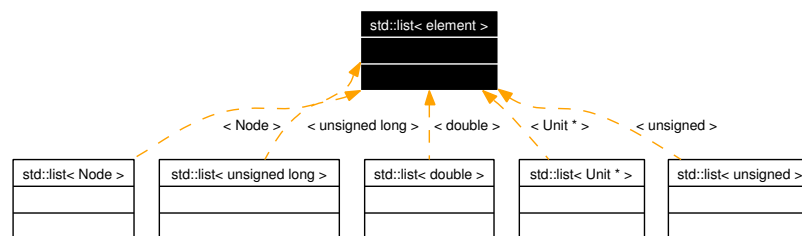
- [HMM.h](#)
- [HMM.cpp](#)

## 6.28 std::list< element > Class Template Reference

STL list template.

```
#include <doxygen.h>
```

Inheritance diagram for std::list< element >:



### 6.28.1 Detailed Description

**template<class element> class std::list< element >**

STL list template.

Definition at line 26 of file doxygen.h.

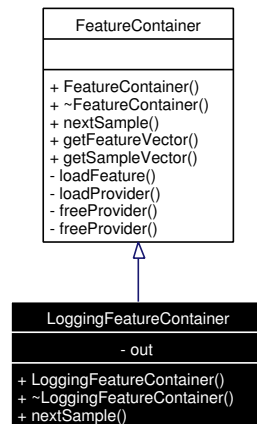
The documentation for this class was generated from the following file:

- [doxygen.h](#)

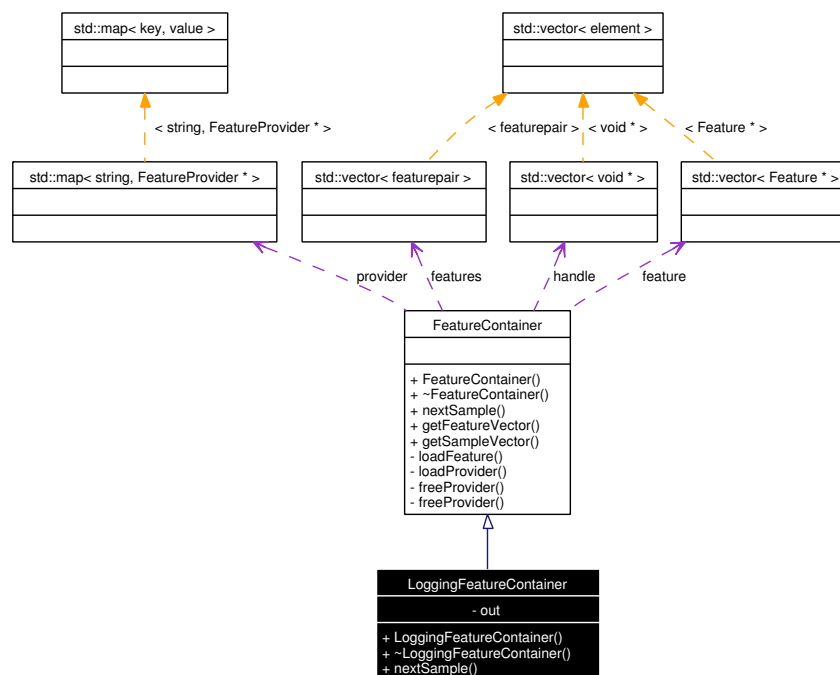
## 6.29 LoggingFeatureContainer Class Reference

```
#include <LoggingFeatureContainer.h>
```

Inheritance diagram for LoggingFeatureContainer:



Collaboration diagram for LoggingFeatureContainer:



### [NOHEADER]

- typedef `pair< string, FeatureProvider * >` `featurepair`

*Protected property.*

- typedef [vector](#)< [featurepair](#) > [featuremap](#)

*Protected property.*

- [featuremap](#) [features](#)

*Protected property.*

## Public Member Functions

- [LoggingFeatureContainer](#) ([ConfigReader](#) \*config)

*Constructor.*

- virtual [~LoggingFeatureContainer](#) ()

*Destructor.*

- virtual void [nextSample](#) ()

*Retrieve next sample.*

- virtual [featurevector](#) [getFeatureVector](#) () const

*Get random feature vector.*

- virtual const [featurevector](#) \* [getSampleVector](#) () const

*Get sample vector.*

## Private Attributes

- FILE \* [out](#)

*Private property.*

### 6.29.1 Detailed Description

#### [Todo](#)

documentation

Definition at line 32 of file [LoggingFeatureContainer.h](#).

### 6.29.2 Member Typedef Documentation

#### 6.29.2.1 [typedef](#) [vector](#)<[featurepair](#)> [FeatureContainer::featuremap](#) [protected, inherited]

Protected property.

#### [Todo](#)

documentation

Definition at line 110 of file [FeatureContainer.h](#).

**6.29.2.2** `typedef pair<string, FeatureProvider*> FeatureContainer::featurepair` [protected, inherited]

Protected property.

#### Todo

documentation

Definition at line 109 of file FeatureContainer.h.

Referenced by FeatureContainer::loadFeature().

## 6.29.3 Constructor & Destructor Documentation

**6.29.3.1** `LoggingFeatureContainer::LoggingFeatureContainer (ConfigReader * config)`

Constructor.

Initiaizes a logfile and logs all operations performed on the underlaying `FeatureContainer` class for later replay.

#### Parameters:

*config* A `ConfigReader` instance

#### See also:

`FeatureContainer`

Definition at line 28 of file LoggingFeatureContainer.cpp.

References `ConfigReader::getGlobals()`, and `out`.

## 6.29.4 Member Function Documentation

**6.29.4.1** `featurevector FeatureContainer::getFeatureVector () const` [virtual, inherited]

Get random feature vector.

A feature vector contains pointers to implementations of the specific features, but initialized with random values. These random positions in the feature space are usually used for initializing classification algorithms. The feature objects returned by this method are allocated automatically for the caller (since the caller can not know the specific implementations, this method acts as a factories), but **must** be freed by the caller. The specific feature objects are created by the FeatureContainer implementations, as this method calls `getFeature` for every dimension.

#### Returns:

A vector of `Feature` implementations. Every element of the returned vector must be freed after use.

#### See also:

`Feature`

`FeatureContainer::getFeature`

Definition at line 179 of file FeatureContainer.cpp.

References `FeatureContainer::features`, and `featurevector`.

Referenced by `ReplayFeatureContainer::nextSample()`, and `Scanner::run()`.

#### 6.29.4.2 `const featurevector * FeatureContainer::getSampleVector () const` [virtual, inherited]

Get sample vector.

A sample vector contains pointers to implementations of the specific features, with values representing the current sensor values. For performance reasons (since this method can be called by multiple callers in the same time step for retrieving the current feature values), this method returns a pointer to a globally allocated object. Callers **must not** free or modify either the returned vector reference or the feature implementations contained therein.

##### Returns:

A reference to a globally allocated vector containing features that represent the current sensor values.

Reimplemented in [ReplayFeatureContainer](#).

Definition at line 193 of file `FeatureContainer.cpp`.

References `FeatureContainer::feature`, and `featurevector`.

Referenced by `evaluate()`, `Java_at_jku_intelligence_context_Context_nativeGetClusterMembership()`, `Java_at_jku_intelligence_context_Context_nativeGetNumClusters()`, `Java_at_jku_intelligence_samples_SampleContainer_nativeGetSampleVector()`, `nextSample()`, and `Scanner::run()`.

#### 6.29.4.3 `void LoggingFeatureContainer::nextSample ()` [virtual]

Retrieve next sample.

Retrieve next samples from all features at once. This method should be called at the beginning of each time step. This method calls `nextSample` for each [FeatureContainer](#) and then allocates the global feature vector which can be returned by successive calls to `getSampleVector`. The specific feature objects are created by the [FeatureContainer](#) implementations, as this method calls `getSample` for every dimension.

##### See also:

[FeatureContainer::nextSample](#)  
[FeatureContainer::getSample](#)  
[getSampleVector](#)

Reimplemented from [FeatureContainer](#).

Definition at line 49 of file `LoggingFeatureContainer.cpp`.

References `featurevector`, `FeatureContainer::getSampleVector()`, `FeatureContainer::nextSample()`, `out`, and `serializeFeatureVector()`.

## 6.29.5 Member Data Documentation

### 6.29.5.1 `featuremap FeatureContainer::features` [protected, inherited]

Protected property.

##### Todo

documentation

Definition at line 112 of file `FeatureContainer.h`.

Referenced by `FeatureContainer::FeatureContainer()`, `FeatureContainer::getFeatureVector()`, `FeatureContainer::loadFeature()`, and `FeatureContainer::nextSample()`.

#### 6.29.5.2 FILE\* `LoggingFeatureContainer::out` [private]

Private property.

#### Todo

documentation

Reimplemented from [FeatureContainer](#).

Definition at line 34 of file `LoggingFeatureContainer.h`.

Referenced by `LoggingFeatureContainer()`, `nextSample()`, and `~LoggingFeatureContainer()`.

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

- [LoggingFeatureContainer.h](#)
- [LoggingFeatureContainer.cpp](#)

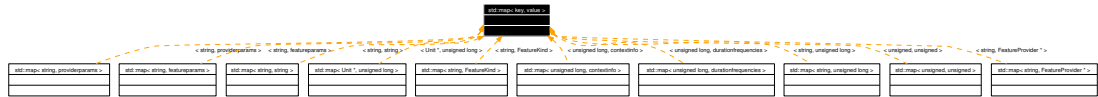


### 6.30 std::map< key, value > Class Template Reference

STL map template.

```
#include <doxygen.h>
```

Inheritance diagram for std::map< key, value >:



#### 6.30.1 Detailed Description

template<class key, class value> class std::map< key, value >

STL map template.

Definition at line 28 of file doxygen.h.

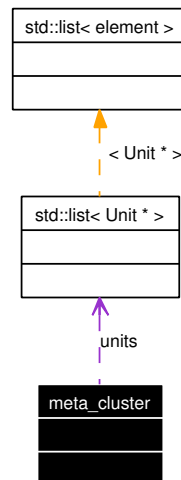
The documentation for this class was generated from the following file:

- [doxygen.h](#)

## 6.31 meta\_cluster Struct Reference

```
#include <GNG.h>
```

Collaboration diagram for meta\_cluster:



### Public Attributes

- unsigned int [cluster\\_id](#)
- [list< Unit \\* >](#) [units](#)
- unsigned int [winner\\_hits](#)

### 6.31.1 Detailed Description

#### Todo

documentation

Definition at line 33 of file GNG.h.

### 6.31.2 Member Data Documentation

#### 6.31.2.1 unsigned int [meta\\_cluster::cluster\\_id](#)

##### Todo

documentation

Definition at line 36 of file GNG.h.

#### 6.31.2.2 [list<Unit\\*>](#) [meta\\_cluster::units](#)

##### Todo

documentation

Definition at line 37 of file GNG.h.

### 6.31.2.3 unsigned int [meta\\_cluster::winner\\_hits](#)

#### **Todo**

documentation

Definition at line 38 of file GNG.h.

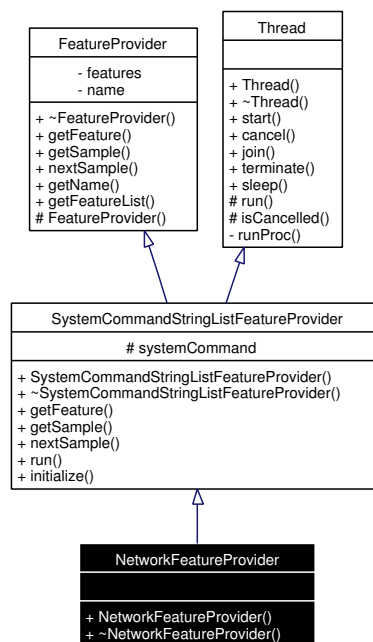
The documentation for this struct was generated from the following file:

- [GNG.h](#)

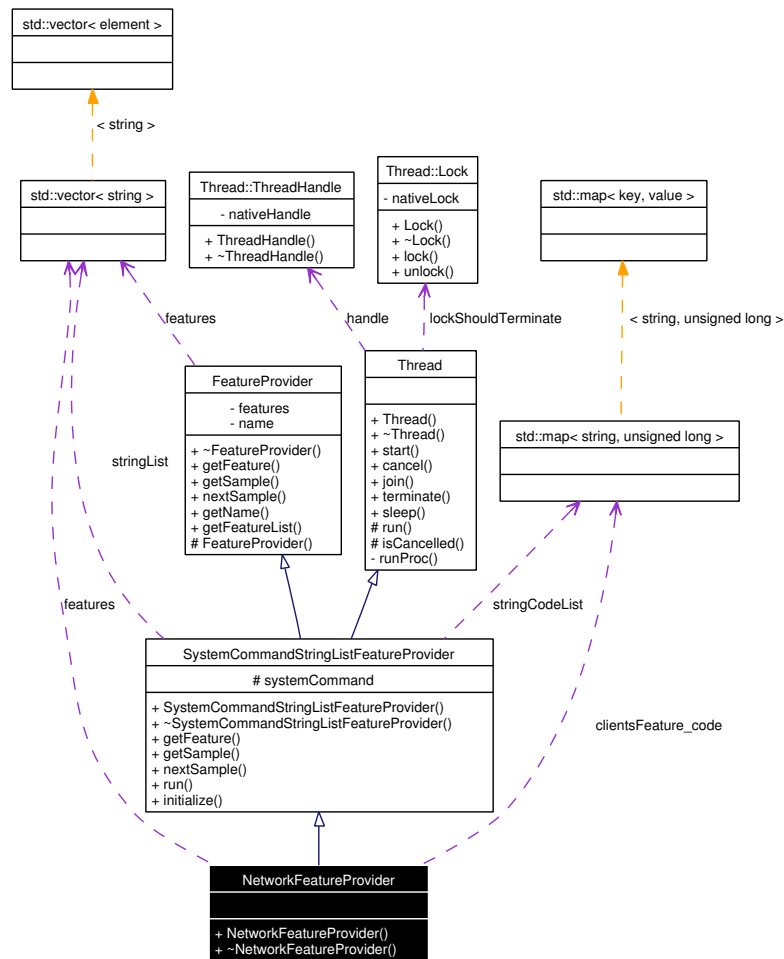
## 6.32 NetworkFeatureProvider Class Reference

```
#include <Network.h>
```

Inheritance diagram for NetworkFeatureProvider:



Collaboration diagram for NetworkFeatureProvider:



## Public Member Functions

- `NetworkFeatureProvider (featureparams &params)`
- `virtual ~NetworkFeatureProvider () throw ()`  
*Destructor.*
- `virtual Feature * getFeature (string name) const`  
*Query feature instance.*
- `virtual Feature * getSample (string name) const`  
*Query sample instance.*
- `virtual void nextSample (clock_t checkpoint)`  
*Prepare sample.*
- `virtual void run () throw ()`  
*Thread main worker function.*
- `string getName () const`

*Query provider name.*

- `stringvector * getFeatureList () const`

*Query list of provided features.*

- `void start ()`

*Start thread execution.*

- `void cancel ()`

*End thread execution.*

- `void join ()`

*Join thread.*

- `void terminate ()`

*Terminate thread execution.*

## Static Public Member Functions

- `void initialize (string name)`
- `void sleep (unsigned milliseconds)`

*sleep function*

## Protected Member Functions

- `bool isCancelled ()`

## Protected Attributes

- `string systemCommand`

*Warning: change with care and only when necessary !*

## Private Attributes

- `stringvector features`
- `stringcode clientsFeature_code`
- `long clientsFeature_maxlen`

### 6.32.1 Detailed Description

#### **Todo**

documentation

Definition at line 32 of file Network.h.

## 6.32.2 Constructor & Destructor Documentation

### 6.32.2.1 NetworkFeatureProvider::NetworkFeatureProvider ([featureparams](#) & *params*)

#### Todo

documentation

#### Todo

check parameter !!

Definition at line 40 of file Network.cpp.

References `DEFAULT_NETWORK`, `SYSTEM_COMMAND_1`, and `SYSTEM_COMMAND_2`.

## 6.32.3 Member Function Documentation

### 6.32.3.1 [Feature](#) \* SystemCommandStringListFeatureProvider::getFeature (string *name*) const [virtual, inherited]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

#### Parameters:

*name* Name of the requested feature.

#### Returns:

An instance of the requested feature.

Implements [FeatureProvider](#).

Definition at line 86 of file SystemCommandStringList.cpp.

References `SystemCommandStringListFeatureProvider::providerName`.

### 6.32.3.2 [stringvector](#)\* FeatureProvider::getFeatureList () const [inline, inherited]

Query list of provided features.

#### Returns:

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by `FeatureContainer::loadFeature()`.

### 6.32.3.3 [string](#) FeatureProvider::getName () const [inline, inherited]

Query provider name.

#### Returns:

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

#### 6.32.3.4 **Feature** \* **SystemCommandStringListFeatureProvider::getSample (string *name*) const** [virtual, inherited]

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

##### Parameters:

***name*** Name of the requested feature.

##### Returns:

An instance of the sample.

##### See also:

[nextSample](#)

Implements [FeatureProvider](#).

Definition at line 95 of file SystemCommandStringList.cpp.

References [SystemCommandStringListFeatureProvider::providerName](#), [SystemCommandStringListFeatureProvider::stringList](#), and [SystemCommandStringListFeatureProvider::stringListValid](#).

#### 6.32.3.5 **void SystemCommandStringListFeatureProvider::initialize (string *name*)** [static, inherited]

##### Todo

documentation

#### 6.32.3.6 **bool Thread::isCancelled ()** [protected, inherited]

##### Returns:

*true* if the thread is cancelled, *false* if the thread is still running

Definition at line 43 of file Thread.cpp.

References [Thread::cancelled](#).

Referenced by [GSMFeatureProvider::nextSample\(\)](#), [WlanLinuxFeatureProvider::run\(\)](#), [SystemCommandStringListFeatureProvider::run\(\)](#), and [BluetoothLinuxFeatureProvider::run\(\)](#).

#### 6.32.3.7 **void SystemCommandStringListFeatureProvider::nextSample (clock\_t *checkpoint*)** [virtual, inherited]

Prepare sample.

Prepare the feature provider to return the next sample from the sensor when [getSample\(\)](#) is called.

##### Parameters:

***checkpoint*** This parameter tells the feature extractor that it must not return sensordata sampled earlier than the given clock value.



See also:

[getSample](#)

Implements [FeatureProvider](#).

Definition at line 106 of file SystemCommandStringList.cpp.

#### 6.32.3.8 void Thread::sleep (unsigned *milliseconds*) [static, inherited]

sleep function

Parameters:

*milliseconds* Timeout in milliseconds

Definition at line 80 of file ThreadPosix.cpp.

Referenced by [Main\(\)](#), [GSMFeatureProvider::readLine\(\)](#), [WlanLinuxFeatureProvider::run\(\)](#), [SystemCommandStringListFeatureProvider::run\(\)](#), [Scanner::run\(\)](#), and [BluetoothLinuxFeatureProvider::run\(\)](#).

### 6.32.4 Member Data Documentation

#### 6.32.4.1 [stringcode NetworkFeatureProvider::clientsFeature\\_code](#) [private]

[Todo](#)

documentation

Definition at line 39 of file Network.h.

#### 6.32.4.2 long [NetworkFeatureProvider::clientsFeature\\_maxlen](#) [private]

[Todo](#)

documentation

Definition at line 40 of file Network.h.

#### 6.32.4.3 [stringvector NetworkFeatureProvider::features](#) [private]

[Todo](#)

documentation

Reimplemented from [FeatureProvider](#).

Definition at line 37 of file Network.h.

#### 6.32.4.4 [stringvector SystemCommandStringListFeatureProvider::stringList](#) [protected, inherited]

[Todo](#)

documentation

Definition at line 102 of file SystemCommandStringList.h.

Referenced by [SystemCommandStringListFeatureProvider::getSample\(\)](#), and [SystemCommandStringListFeatureProvider::run\(\)](#).

**Todo**  
documentation

Referenced by `SystemCommandStringListFeatureProvider::getSample()`, `SystemCommandStringListFeatureProvider::run()`, and `SystemCommandStringListFeatureProvider::SystemCommandStringListFeatureProvider()`.

- Network.h
- Network.cpp

## 6.33 next\_context\_description Struct Reference

```
#include <SingleStepDuration.h>
```

### Public Attributes

- unsigned long [duration](#)
- double [probability](#)

### 6.33.1 Detailed Description

**Todo**

documentation

Definition at line 32 of file SingleStepDuration.h.

### 6.33.2 Member Data Documentation

#### 6.33.2.1 unsigned long [next\\_context\\_description::duration](#)

**Todo**

documentation

Definition at line 35 of file SingleStepDuration.h.

#### 6.33.2.2 double [next\\_context\\_description::probability](#)

**Todo**

documentation

Definition at line 36 of file SingleStepDuration.h.

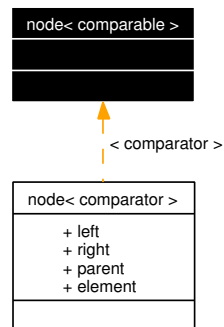
The documentation for this struct was generated from the following file:

- [SingleStepDuration.h](#)

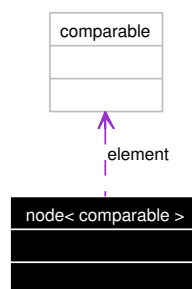
## 6.34 node< comparable > Struct Template Reference

```
#include <splaytree.h>
```

Inheritance diagram for node< comparable >:



Collaboration diagram for node< comparable >:



### Public Attributes

- node< comparable > \* [left](#)
- node< comparable > \* [right](#)
- node< comparable > \* [parent](#)
- comparable [element](#)

#### 6.34.1 Detailed Description

```
template<class comparable> struct node< comparable >
```

**Todo**

documentation

Definition at line 34 of file splaytree.h.

## 6.34.2 Member Data Documentation

### 6.34.2.1 `template<class comparable> comparable node< comparable >::element`

#### Todo

documentation

Definition at line 40 of file `splaytree.h`.

Referenced by `splaytree< comparable >::findMax()`, `splaytree< comparable >::findMin()`, `splaytree< comparable >::insert()`, and `splaytree< comparable >::sort()`.

### 6.34.2.2 `template<class comparable> node<comparable>* node< comparable >::left`

#### Todo

documentation

Definition at line 37 of file `splaytree.h`.

Referenced by `splaytree< comparable >::findMin()`, and `splaytree< comparable >::insert()`.

### 6.34.2.3 `template<class comparable> node<comparable>* node< comparable >::parent`

#### Todo

documentation

Definition at line 39 of file `splaytree.h`.

Referenced by `splaytree< comparable >::insert()`.

### 6.34.2.4 `template<class comparable> node<comparable>* node< comparable >::right`

#### Todo

documentation

Definition at line 38 of file `splaytree.h`.

Referenced by `splaytree< comparable >::findMax()`, `splaytree< comparable >::insert()`, and `splaytree< comparable >::remove()`.

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

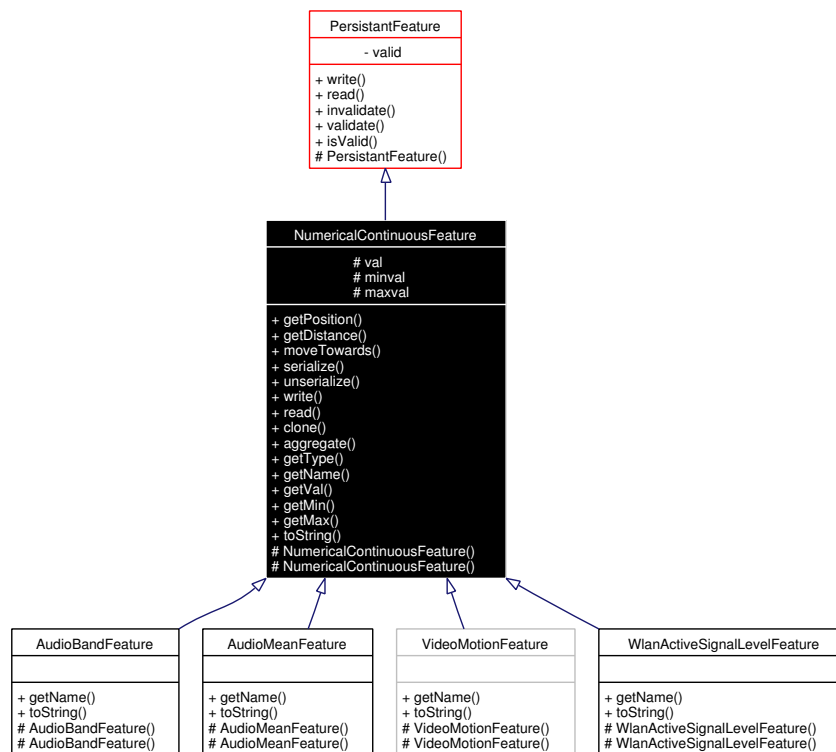
- [splaytree.h](#)

## 6.35 NumericalContinuousFeature Class Reference

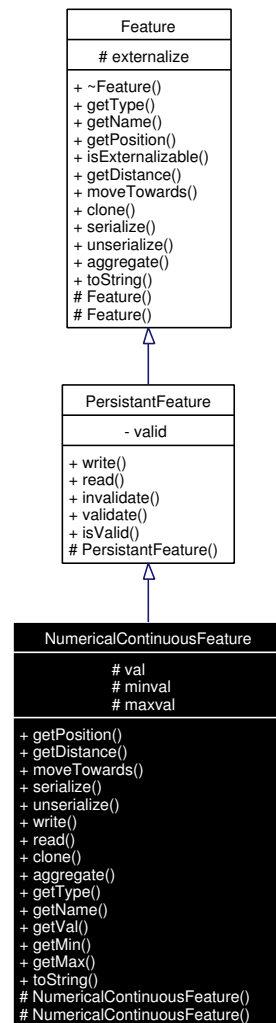
Abstract continuous numerical feature.

```
#include <Numerical.h>
```

Inheritance diagram for NumericalContinuousFeature:



Collaboration diagram for NumericalContinuousFeature:



## Public Types

- enum [FeatureType](#) {  
**boolean, nominal, ordinal, numerical\_discrete,**  
**numerical\_continuous** }  
*Possible feature types.*

## Public Member Functions

- virtual double [getPosition](#) () const  
*Query a features position.*
- virtual double [getDistance](#) ([Feature](#) \*f) const  
*Calculates the distance between two features.*

- virtual void [moveTowards](#) ([Feature](#) \*f, double factor)

*Move feature.*

- virtual string [serialize](#) () const

*Serialize a samples data to a string.*

- virtual void [unserialize](#) (string value)

*Unserialize a samples data from a string.*

- virtual [featureparams](#) [write](#) () const

*Externalize feature.*

- virtual void [read](#) ([featureparams](#) \*param)

*Load feature from persistant data.*

- virtual [Feature](#) \* [clone](#) () const

*Clone a feature.*

- virtual void [aggregate](#) ([aggregatelist](#) samples)

*Aggregate a sample values from other sources.*

- virtual [FeatureType](#) [getType](#) () const

*Query a features type.*

- virtual const string [getName](#) () const

*Query a features name.*

- const double [getVal](#) () const

- const double [getMin](#) () const

- const double [getMax](#) () const

- virtual string [toString](#) () const

*This is only for testing.*

- void [invalidate](#) ()

*Invalidate feature.*

- void [validate](#) ()

*Set the validation flag to true.*

- bool [isValid](#) ()

*Query validation flag.*

- bool [isExternalizable](#) ()

*Query externalization flag.*



## Protected Member Functions

- [NumericalContinuousFeature](#) (double \*[minval](#), double \*[maxval](#))  
*Feature constructor*
- [NumericalContinuousFeature](#) (double \*[minval](#), double \*[maxval](#), double [val](#))  
*Sample constructor.*

## Protected Attributes

- double [val](#)  
*The feature value.*
- double \* [minval](#)  
*A reference to the static, persistent minimum value seen so far.*
- double \* [maxval](#)  
*A reference to the static, persistent maximum value seen so far.*
- const bool [externalize](#)  
*Externalization flag.*

### 6.35.1 Detailed Description

Abstract continuous numerical feature.

A feature of type `numerical_continuous`. Please see [NumericalDiscreteFeature](#) for an explanation of the details, which is also valid for this class.

See also:

[NumericalDiscreteFeature](#)

Definition at line 126 of file `Numerical.h`.

### 6.35.2 Constructor & Destructor Documentation

#### 6.35.2.1 `NumericalContinuousFeature::NumericalContinuousFeature` (double \* *minval*, double \* *maxval*) [protected]

Feature constructor

This constructor initializes the feature with a random value and should thus only be used for creating e.g. prototypes of points in a clustering space. For creating a specific sample of a feature, the second constructor should be used.

**Parameters:**

*minval* A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the minimum value.

***maxval*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum value.

Definition at line 175 of file Numerical.cpp.

References `rand_double`, and `val`.

Referenced by `clone()`.

### 6.35.2.2 **NumericalContinuousFeature::NumericalContinuousFeature** (`double * minval`, `double * maxval`, `double val`) [`protected`]

Sample constructor.

This constructor initializes the feature with a specific value and should thus be used for samples.

#### Parameters:

***minval*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the minimum value.

***maxval*** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum value.

***val*** The sample value.

Definition at line 187 of file Numerical.cpp.

References `PersistentFeature::invalidate()`.

## 6.35.3 Member Function Documentation

### 6.35.3.1 **void NumericalContinuousFeature::aggregate** (`aggregatelist samples`) [`virtual`]

Aggregate a sample values from other sources.

#### Parameters:

***samples*** A list of `<timestamp, sample>` tuples.

Implements [Feature](#).

Definition at line 309 of file Numerical.cpp.

References `aggregatelist`.

### 6.35.3.2 **Feature \* NumericalContinuousFeature::clone** () `const` [`virtual`]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Definition at line 304 of file Numerical.cpp.

References `maxval`, `minval`, `NumericalContinuousFeature()`, and `val`.

**6.35.3.3 double NumericalContinuousFeature::getDistance (Feature \*f) const** [virtual]

Calculates the distance between two features.

**Parameters:**

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Definition at line 220 of file Numerical.cpp.

References [getPosition\(\)](#), [maxval](#), and [minval](#).

**6.35.3.4 virtual const string NumericalContinuousFeature::getName () const** [inline, virtual]

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Implements [Feature](#).

Reimplemented in [AudioBandFeature](#), [AudioMeanFeature](#), and [WlanActiveSignalLevelFeature](#).

Definition at line 184 of file Numerical.h.

**6.35.3.5 double NumericalContinuousFeature::getPosition () const** [virtual]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Definition at line 204 of file Numerical.cpp.

References [maxval](#), [minval](#), and [val](#).

Referenced by [getDistance\(\)](#).

**6.35.3.6 virtual FeatureType NumericalContinuousFeature::getType () const** [inline, virtual]

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 183 of file Numerical.h.

**6.35.3.7 const double NumericalContinuousFeature::getVal () const [inline]****Returns:**

The value of the feature.

Definition at line 187 of file Numerical.h.

Referenced by `getProvider()`, `Java_at_jku_intelligence_samples_NumericalContinuousSample_nativeGetVal()`, `WlanActiveSignalLevelFeature::toString()`, and `toString()`.

**6.35.3.8 void PersistentFeature::invalidate () [inline, inherited]**

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by `AbstractStringFeature::getCodeForName()`, `NumericalContinuousFeature()`, `NumericalDiscreteFeature::NumericalDiscreteFeature()`, `TimeFeature::TimeFeature()`, `unserialize()`, and `NumericalDiscreteFeature::unserialize()`.

**6.35.3.9 bool Feature::isExternalizable () [inline, inherited]**

Query externalization flag.

**Returns:**

*true* if the feature is externalizable ( i.e. it has persistent data that should be stored across restarts), *false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by `FeatureContainer::FeatureContainer()`, and `FeatureContainer::nextSample()`.

**6.35.3.10 bool PersistentFeature::isValid () [inline, inherited]**

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by `FeatureContainer::nextSample()`.

**6.35.3.11 void NumericalContinuousFeature::moveTowards (Feature \* *f*, double *factor*)**  
[virtual]

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

**Parameters:**

*f* [Feature](#) used for distance measurement.

*factor* Distance weight.

Implements [Feature](#).

Definition at line 227 of file Numerical.cpp.

References maxval, minval, and val.

**6.35.3.12 void NumericalContinuousFeature::read (featureparams \* *param*)** [virtual]

Load feature from persistent data.

Initializes the persistent feature data from the given representation.

**Parameters:**

*param* Persistent feature data.

**See also:**

[write](#)

Implements [PersistentFeature](#).

Definition at line 291 of file Numerical.cpp.

References featureparams, maxval, and minval.

**6.35.3.13 string NumericalContinuousFeature::serialize () const** [virtual]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [Feature](#).

Definition at line 246 of file Numerical.cpp.

References val.

**6.35.3.14 void NumericalContinuousFeature::unserialize (string *value*)** [virtual]

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Definition at line 254 of file Numerical.cpp.

References `PersistentFeature::invalidate()`, `maxval`, `minval`, and `val`.

**6.35.3.15 [featureparams](#) NumericalContinuousFeature::write () const [virtual]**

Externalize feature.

**Returns:**

Persistent feature data.

**See also:**

[read](#)

Implements [PersistentFeature](#).

Definition at line 277 of file Numerical.cpp.

References `featureparams`, `maxval`, and `minval`.

## 6.35.4 Member Data Documentation

**6.35.4.1 `const bool` [Feature::externalize](#) [protected, inherited]**

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to `PersistentFeature`, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**

[PersistentFeature](#)

Definition at line 187 of file `Feature.h`.

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

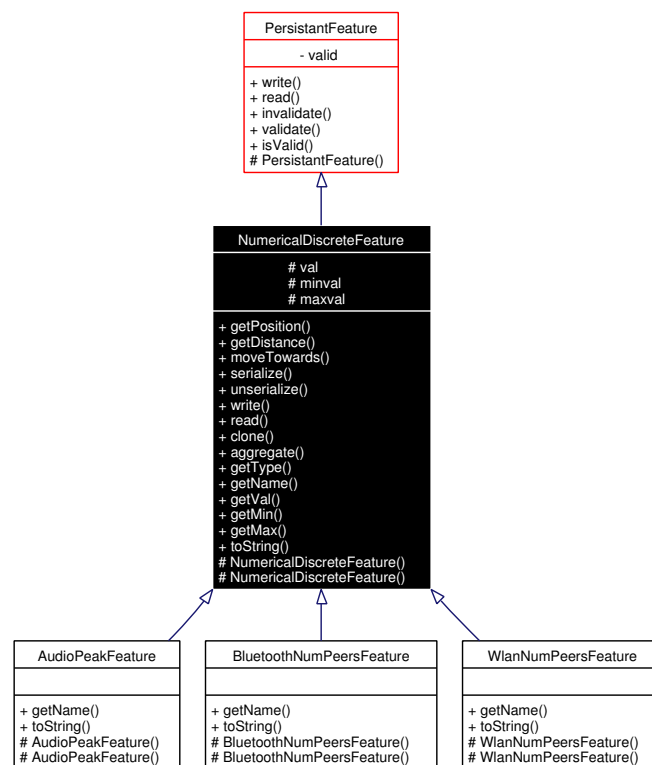
- [Numerical.h](#)
- [Numerical.cpp](#)

## 6.36 NumericalDiscreteFeature Class Reference

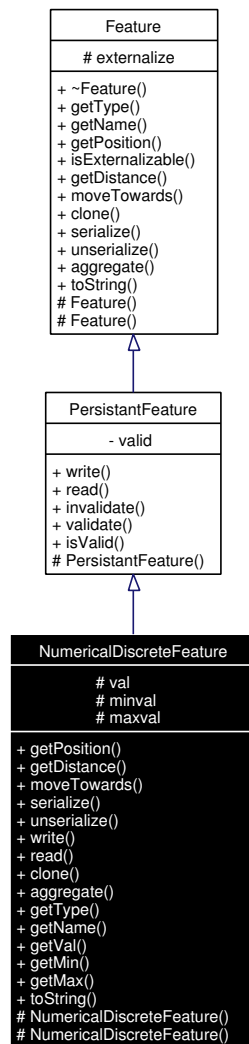
Abstract discrete numerical feature.

```
#include <Numerical.h>
```

Inheritance diagram for NumericalDiscreteFeature:



Collaboration diagram for NumericalDiscreteFeature:



## Public Types

- enum [FeatureType](#) {  
**boolean, nominal, ordinal, numerical\_discrete,**  
**numerical\_continuous** }  
*Possible feature types.*

## Public Member Functions

- virtual double [getPosition](#) () const  
*Query a features position.*
- virtual double [getDistance](#) ([Feature](#) \*f) const  
*Calculates the distance between two features.*



- virtual void [moveTowards](#) ([Feature](#) \*f, double factor)

*Move feature.*

- virtual string [serialize](#) () const

*Serialize a samples data to a string.*

- virtual void [unserialize](#) (string value)

*Unserialize a samples data from a string.*

- virtual [featureparams](#) [write](#) () const

*Externalize feature.*

- virtual void [read](#) ([featureparams](#) \*param)

*Load feature from persistant data.*

- virtual [Feature](#) \* [clone](#) () const

*Clone a feature.*

- virtual void [aggregate](#) ([aggregatelist](#) samples)

*Aggregate a sample values from other sources.*

- virtual [FeatureType](#) [getType](#) () const

*Query a features type.*

- virtual const string [getName](#) () const

*Query a features name.*

- const long [getVal](#) () const

- const long [getMin](#) () const

- const long [getMax](#) () const

- virtual string [toString](#) () const

*This is only for testing.*

- void [invalidate](#) ()

*Invalidate feature.*

- void [validate](#) ()

*Set the validation flag to true.*

- bool [isValid](#) ()

*Query validation flag.*

- bool [isExternalizable](#) ()

*Query externalization flag.*

## Protected Member Functions

- [NumericalDiscreteFeature](#) (long \*[minval](#), long \*[maxval](#))  
*Feature constructor*
- [NumericalDiscreteFeature](#) (long \*[minval](#), long \*[maxval](#), long [val](#))  
*Sample constructor.*

## Protected Attributes

- double [val](#)  
*The feature value.*
- long \* [minval](#)  
*A reference to the static, persistant minimum value seen so far.*
- long \* [maxval](#)  
*A reference to the static, persistant maximum value seen so far.*
- const bool [externalize](#)  
*Externalization flag.*

### 6.36.1 Detailed Description

Abstract discrete numerical feature.

A feature of type `numerical_discrete`. This class implements the `getType` method of the [Feature](#) interface and, as a base class for special features, implements the complete handling of numerical discrete values. This includes the implementations of `getDistance` and `moveTowards` as well as `getPosition`, `serialize` and `unserialize`. Additionally, minimum and maximum seen values are stored as persistent values of the feature and used for computing the normalized distance and position. Although these values are handled inside this class, the storage for them has to be provided by the subclasses (or generally the factory that creates objects of this class).

Definition at line 46 of file `Numerical.h`.

### 6.36.2 Constructor & Destructor Documentation

#### 6.36.2.1 [NumericalDiscreteFeature::NumericalDiscreteFeature](#) (long \* [minval](#), long \* [maxval](#)) [protected]

Feature constructor

This constructor initializes the feature with a random value and should thus only be used for creating e.g. prototypes of points in a clustering space. For creating a specific sample of a feature, the second constructor should be used.

#### Parameters:

**[minval](#)** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistently storing the minimum value.

**maxval** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum value.

Definition at line 28 of file Numerical.cpp.

References `rand_double`, and `val`.

Referenced by `clone()`.

### 6.36.2.2 NumericalDiscreteFeature::NumericalDiscreteFeature (long \* minval, long \* maxval, long val) [protected]

Sample constructor.

This constructor initializes the feature with a specific value and should thus be used for samples.

#### Parameters:

**minval** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the minimum value.

**maxval** A reference to a static variable (it must be same for all created objects of `_one_` feature) for persistantly storing the maximum value.

**val** The sample value.

Definition at line 40 of file Numerical.cpp.

References `PersistentFeature::invalidate()`.

## 6.36.3 Member Function Documentation

### 6.36.3.1 void NumericalDiscreteFeature::aggregate (aggregatelist samples) [virtual]

Aggregate a sample values from other sources.

#### Parameters:

**samples** A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 159 of file Numerical.cpp.

References `aggregatelist`.

### 6.36.3.2 Feature \* NumericalDiscreteFeature::clone () const [virtual]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Definition at line 154 of file Numerical.cpp.

References `maxval`, `minval`, `NumericalDiscreteFeature()`, and `val`.

**6.36.3.3 double NumericalDiscreteFeature::getDistance (Feature \*f) const [virtual]**

Calculates the distance between two features.

**Parameters:**

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Definition at line 73 of file Numerical.cpp.

References getPosition(), maxval, and minval.

**6.36.3.4 virtual const string NumericalDiscreteFeature::getName () const [inline, virtual]**

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Implements [Feature](#).

Reimplemented in [AudioPeakFeature](#), [BluetoothNumPeersFeature](#), and [WlanNumPeersFeature](#).

Definition at line 104 of file Numerical.h.

**6.36.3.5 double NumericalDiscreteFeature::getPosition () const [virtual]**

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Definition at line 57 of file Numerical.cpp.

References maxval, minval, and val.

Referenced by getDistance().

**6.36.3.6 virtual FeatureType NumericalDiscreteFeature::getType () const [inline, virtual]**

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 103 of file Numerical.h.

**6.36.3.7 const long NumericalDiscreteFeature::getVal () const [inline]****Returns:**

The value of the feature.

Definition at line 107 of file Numerical.h.

References val.

Referenced by Java\_at\_jku\_intelligence\_samples\_NumericalDiscreteSample\_nativeGetVal(), WlanNumPeersFeature::toString(), toString(), and BluetoothNumPeersFeature::toString().

**6.36.3.8 void PersistantFeature::invalidate () [inline, inherited]**

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistant feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by AbstractStringFeature::getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature(), TimeFeature::TimeFeature(), NumericalContinuousFeature::unserialize(), and unserialize().

**6.36.3.9 bool Feature::isExternalizable () [inline, inherited]**

Query externalization flag.

**Returns:**

*true* if the feature is externalizeable ( i.e. it has persistend data that should be stored across restarts), *false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.36.3.10 bool PersistantFeature::isValid () [inline, inherited]**

Query validation flag.

**Returns:**

*true* if the features persistant data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

**6.36.3.11 void NumericalDiscreteFeature::moveTowards (Feature \**f*, double *factor*) [virtual]**

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

**Parameters:**

*f* Feature used for distance measurement.

*factor* Distance weight.

Implements Feature.

Definition at line 80 of file Numerical.cpp.

References maxval, minval, and val.

**6.36.3.12 void NumericalDiscreteFeature::read (featureparams \**param*) [virtual]**

Load feature from persistant data.

Initializes the persistant feature data from the given representation.

**Parameters:**

*param* Persistant feature data.

**See also:**

[write](#)

Implements PersistantFeature.

Definition at line 141 of file Numerical.cpp.

References featureparams, maxval, and minval.

**6.36.3.13 string NumericalDiscreteFeature::serialize () const [virtual]**

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements Feature.

Definition at line 96 of file Numerical.cpp.

References val.

**6.36.3.14 void NumericalDiscreteFeature::unserialize (string *value*) [virtual]**

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Definition at line 104 of file Numerical.cpp.

References `PersistentFeature::invalidate()`, `maxval`, `minval`, and `val`.

#### 6.36.3.15 [featureparams](#) `NumericalDiscreteFeature::write () const` `[virtual]`

Externalize feature.

**Returns:**

Persistent feature data.

**See also:**

[read](#)

Implements [PersistentFeature](#).

Definition at line 127 of file Numerical.cpp.

References `featureparams`, `maxval`, and `minval`.

### 6.36.4 Member Data Documentation

#### 6.36.4.1 `const bool` [Feature::externalize](#) `[protected, inherited]`

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to `PersistentFeature`, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**

[PersistentFeature](#)

Definition at line 187 of file Feature.h.

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

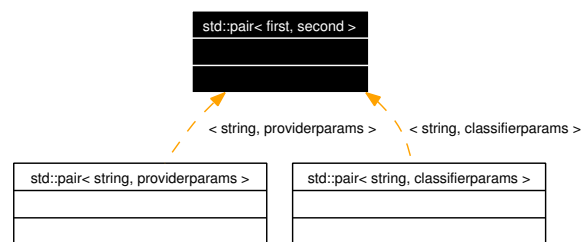
- [Numerical.h](#)
- [Numerical.cpp](#)

## 6.37 `std::pair< first, second >` Class Template Reference

STL pair template.

```
#include <doxygen.h>
```

Inheritance diagram for `std::pair< first, second >`:



### 6.37.1 Detailed Description

**template<class first, class second> class `std::pair< first, second >`**

STL pair template.

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

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

- [doxygen.h](#)

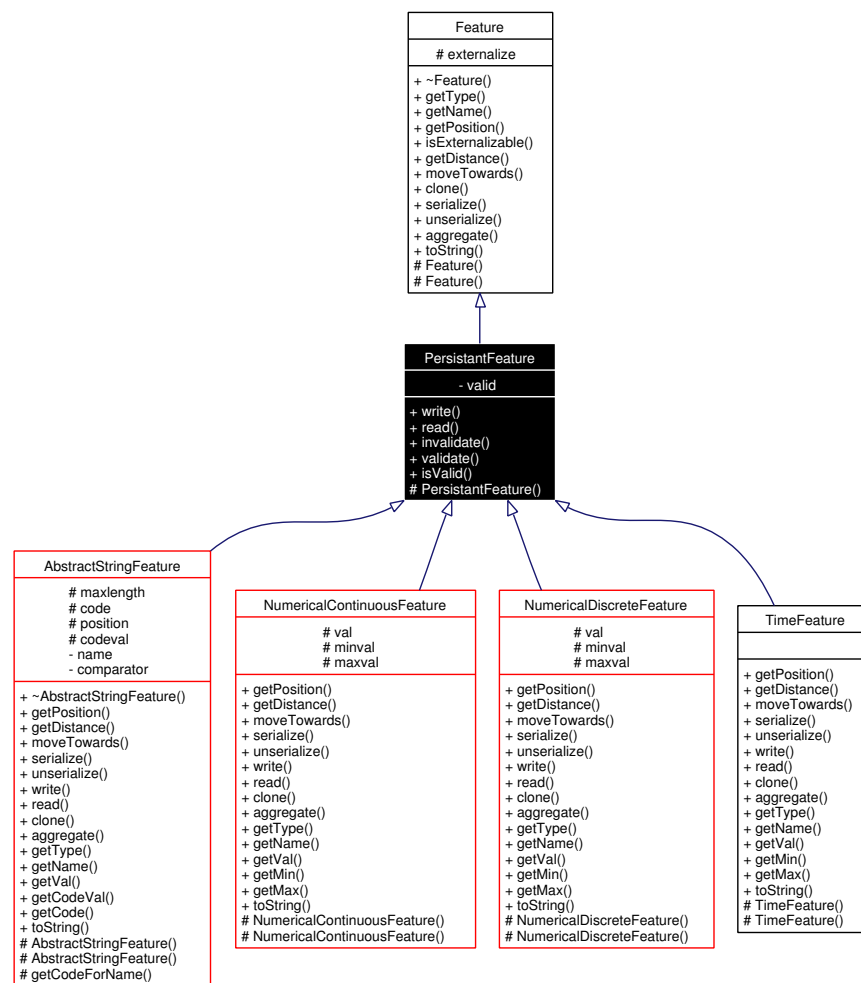


## 6.38 PersistentFeature Class Reference

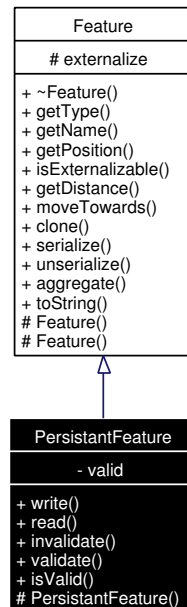
PersistentFeature interface.

```
#include <Feature.h>
```

Inheritance diagram for PersistentFeature:



Collaboration diagram for PersistentFeature:



## Public Types

- enum `FeatureType` {  
**boolean, nominal, ordinal, numerical\_discrete,**  
**numerical\_continuous** }  
*Possible feature types.*

## Public Member Functions

- virtual `featureparams write ()` const =0  
*Externalize feature.*
- virtual void `read (featureparams *param)`=0  
*Load feature from persistant data.*
- void `invalidate ()`  
*Invalidate feature.*
- void `validate ()`  
*Set the validation flag to true.*
- bool `isValid ()`  
*Query validation flag.*
- virtual `FeatureType getType ()` const =0  
*Query a features type.*

- virtual const string `getName ()` const =0  
*Query a features name.*
- virtual double `getPosition ()` const =0  
*Query a features position.*
- bool `isExternalizable ()`  
*Query externalization flag.*
- virtual double `getDistance (Feature *f)` const =0  
*Calculates the distance between two features.*
- virtual void `moveTowards (Feature *f, double factor)`=0  
*Move feature.*
- virtual `Feature * clone ()` const =0  
*Clone a feature.*
- virtual string `serialize ()` const =0  
*Serialize a samples data to a string.*
- virtual void `unserialize (string value)`=0  
*Unserialize a samples data from a string.*
- virtual void `aggregate (aggregatelist samples)`=0  
*Aggregate a sample values from other sources.*
- virtual string `toString ()` const =0  
*This is only for testing.*

## Protected Member Functions

- `PersistentFeature ()`  
*Default constructor.*

## Protected Attributes

- const bool `externalize`  
*Externalization flag.*

## Private Attributes

- bool `valid`  
*Validation flag.*

### 6.38.1 Detailed Description

PersistentFeature interface.

This interface only enhances the [Feature](#) interface by methods to handle persistent data of a feature. It sets the externalization flag to *true*.

See also:

[externalize](#)

Definition at line 321 of file Feature.h.

### 6.38.2 Constructor & Destructor Documentation

#### 6.38.2.1 PersistentFeature::PersistentFeature () [inline, protected]

Default constructor.

Sets the externalization flag and the validation flag to *true*.

See also:

[externalize](#)

[valid](#)

Definition at line 342 of file Feature.h.

### 6.38.3 Member Function Documentation

#### 6.38.3.1 virtual void Feature::aggregate ([aggregatelist samples](#)) [pure virtual, inherited]

Aggregate a sample values from other sources.

Parameters:

*samples* A list of <timestamp, sample> tuples.

Implemented in [AbstractStringFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

#### 6.38.3.2 virtual [Feature\\*](#) Feature::clone () const [pure virtual, inherited]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

**6.38.3.3 virtual double Feature::getDistance (Feature \*f) const** [pure virtual, inherited]

Calculates the distance between two features.

**Parameters:**

*f* Feature used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

**6.38.3.4 virtual const string Feature::getName () const** [pure virtual, inherited]

Query a features name.

**Returns:**

Name of the Feature in the format "Featureprovider.Feature"

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [ActiveWindowFeature](#), [AudioBandFeature](#), [AudioMeanFeature](#), [AudioPeakFeature](#), [BluetoothPeersFeature](#), [BluetoothNumPeersFeature](#), [GSMCellFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [SystemCommandStringListFeature](#), [TimeFeature](#), [WlanActiveEssidFeature](#), [WlanActiveMacAddressFeature](#), [WlanActiveModeFeature](#), [WlanActiveSignalLevelFeature](#), [WlanPeersFeature](#), and [WlanNumPeersFeature](#).

Referenced by `Java_at_jku_intelligence_samples_Sample_nativeGetName()`.

**6.38.3.5 virtual double Feature::getPosition () const** [pure virtual, inherited]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

Referenced by `Java_at_jku_intelligence_samples_Sample_nativeGetPosition()`.

**6.38.3.6 virtual FeatureType Feature::getType () const** [pure virtual, inherited]

Query a features type.

**Returns:**

The type of the Feature.

Implemented in [AbstractStringFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

Referenced by `Java_at_jku_intelligence_samples_Sample_nativeGetType()`.

**6.38.3.7 void PersistentFeature::invalidate () [inline]**

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file `Feature.h`.

Referenced by `AbstractStringFeature::getCodeForName()`, `NumericalContinuousFeature::NumericalContinuousFeature()`, `NumericalDiscreteFeature::NumericalDiscreteFeature()`, `TimeFeature::TimeFeature()`, `NumericalContinuousFeature::unserialize()`, and `NumericalDiscreteFeature::unserialize()`.

**6.38.3.8 bool Feature::isExternalizable () [inline, inherited]**

Query externalization flag.

**Returns:**

*true* if the feature is externalizable ( i.e. it has persistent data that should be stored across restarts), *false* otherwise.

Definition at line 250 of file `Feature.h`.

Referenced by `FeatureContainer::FeatureContainer()`, and `FeatureContainer::nextSample()`.

**6.38.3.9 bool PersistentFeature::isValid () [inline]**

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file `Feature.h`.

Referenced by `FeatureContainer::nextSample()`.

**6.38.3.10 virtual void Feature::moveTowards (Feature \*f, double factor) [pure virtual, inherited]**

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

**Parameters:**

*f* Feature used for distance measurement.

*factor* Distance weight.

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

Referenced by `Unit::splitUnit()`.

#### 6.38.3.11 virtual void PersistantFeature::read ([featureparams](#) \* *param*) [pure virtual]

Load feature from persistant data.

Initializes the persistant feature data from the given representation.

##### Parameters:

*param* Persistant feature data.

##### See also:

[write](#)

Implemented in [AbstractStringFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), and [TimeFeature](#).

#### 6.38.3.12 virtual string Feature::serialize () const [pure virtual, inherited]

Serialize a samples data to a string.

##### Returns:

String representation of the samples data.

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

#### 6.38.3.13 virtual void Feature::unserialize (string *value*) [pure virtual, inherited]

Unserialize a samples data from a string.

##### Parameters:

*value* String representation of the samples data.

Implemented in [AbstractStringFeature](#), [AbstractStringListFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), [PowerFeature](#), [TimeFeature](#), and [WlanActiveModeFeature](#).

Referenced by `unserializeFeatureVector()`.

#### 6.38.3.14 virtual [featureparams](#) PersistantFeature::write () const [pure virtual]

Externalize feature.

##### Returns:

Persistant feature data.

**See also:**[read](#)

Implemented in [AbstractStringFeature](#), [NumericalDiscreteFeature](#), [NumericalContinuousFeature](#), and [TimeFeature](#).

Referenced by `FeatureContainer::nextSample()`.

## 6.38.4 Member Data Documentation

### 6.38.4.1 `const bool Feature::externalize` [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to `PersistentFeature`, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**[PersistentFeature](#)

Definition at line 187 of file `Feature.h`.

### 6.38.4.2 `bool PersistentFeature::valid` [private]

Validation flag.

*true* when the persistent feature data is valid (i.e. it has not been modified since the last call to write), *false* if it has been modified and needs to be written. Definition at line 330 of file `Feature.h`.

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

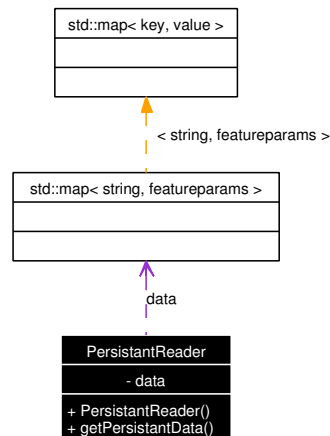
- [Feature.h](#)



## 6.39 PersistentReader Class Reference

```
#include <PersistentReader.h>
```

Collaboration diagram for PersistentReader:



### Public Member Functions

- [PersistentReader](#) (const char \*file)  
*Constructor.*
- [featureparams](#) \* [getPersistentData](#) (string feature)

### Private Attributes

- [map](#)< string, [featureparams](#) > [data](#)

#### 6.39.1 Detailed Description

##### Todo

documentation

Definition at line 33 of file PersistentReader.h.

### 6.39.2 Constructor & Destructor Documentation

#### 6.39.2.1 PersistentReader::PersistentReader (const char \*file)

Constructor.

Load persistent file, validate xml and copy the information into the data collection.

##### Parameters:

*file* Path to the persistent file

Definition at line 27 of file PersistentReader.cpp.

References data.

### 6.39.3 Member Function Documentation

#### 6.39.3.1 `map< string, string > * PersistentReader::getPersistentData (string feature)`

**Todo**

documentation

Definition at line 61 of file PersistentReader.cpp.

References data.

Referenced by FeatureContainer::FeatureContainer().

### 6.39.4 Member Data Documentation

#### 6.39.4.1 `map<string, featureparams> PersistentReader::data` [private]

**Todo**

name type

Definition at line 39 of file PersistentReader.h.

Referenced by getPersistentData(), PersistentReader(), and PersistentWriter::write().

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

- [PersistentReader.h](#)
- [PersistentReader.cpp](#)

## 6.40 PersistentWriter Class Reference

```
#include <PersistentWriter.h>
```

### Static Public Member Functions

- void [write](#) (const char \*file, [map](#)< string, [featureparams](#) > persistdata)

#### 6.40.1 Detailed Description

**Todo**

documentation

Definition at line 32 of file PersistentWriter.h.

#### 6.40.2 Member Function Documentation

**6.40.2.1** void PersistentWriter::write (const char \**file*, [map](#)< string, [featureparams](#) > *persistdata*)  
[static]

**Todo**

documentation

Definition at line 28 of file PersistentWriter.cpp.

References PersistentReader::data.

Referenced by FeatureContainer::nextSample().

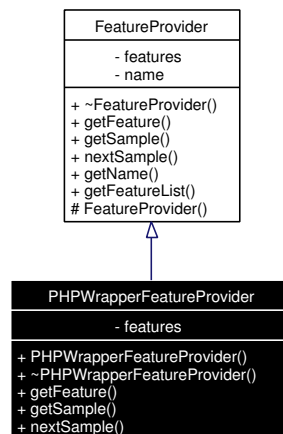
The documentation for this class was generated from the following files:

- [PersistentWriter.h](#)
- [PersistentWriter.cpp](#)

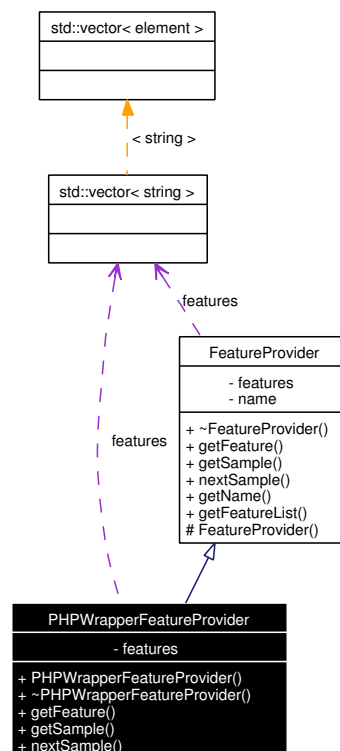
## 6.41 PHPWrapperFeatureProvider Class Reference

```
#include <PHPWrapper.h>
```

Inheritance diagram for PHPWrapperFeatureProvider:



Collaboration diagram for PHPWrapperFeatureProvider:



## Public Member Functions

- [PHPWrapperFeatureProvider](#) ([providerparams](#) &params)
- virtual [~PHPWrapperFeatureProvider](#) ()  
*Destructor.*
- virtual [Feature](#) \* [getFeature](#) (string [name](#)) const  
*Query feature instance.*
- virtual [Feature](#) \* [getSample](#) (string [name](#)) const  
*Query sample instance.*
- virtual void [nextSample](#) (clock\_t checkpoint)  
*Prepare sample.*
- string [getName](#) () const  
*Query provider name.*
- [stringvector](#) \* [getFeatureList](#) () const  
*Query list of provided features.*

## Static Private Attributes

- [stringvector](#) [features](#)  
*List of features.*

### 6.41.1 Detailed Description

#### Todo

documentation

Definition at line 32 of file PHPWrapper.h.

### 6.41.2 Constructor & Destructor Documentation

#### 6.41.2.1 PHPWrapperFeatureProvider::PHPWrapperFeatureProvider ([providerparams](#) & [params](#))

#### Todo

documentation

Definition at line 44 of file PHPWrapper.cpp.

References [PHPFeatureType](#).

### 6.41.3 Member Function Documentation

#### 6.41.3.1 [Feature](#) \* PHPWrapperFeatureProvider::getFeature (string *name*) const [virtual]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the requested feature.

Implements [FeatureProvider](#).

Definition at line 67 of file PHPWrapper.cpp.

#### 6.41.3.2 [stringvector](#)\* FeatureProvider::getFeatureList () const [inline, inherited]

Query list of provided features.

**Returns:**

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by FeatureContainer::loadFeature().

#### 6.41.3.3 [string](#) FeatureProvider::getName () const [inline, inherited]

Query provider name.

**Returns:**

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

#### 6.41.3.4 [Feature](#) \* PHPWrapperFeatureProvider::getSample (string *name*) const [virtual]

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the sample.

**See also:**

[nextSample](#)

Implements [FeatureProvider](#).

Definition at line 72 of file PHPWrapper.cpp.

#### 6.41.3.5 void PHPWrapperFeatureProvider::nextSample (clock\_t *checkpoint*) [virtual]

Prepare sample.

Prepare the feature provider to return the next sample from the sensor when [getSample\(\)](#) is called.

##### Parameters:

*checkpoint* This parameter tells the feature extractor that it must not return sensordata sampled earlier than the given clock value.

##### See also:

[getSample](#)

Implements [FeatureProvider](#).

Definition at line 63 of file PHPWrapper.cpp.

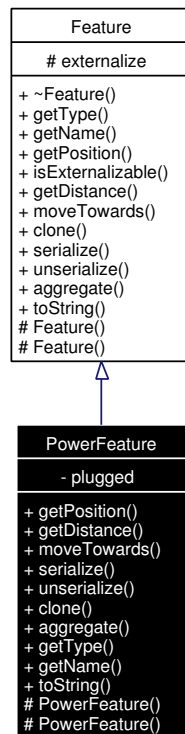
The documentation for this class was generated from the following files:

- [PHPWrapper.h](#)
- [PHPWrapper.cpp](#)

## 6.42 PowerFeature Class Reference

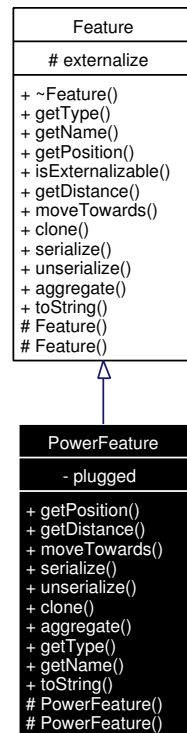
```
#include <Power.h>
```

Inheritance diagram for PowerFeature:



Collaboration diagram for PowerFeature:





## Public Types

- enum `FeatureType` {  
**boolean, nominal, ordinal, numerical\_discrete,**  
**numerical\_continuous** }

*Possible feature types.*

## Public Member Functions

- virtual double `getPosition()` const  
*Query a features position.*
- virtual double `getDistance (Feature *f)` const  
*Calculates the distance between two features.*
- virtual void `moveTowards (Feature *f, double factor)`  
*Move feature.*
- virtual string `serialize()` const  
*Serialize a samples data to a string.*
- virtual void `unserialize (string value)`  
*Unserialize a samples data from a string.*

- virtual [Feature](#) \* [clone](#) () const  
*Clone a feature.*
- virtual void [aggregate](#) ([aggregatelist](#) samples)  
*Aggregate a sample values from other sources.*
- virtual [FeatureType](#) [getType](#) () const  
*Query a features type.*
- virtual const string [getName](#) () const  
*Query a features name.*
- virtual string [toString](#) () const  
*This is only for testing.*
- bool [isExternalizable](#) ()  
*Query externalization flag.*

## Protected Member Functions

- [PowerFeature](#) ()  
*Feature constructor*
- [PowerFeature](#) (bool [plugged](#))  
*Sample constructor.*

## Protected Attributes

- const bool [externalize](#)  
*Externalization flag.*

## Private Attributes

- bool [plugged](#)  
*Indicates wheter the device is plugged or running on battery.*

### 6.42.1 Detailed Description

#### [Todo](#)

documentation

Definition at line 32 of file Power.h.

## 6.42.2 Constructor & Destructor Documentation

### 6.42.2.1 PowerFeature::PowerFeature () [protected]

Feature constructor

This constructor initializes the feature with a random value and should thus only be used for creating e.g. prototypes of points in a clustering space. For creating a specific sample of a feature, the second constructor should be used. Definition at line 34 of file Power.cpp.

### 6.42.2.2 PowerFeature::PowerFeature (bool *plugged*) [protected]

Sample constructor.

This constructor initializes the feature with a specific value and should thus be used for samples. Definition at line 39 of file Power.cpp.

References *plugged*.

## 6.42.3 Member Function Documentation

### 6.42.3.1 void PowerFeature::aggregate ([aggregatelist](#) *samples*) [virtual]

Aggregate a sample values from other sources.

#### Parameters:

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 98 of file Power.cpp.

References *providerparams*.

### 6.42.3.2 [Feature](#) \* PowerFeature::clone () const [virtual]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Definition at line 93 of file Power.cpp.

References *aggregatelist*.

### 6.42.3.3 double PowerFeature::getDistance ([Feature](#) \**f*) const [virtual]

Calculates the distance between two features.

#### Parameters:

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Definition at line 49 of file Power.cpp.

**6.42.3.4 virtual const string PowerFeature::getName () const [inline, virtual]**

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Implements [Feature](#).

Definition at line 71 of file Power.h.

**6.42.3.5 double PowerFeature::getPosition () const [virtual]**

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Definition at line 44 of file Power.cpp.

References plugged.

**6.42.3.6 virtual FeatureType PowerFeature::getType () const [inline, virtual]**

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 70 of file Power.h.

**6.42.3.7 bool Feature::isExternalizable () [inline, inherited]**

Query externalization flag.

**Returns:**

*true* if the feature is externalizeable ( i.e. it has persistend data that should be stored across restarts),  
*false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.42.3.8 void PowerFeature::moveTowards (Feature \**f*, double *factor*) [virtual]**

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

**Parameters:**

*f* Feature used for distance measurement.

*factor* Distance weight.

Implements Feature.

Definition at line 56 of file Power.cpp.

**6.42.3.9 string PowerFeature::serialize () const [virtual]**

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements Feature.

Definition at line 64 of file Power.cpp.

**6.42.3.10 void PowerFeature::unserialize (string *value*) [virtual]**

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements Feature.

Definition at line 72 of file Power.cpp.

References plugged.

**6.42.4 Member Data Documentation****6.42.4.1 const bool Feature::externalize [protected, inherited]**

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to PersistentFeature, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**

[PersistentFeature](#)

Definition at line 187 of file Feature.h.

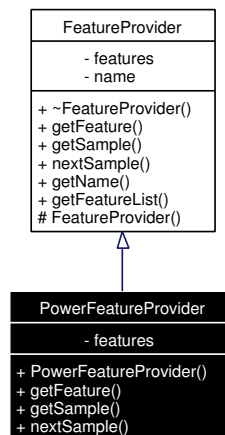
The documentation for this class was generated from the following files:

- [Power.h](#)
- [Power.cpp](#)

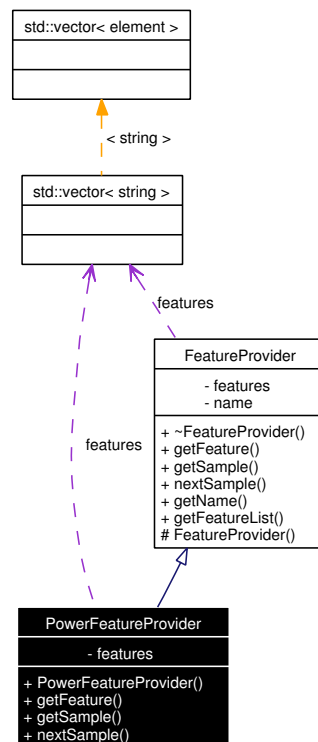
## 6.43 PowerFeatureProvider Class Reference

```
#include <Power.h>
```

Inheritance diagram for PowerFeatureProvider:



Collaboration diagram for PowerFeatureProvider:



## Public Member Functions

- [PowerFeatureProvider](#) ([providerparams](#) &params)  
*Constructor.*
- virtual [Feature](#) \* [getFeature](#) (string [name](#)) const  
*Query feature instance.*
- virtual [Feature](#) \* [getSample](#) (string [name](#)) const  
*Query sample instance.*
- virtual void [nextSample](#) (clock\_t checkpoint)  
*Prepare sample.*
- string [getName](#) () const  
*Query provider name.*
- [stringvector](#) \* [getFeatureList](#) () const  
*Query list of provided features.*

## Private Attributes

- [stringvector](#) [features](#)  
*List of features.*
- bool [plugged](#)
- bool [pluggedValid](#)

### 6.43.1 Detailed Description

#### [Todo](#)

documentation

Definition at line 81 of file Power.h.

### 6.43.2 Constructor & Destructor Documentation

#### 6.43.2.1 [PowerFeatureProvider::PowerFeatureProvider](#) ([providerparams](#) & [params](#))

Constructor.

#### Parameters:

*params* Map of initialization parameters

Definition at line 103 of file Power.cpp.



### 6.43.3 Member Function Documentation

#### 6.43.3.1 [Feature](#) \* PowerFeatureProvider::getFeature (string *name*) const [virtual]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the requested feature.

Implements [FeatureProvider](#).

Definition at line 111 of file Power.cpp.

#### 6.43.3.2 [stringvector](#)\* FeatureProvider::getFeatureList () const [inline, inherited]

Query list of provided features.

**Returns:**

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by FeatureContainer::loadFeature().

#### 6.43.3.3 [string](#) FeatureProvider::getName () const [inline, inherited]

Query provider name.

**Returns:**

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

#### 6.43.3.4 [Feature](#) \* PowerFeatureProvider::getSample (string *name*) const [virtual]

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the sample.

**See also:**

[nextSample](#)

Implements [FeatureProvider](#).

Definition at line 120 of file Power.cpp.

#### 6.43.3.5 void [PowerFeatureProvider::nextSample](#) (clock\_t *checkpoint*) [virtual]

Prepare sample.

Prepare the feature provider to return the next sample from the sensor when [getSample\(\)](#) is called.

##### Parameters:

*checkpoint* This parameter tells the feature extractor that it must not return sensordata sampled earlier than the given clock value.

##### See also:

[getSample](#)

Implements [FeatureProvider](#).

Definition at line 28 of file PowerLinux.cpp.

References [plugged](#), and [pluggedValid](#).

### 6.43.4 Member Data Documentation

#### 6.43.4.1 bool [PowerFeatureProvider::plugged](#) [private]

##### [Todo](#)

documentation

Definition at line 89 of file Power.h.

Referenced by [nextSample\(\)](#).

#### 6.43.4.2 bool [PowerFeatureProvider::pluggedValid](#) [private]

##### [Todo](#)

documentation

Definition at line 90 of file Power.h.

Referenced by [nextSample\(\)](#).

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

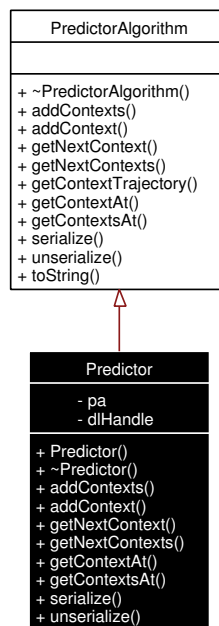
- [Power.h](#)
- [Power.cpp](#)
- [PowerLinux.cpp](#)
- [PowerWindows.cpp](#)

## 6.44 Predictor Class Reference

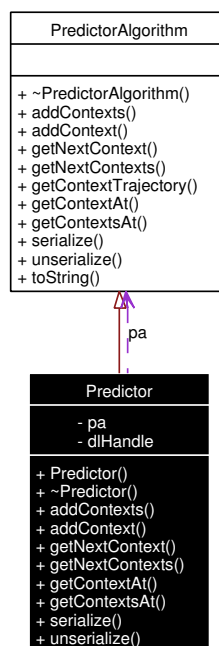
Predictor Wrapper.

```
#include <Predictor.h>
```

Inheritance diagram for Predictor:



Collaboration diagram for Predictor:



## Public Member Functions

- [Predictor](#) (const string &name, const [predictorparams](#) &params)

*Creates the classifier.*

- virtual [~Predictor](#) ()

*Destructor.*

- virtual void [addContexts](#) (const [membershiplist](#) \*contexts, time\_t time)
- virtual void [addContext](#) (unsigned long contextId, time\_t time)
- virtual unsigned long [getNextContext](#) () const
- virtual [membershiplist](#) [getNextContexts](#) () const
- virtual unsigned long [getContextAt](#) (time\_t time) const
- virtual [membershiplist](#) [getContextsAt](#) (time\_t time) const
- virtual string [serialize](#) () const

*Serialize a predictors data to a string.*

- virtual void [unserialize](#) (string data)

*Unserialize a predictors data from a string.*

## Private Member Functions

- virtual [contexttrajectory](#) [getContextTrajectory](#) (unsigned int start, unsigned int end) const =0
- virtual string [toString](#) () const =0

*This is only for testing.*

## Private Attributes

- [PredictorAlgorithm](#) \* pa

*Implementation.*

- void \* [dlHandle](#)

*Library handle.*

### 6.44.1 Detailed Description

Predictor Wrapper.

The Predictor Class is a wrapper class that loads a dll containing the actual implementation during runtime.

Definition at line 140 of file Predictor.h.

## 6.44.2 Constructor & Destructor Documentation

### 6.44.2.1 Predictor::Predictor (const string & *name*, const [predictorparams](#) & *params*)

Creates the classifier.

**Parameters:**

*name* The name of the requested classifier implementation.

*params* A map of parameters to initialize the classifier.

Definition at line 26 of file WindowsPredictor.cpp.

References [getPredictor\(\)](#), [getPredictor\\_t](#), [pa](#), and [predictorparams](#).

## 6.44.3 Member Function Documentation

### 6.44.3.1 virtual void Predictor::addContext (unsigned long *contextId*, time\_t *time*) [inline, virtual]

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 166 of file Predictor.h.

### 6.44.3.2 virtual void Predictor::addContexts (const [membershiplist](#) \* *contexts*, time\_t *time*) [inline, virtual]

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 165 of file Predictor.h.

### 6.44.3.3 virtual unsigned long Predictor::getContextAt (time\_t *time*) const [inline, virtual]

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 169 of file Predictor.h.

### 6.44.3.4 virtual [membershiplist](#) Predictor::getContextsAt (time\_t *time*) const [inline, virtual]

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 170 of file Predictor.h.

References [PredictorAlgorithm::addContexts\(\)](#), [membershipList](#), and [pa](#).

**6.44.3.5** `virtual contextTrajectory PredictorAlgorithm::getContextTrajectory (unsigned int start, unsigned int end) const` [pure virtual, inherited]

**Todo**

documentation

Implemented in [ActiveLezi](#), [HMM](#), [AveragePredictor](#), and [SingleStepDurationPredictor](#).

Referenced by [PredWrapper::predictAndUpdateMultiStep\(\)](#).

**6.44.3.6** `virtual unsigned long Predictor::getNextContext () const` [inline, virtual]

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 167 of file Predictor.h.

**6.44.3.7** `virtual membershipList Predictor::getNextContexts () const` [inline, virtual]

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 168 of file Predictor.h.

**6.44.3.8** `virtual string Predictor::serialize () const` [inline, virtual]

Serialize a predictors data to a string.

**Returns:**

String representation of the predictor data.

Implements [PredictorAlgorithm](#).

Definition at line 171 of file Predictor.h.

References [PredictorAlgorithm::addContext\(\)](#), and [pa](#).

**6.44.3.9** `virtual void Predictor::unserialize (string data)` [inline, virtual]

Unserialize a predictors data from a string.

**Parameters:**

*data* String representation of the predictor data.

Implements [PredictorAlgorithm](#).

Definition at line 172 of file Predictor.h.

References [PredictorAlgorithm::getNextContext\(\)](#), and [pa](#).

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

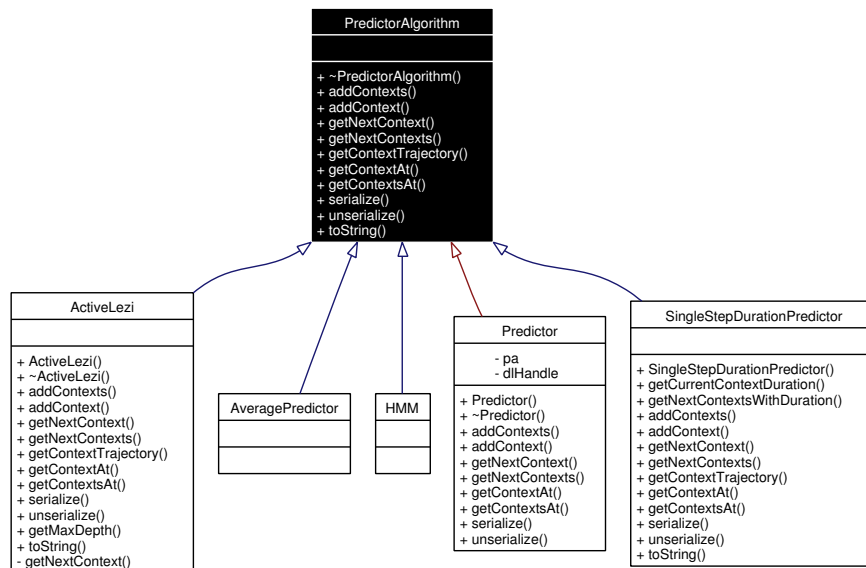
- [Predictor.h](#)
- [WindowsPredictor.cpp](#)

## 6.45 PredictorAlgorithm Class Reference

[Predictor](#) Interface.

```
#include <Predictor.h>
```

Inheritance diagram for PredictorAlgorithm:



### Public Member Functions

- virtual void [addContexts](#) (const [membershiplist](#) \*contexts, time\_t time)=0
- virtual void [addContext](#) (unsigned long contextId, time\_t time)=0
- virtual unsigned long [getNextContext](#) () const =0
- virtual [membershiplist](#) [getNextContexts](#) () const =0
- virtual [contexttrajectory](#) [getContextTrajectory](#) (unsigned int start, unsigned int end) const =0
- virtual unsigned long [getContextAt](#) (time\_t time) const =0
- virtual [membershiplist](#) [getContextsAt](#) (time\_t time) const =0
- virtual string [serialize](#) () const =0

*Serialize a predictors data to a string.*

- virtual void [unserialize](#) (string data)=0

*Unserialize a predictors data from a string.*

- virtual string [toString](#) () const =0

*This is only for testing.*

### 6.45.1 Detailed Description

[Predictor](#) Interface.



**Todo**

documentation

Definition at line 82 of file Predictor.h.

## 6.45.2 Member Function Documentation

**6.45.2.1** `virtual void PredictorAlgorithm::addContext (unsigned long contextId, time_t time)`  
[pure virtual]

**Todo**

documentation

Implemented in [ActiveLezi](#), [HMM](#), [Predictor](#), [AveragePredictor](#), and [SingleStepDurationPredictor](#).

Referenced by `PredWrapper::predictAndUpdate()`, `PredWrapper::predictAndUpdateMultiStep()`, and `Predictor::serialize()`.

**6.45.2.2** `virtual void PredictorAlgorithm::addContexts (const membershiplist * contexts, time_t time)` [pure virtual]

**Todo**

documentation

Implemented in [ActiveLezi](#), [HMM](#), [Predictor](#), [AveragePredictor](#), and [SingleStepDurationPredictor](#).

Referenced by `Predictor::getContextsAt()`.

**6.45.2.3** `virtual unsigned long PredictorAlgorithm::getContextAt (time_t time) const` [pure virtual]

**Todo**

documentation

Implemented in [ActiveLezi](#), [HMM](#), [Predictor](#), [AveragePredictor](#), and [SingleStepDurationPredictor](#).

**6.45.2.4** `virtual membershiplist PredictorAlgorithm::getContextsAt (time_t time) const` [pure virtual]

**Todo**

documentation

Implemented in [ActiveLezi](#), [HMM](#), [Predictor](#), [AveragePredictor](#), and [SingleStepDurationPredictor](#).

**6.45.2.5** `virtual contexttrajectory PredictorAlgorithm::getContextTrajectory (unsigned int start, unsigned int end) const` [pure virtual]

**Todo**

documentation

Implemented in [ActiveLezi](#), [HMM](#), [AveragePredictor](#), and [SingleStepDurationPredictor](#).

Referenced by `PredWrapper::predictAndUpdateMultiStep()`.

**6.45.2.6 virtual unsigned long PredictorAlgorithm::getNextContext () const** [pure virtual]**Todo**

documentation

Implemented in [ActiveLezi](#), [HMM](#), [Predictor](#), [AveragePredictor](#), and [SingleStepDurationPredictor](#).

Referenced by [PredWrapper::predictAndUpdate\(\)](#), and [Predictor::unserialize\(\)](#).

**6.45.2.7 virtual [membership](#)list PredictorAlgorithm::getNextContexts () const** [pure virtual]**Todo**

documentation

Implemented in [ActiveLezi](#), [HMM](#), [Predictor](#), [AveragePredictor](#), and [SingleStepDurationPredictor](#).

**6.45.2.8 virtual string PredictorAlgorithm::serialize () const** [pure virtual]

Serialize a predictors data to a string.

**Returns:**

String representation of the predictor data.

Implemented in [ActiveLezi](#), [HMM](#), [Predictor](#), [AveragePredictor](#), and [SingleStepDurationPredictor](#).

**6.45.2.9 virtual void PredictorAlgorithm::unserialize (string *data*)** [pure virtual]

Unserialize a predictors data from a string.

**Parameters:**

*data* String representation of the predictor data.

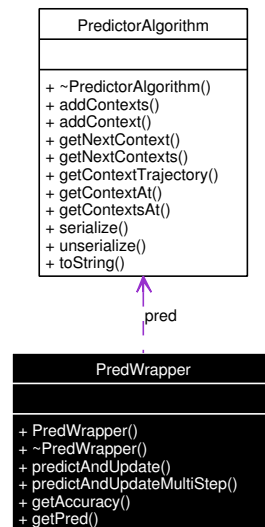
Implemented in [ActiveLezi](#), [HMM](#), [Predictor](#), [AveragePredictor](#), and [SingleStepDurationPredictor](#).

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

- [Predictor.h](#)

## 6.46 PredWrapper Class Reference

Collaboration diagram for PredWrapper:



### Public Member Functions

- [PredWrapper](#) ([PredictorAlgorithm](#) \*pred)
- [~PredWrapper](#) ()  
*Destructor.*
- void [predictAndUpdate](#) (unsigned long curContext)
- void [predictAndUpdateMultiStep](#) ([vector](#)< unsigned int > \*values, unsigned int curPos, unsigned int multiStepOrder)
- double [getAccuracy](#) ()
- const [PredictorAlgorithm](#) \* [getPred](#) ()

### Private Attributes

- [PredictorAlgorithm](#) \* pred
- unsigned int [correctPreds](#)
- unsigned int [totalPreds](#)

#### 6.46.1 Detailed Description

##### Todo

documentation

Definition at line 44 of file main-prediction.cpp.

## 6.46.2 Constructor & Destructor Documentation

### 6.46.2.1 PredWrapper::PredWrapper ([PredictorAlgorithm](#) \* *pred*) [`inline`]

#### [Todo](#)

documentation

Definition at line 54 of file main-prediction.cpp.

References `correctPreds`, and `totalPreds`.

## 6.46.3 Member Function Documentation

### 6.46.3.1 double PredWrapper::getAccuracy () [`inline`]

#### [Todo](#)

documentation

Definition at line 89 of file main-prediction.cpp.

References `correctPreds`, and `totalPreds`.

Referenced by `main()`.

### 6.46.3.2 const [PredictorAlgorithm](#)\* PredWrapper::getPred () [`inline`]

#### [Todo](#)

documentation

Definition at line 92 of file main-prediction.cpp.

References `pred`.

Referenced by `main()`.

### 6.46.3.3 void PredWrapper::predictAndUpdate (unsigned long *curContext*) [`inline`]

#### [Todo](#)

documentation

Definition at line 60 of file main-prediction.cpp.

References `PredictorAlgorithm::addContext()`, `correctPreds`, `PredictorAlgorithm::getNextContext()`, `pred`, and `totalPreds`.

Referenced by `main()`.

### 6.46.3.4 void PredWrapper::predictAndUpdateMultiStep ([vector](#)< unsigned int > \* *values*, unsigned int *curPos*, unsigned int *multiStepOrder*) [`inline`]

#### [Todo](#)

documentation

Definition at line 69 of file main-prediction.cpp.

References `PredictorAlgorithm::addContext()`, `contexttrajectory`, `correctPreds`, `PredictorAlgorithm::getContextTrajectory()`, `pred`, and `totalPreds`.

Referenced by `main()`.

## 6.46.4 Member Data Documentation

### 6.46.4.1 unsigned int `PredWrapper::correctPreds` [private]

#### Todo

documentation

Definition at line 49 of file main-prediction.cpp.

Referenced by `getAccuracy()`, `predictAndUpdate()`, `predictAndUpdateMultiStep()`, and `PredWrapper()`.

### 6.46.4.2 `PredictorAlgorithm*` `PredWrapper::pred` [private]

#### Todo

documentation

Definition at line 48 of file main-prediction.cpp.

Referenced by `getPred()`, `predictAndUpdate()`, `predictAndUpdateMultiStep()`, and `~PredWrapper()`.

### 6.46.4.3 unsigned int `PredWrapper::totalPreds` [private]

#### Todo

documentation

Definition at line 49 of file main-prediction.cpp.

Referenced by `getAccuracy()`, `predictAndUpdate()`, `predictAndUpdateMultiStep()`, and `PredWrapper()`.

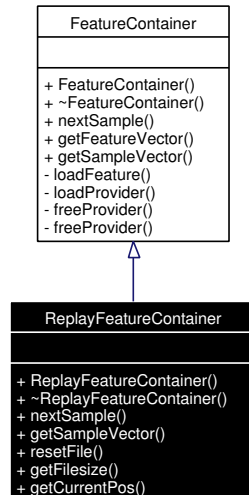
The documentation for this class was generated from the following file:

- [main-prediction.cpp](#)

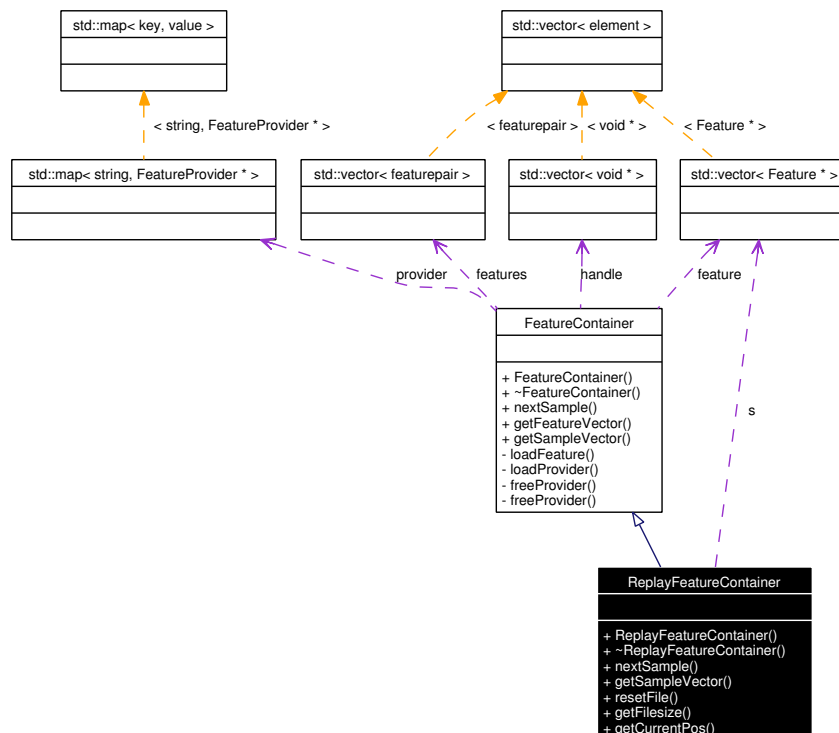
## 6.47 ReplayFeatureContainer Class Reference

```
#include <ReplayFeatureContainer.h>
```

Inheritance diagram for ReplayFeatureContainer:



Collaboration diagram for ReplayFeatureContainer:



**[NOHEADER]**

- typedef [pair](#)< string, [FeatureProvider](#) \* > [featurepair](#)  
*Protected property.*
- typedef [vector](#)< [featurepair](#) > [featuremap](#)  
*Protected property.*
- [featuremap](#) [features](#)  
*Protected property.*

**Public Member Functions**

- [ReplayFeatureContainer](#) ([ConfigReader](#) \*config)  
*Constructor.*
- virtual [~ReplayFeatureContainer](#) ()  
*Destructor.*
- virtual void [nextSample](#) ()  
*Retrieve next sample.*
- virtual const [featurevector](#) \* [getSampleVector](#) () const  
*Get sample vector.*
- void [resetFile](#) ()  
*Reset log file handle position.*
- long [getFilesize](#) ()  
*Get log file size.*
- long [getCurrentPos](#) ()  
*Get log file handle position.*
- virtual [featurevector](#) [getFeatureVector](#) () const  
*Get random feature vector.*

**Private Attributes**

- FILE \* [in](#)
- [featurevector](#) [s](#)

**Related Functions**

(Note that these are not member functions.)

- string [serializeFeatureVector](#) (const [featurevector](#) &samples)

*These are just two helper functions for serializing and deserializing a whole feature vector just returns, for a given feature vector, the serialized, correctly escaped form with ';' as delimiter.*

- `size_t unserializeFeatureVector (const featurevector &features, featurevector &samples, char *data)`  
*parameters: features must contain all \_features\_ that will be unserialized from data, while samples must be empty as this method will fill it with newly created [Feature](#) objects of the correct type (by cloning them from the objects in features) beware that data will be changed by this method ! returns the number of characters processed from dat*

## 6.47.1 Detailed Description

### Todo

documentation

Definition at line 32 of file `ReplayFeatureContainer.h`.

## 6.47.2 Member Typedef Documentation

**6.47.2.1** `typedef vector<featurepair> FeatureContainer::featuremap` [protected, inherited]

Protected property.

### Todo

documentation

Definition at line 110 of file `FeatureContainer.h`.

**6.47.2.2** `typedef pair<string, FeatureProvider*> FeatureContainer::featurepair` [protected, inherited]

Protected property.

### Todo

documentation

Definition at line 109 of file `FeatureContainer.h`.

Referenced by `FeatureContainer::loadFeature()`.

## 6.47.3 Constructor & Destructor Documentation

**6.47.3.1** `ReplayFeatureContainer::ReplayFeatureContainer (ConfigReader * config)`

Constructor.

Replays any transactions stored in the logfile from a [LoggingFeatureContainer](#).

**Parameters:**

*config* A [ConfigReader](#) instance



Definition at line 28 of file ReplayFeatureContainer.cpp.

References ConfigReader::getGlobals(), and in.

## 6.47.4 Member Function Documentation

### 6.47.4.1 **featurevector** FeatureContainer::getFeatureVector () const [virtual, inherited]

Get random feature vector.

A feature vector contains pointers to implementations of the specific features, but initialized with random values. These random positions in the feature space are usually used for initializing classification algorithms. The feature objects returned by this method are allocated automatically for the caller (since the caller can not know the specific implementations, this method acts as a factory), but **must** be freed by the caller. The specific feature objects are created by the FeatureContainer implementations, as this method calls getFeature for every dimension.

#### Returns:

A vector of [Feature](#) implementations. Every element of the returned vector must be freed after use.

#### See also:

[Feature](#)

FeatureContainer::getFeature

Definition at line 179 of file FeatureContainer.cpp.

References FeatureContainer::features, and featurevector.

Referenced by nextSample(), and Scanner::run().

### 6.47.4.2 **const featurevector \*** ReplayFeatureContainer::getSampleVector () const [virtual]

Get sample vector.

A sample vector contains pointers to implementations of the specific features, with values representing the current sensor values. For performance reasons (since this method can be called by multiple callers in the same time step for retrieving the current feature values), this method returns a pointer to a globally allocated object. Callers **must not** free or modify either the returned vector reference or the feature implementations contained therein.

#### Returns:

A reference to a globally allocated vector containing features that represent the current sensor values.

Reimplemented from [FeatureContainer](#).

Definition at line 71 of file ReplayFeatureContainer.cpp.

References featurevector, and s.

### 6.47.4.3 **void** ReplayFeatureContainer::nextSample () [virtual]

Retrieve next sample.

Retrieve next samples from all features at once. This method should be called at the beginning of each time step. This method calls nextSample for each [FeatureContainer](#) and then allocates the global feature vector

which can be returned by successive calls to `getSampleVector`. The specific feature objects are created by the [FeatureContainer](#) implementations, as this method calls `getSample` for every dimension.

**See also:**

[FeatureContainer::nextSample](#)  
[FeatureContainer::getSample](#)  
[getSampleVector](#)

Reimplemented from [FeatureContainer](#).

Definition at line 40 of file `ReplayFeatureContainer.cpp`.

References `featurevector`, `FeatureContainer::getFeatureVector()`, `in`, `s`, and `unserializeFeatureVector()`.

## 6.47.5 Member Data Documentation

### 6.47.5.1 [featuremap](#) [FeatureContainer::features](#) [protected, inherited]

Protected property.

**Todo**

documentation

Definition at line 112 of file `FeatureContainer.h`.

Referenced by `FeatureContainer::FeatureContainer()`, `FeatureContainer::getFeatureVector()`, `FeatureContainer::loadFeature()`, and `FeatureContainer::nextSample()`.

### 6.47.5.2 **FILE\*** [ReplayFeatureContainer::in](#) [private]

**Todo**

documentation

Definition at line 37 of file `ReplayFeatureContainer.h`.

Referenced by `getCurrentPos()`, `getFileSize()`, `nextSample()`, `ReplayFeatureContainer()`, `resetFile()`, and `~ReplayFeatureContainer()`.

### 6.47.5.3 [featurevector](#) [ReplayFeatureContainer::s](#) [private]

**Todo**

documentation

Definition at line 38 of file `ReplayFeatureContainer.h`.

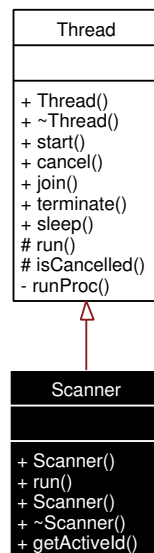
Referenced by `getSampleVector()`, and `nextSample()`.

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

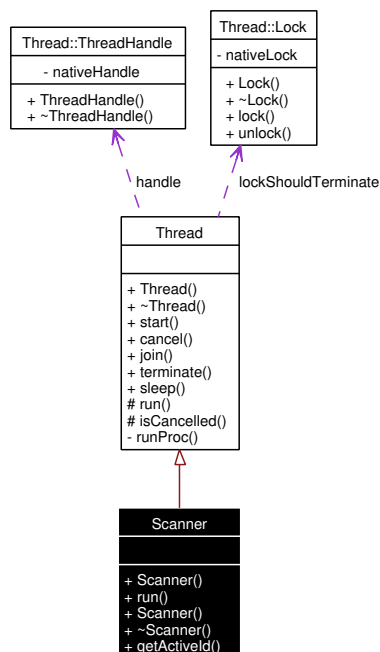
- [ReplayFeatureContainer.h](#)
- [ReplayFeatureContainer.cpp](#)
- [featurevector.h](#)

## 6.48 Scanner Class Reference

Inheritance diagram for Scanner:



Collaboration diagram for Scanner:



### Public Member Functions

- virtual void [run](#) ()

*Thread* main worker function.

- long `getActiveId ()`

## Private Member Functions

- void `start ()`  
*Start thread execution.*
- void `cancel ()`  
*End thread execution.*
- void `join ()`  
*Join thread.*
- void `terminate ()`  
*Terminate thread execution.*
- bool `isCancelled ()`

## Static Private Member Functions

- void `sleep` (unsigned milliseconds)  
*sleep function*

### 6.48.1 Detailed Description

**Todo**

documentation

Definition at line 144 of file main-features.cpp.

### 6.48.2 Member Function Documentation

#### 6.48.2.1 `bool Thread::isCancelled ()` [protected, inherited]

##### Returns:

*true* if the thread is cancelled, *false* if the thread is still running

Definition at line 43 of file Thread.cpp.

References `Thread::cancelled`.

Referenced by `GSMFeatureProvider::nextSample()`, `WlanLinuxFeatureProvider::run()`, `SystemCommandStringListFeatureProvider::run()`, and `BluetoothLinuxFeatureProvider::run()`.

**6.48.2.2 void Thread::sleep (unsigned *milliseconds*)** [static, inherited]

sleep function

**Parameters:**

*milliseconds* Timeout in milliseconds

Definition at line 80 of file ThreadPosix.cpp.

Referenced by Main(), GSMFeatureProvider::readLine(), WlanLinuxFeatureProvider::run(), System-CommandStringListFeatureProvider::run(), run(), and BluetoothLinuxFeatureProvider::run().

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

- [main-features.cpp](#)
- Scanner.h
- LinuxScanner.cpp
- WindowsScanner.cpp

## 6.49 ServiceHost Class Reference

```
#include <servicehost.h>
```

### Static Public Member Functions

- Service \* [getService](#) ()  
*Get Service instance.*

### Protected Member Functions

- [ServiceHost](#) (LPTSTR name, LPTSTR desc)  
*Consturctor.*
- virtual bool **onInstall** ()
- virtual bool **onUninstall** ()
- virtual bool [onInit](#) ()  
*Init service.*
- virtual void [onStop](#) ()  
*Stop service.*
- virtual void [run](#) ()  
*Service main.*

### Static Private Attributes

- bool [keepRunning](#)  
*Flag indicating wheter the service should end or keep running.*

#### 6.49.1 Detailed Description

[Todo](#)

documentation

Definition at line 30 of file servicehost.h.

#### 6.49.2 Constructor & Destructor Documentation

##### 6.49.2.1 ServiceHost::ServiceHost (LPTSTR *name*, LPTSTR *desc*) [*inline*, *protected*]

Consturctor.

**Parameters:**

*name* Servicename

*desc* Description

Definition at line 43 of file servicehost.h.

Referenced by getService().

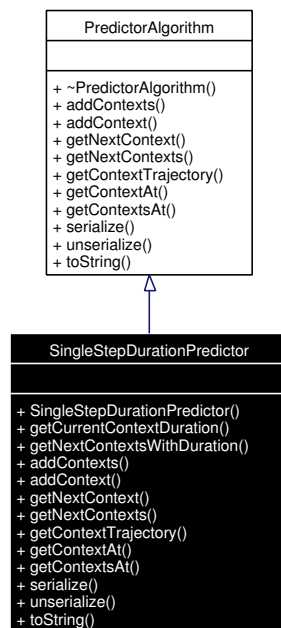
The documentation for this class was generated from the following files:

- [servicehost.h](#)
- [servicehost.cpp](#)

## 6.50 SingleStepDurationPredictor Class Reference

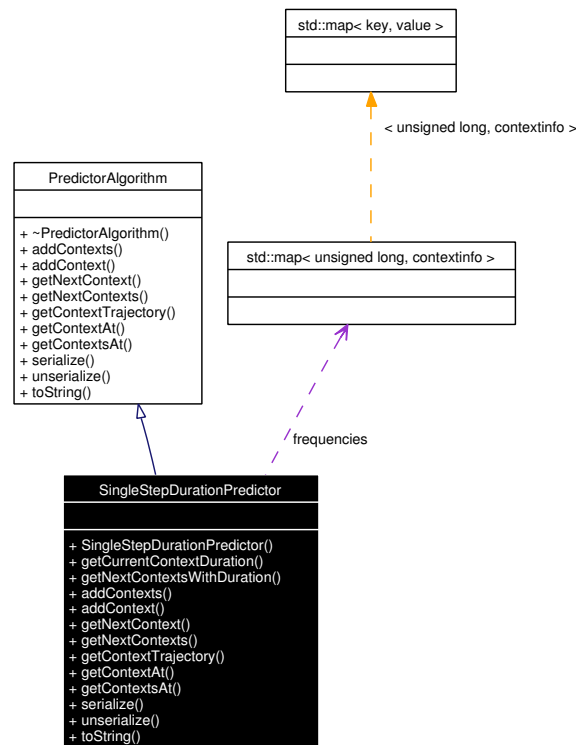
```
#include <SingleStepDuration.h>
```

Inheritance diagram for SingleStepDurationPredictor:



Collaboration diagram for SingleStepDurationPredictor:



**[NOHEADER]**

- typedef [vector](#)< unsigned long > [durationfrequencies](#)
- typedef [map](#)< unsigned long, [durationfrequencies](#) > [nextcontexts](#)
- const unsigned [startfrequencieslength](#) = 20
- const unsigned [incrementfrequencieslength](#) = 10

**Public Member Functions**

- [SingleStepDurationPredictor](#) ([predictorparams](#) &params)
- unsigned long [getCurrentContextDuration](#) () const
- [membershipplist\\_duration](#) [getNextContextsWithDuration](#) (unsigned int offset=0) const
- virtual void [addContexts](#) (const [membershipplist](#) \*contexts, time\_t time)
- virtual void [addContext](#) (unsigned long contextId, time\_t time)
- virtual unsigned long [getNextContext](#) () const
- virtual [membershipplist](#) [getNextContexts](#) () const
- virtual [contexttrajectory](#) [getContextTrajectory](#) (unsigned int start, unsigned int end) const
- virtual unsigned long [getContextAt](#) (time\_t time) const
- virtual [membershipplist](#) [getContextsAt](#) (time\_t time) const
- virtual string [serialize](#) () const  
*Serialize a predictors data to a string.*
- virtual void [unserialize](#) (string data)  
*Unserialize a predictors data from a string.*

- virtual string [toString](#) () const

*This is only for testing.*

## Private Attributes

- [map](#)< unsigned long, [contextinfo](#) > [frequencies](#)
- unsigned long [curContext](#)
- unsigned long [curDuration](#)
- bool [firstEntry](#)
- unsigned int [smoothingWindowWidth](#)
- double [smoothingFactor](#)
- double [peakDifference](#)

### 6.50.1 Detailed Description

#### Todo

documentation

Definition at line 46 of file SingleStepDuration.h.

### 6.50.2 Member Typedef Documentation

**6.50.2.1** typedef [vector](#)<unsigned long> [SingleStepDurationPredictor::durationfrequencies](#) [private]

#### Todo

documentation

Definition at line 51 of file SingleStepDuration.h.

**6.50.2.2** typedef [map](#)<unsigned long, [durationfrequencies](#)> [SingleStepDurationPredictor::nextcontexts](#) [private]

#### Todo

documentation

Definition at line 52 of file SingleStepDuration.h.

Referenced by getNextContextsWithDuration().

### 6.50.3 Constructor & Destructor Documentation

**6.50.3.1** [SingleStepDurationPredictor::SingleStepDurationPredictor](#) ([predictorparams](#) & *params*)

#### Todo

documentation

**Todo**

check parameter !!  
check parameter !!  
check parameter !!

Definition at line 37 of file SingleStepDuration.cpp.

References `DEFAULT_PEAK_THRESHOLD`, `DEFAULT_SMOOTHING_FACTOR`, `DEFAULT_SMOOTHING_WINDOW_WIDTH`, `peakDifference`, `smoothingFactor`, and `smoothingWindowWidth`.

## 6.50.4 Member Function Documentation

### 6.50.4.1 `void SingleStepDurationPredictor::addContext (unsigned long contextId, time_t time) [virtual]`

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 68 of file SingleStepDuration.cpp.

References `curContext`, `curDuration`, `firstEntry`, and `frequencies`.

### 6.50.4.2 `void SingleStepDurationPredictor::addContexts (const membershipList * contexts, time_t time) [virtual]`

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 61 of file SingleStepDuration.cpp.

### 6.50.4.3 `unsigned long SingleStepDurationPredictor::getContextAt (time_t time) const [virtual]`

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 275 of file SingleStepDuration.cpp.

### 6.50.4.4 `membershipList SingleStepDurationPredictor::getContextsAt (time_t time) const [virtual]`

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 282 of file SingleStepDuration.cpp.

**6.50.4.5** [contexttrajectory](#) `SingleStepDurationPredictor::getContextTrajectory (unsigned int start, unsigned int end) const` `[virtual]`

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 236 of file `SingleStepDuration.cpp`.

References `getNextContextsWithDuration()`, and `membershiplist_duration`.

**6.50.4.6** `unsigned long SingleStepDurationPredictor::getCurrentContextDuration () const`

**Todo**

documentation

Definition at line 115 of file `SingleStepDuration.cpp`.

References `curContext`, `frequencies`, and `membershiplist`.

**6.50.4.7** `unsigned long SingleStepDurationPredictor::getNextContext () const` `[virtual]`

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 100 of file `SingleStepDuration.cpp`.

**6.50.4.8** [membershiplist](#) `SingleStepDurationPredictor::getNextContexts () const` `[virtual]`

**Todo**

documentation

Implements [PredictorAlgorithm](#).

Definition at line 120 of file `SingleStepDuration.cpp`.

References `curContext`, `curDuration`, and `frequencies`.

**6.50.4.9** [membershiplist\\_duration](#) `SingleStepDurationPredictor::getNextContextsWithDuration (unsigned int offset = 0) const`

**Todo**

documentation

Definition at line 139 of file `SingleStepDuration.cpp`.

References `frequencies`, and `nextcontexts`.

Referenced by `getContextTrajectory()`.

**6.50.4.10** `string SingleStepDurationPredictor::serialize () const` [virtual]

Serialize a predictors data to a string.

**Returns:**

String representation of the predictor data.

Implements [PredictorAlgorithm](#).

Definition at line 289 of file SingleStepDuration.cpp.

**6.50.4.11** `void SingleStepDurationPredictor::unserialize (string data)` [virtual]

Unserialize a predictors data from a string.

**Parameters:**

*data* String representation of the predictor data.

Implements [PredictorAlgorithm](#).

Definition at line 294 of file SingleStepDuration.cpp.

**6.50.5 Member Data Documentation****6.50.5.1** `unsigned long SingleStepDurationPredictor::curContext` [private]**Todo**

documentation

Definition at line 74 of file SingleStepDuration.h.

Referenced by `addContext()`, `getCurrentContextDuration()`, and `getNextContexts()`.

**6.50.5.2** `unsigned long SingleStepDurationPredictor::curDuration` [private]**Todo**

documentation

Definition at line 75 of file SingleStepDuration.h.

Referenced by `addContext()`, and `getNextContexts()`.

**6.50.5.3** `bool SingleStepDurationPredictor::firstEntry` [private]**Todo**

documentation

Definition at line 76 of file SingleStepDuration.h.

Referenced by `addContext()`.

**6.50.5.4** `map<unsigned long, contextinfo> SingleStepDurationPredictor::frequencies`  
[private]

**Todo**

documentation

Definition at line 73 of file SingleStepDuration.h.

Referenced by `addContext()`, `getCurrentContextDuration()`, `getNextContexts()`, `getNextContextsWithDuration()`, and `toString()`.

**6.50.5.5** `const unsigned SingleStepDurationPredictor::incrementfrequencieslength = 10`  
[static, private]

**Todo**

documentation

Definition at line 55 of file SingleStepDuration.h.

**6.50.5.6** `double SingleStepDurationPredictor::peakDifference` [private]

**Todo**

documentation

Definition at line 81 of file SingleStepDuration.h.

Referenced by `SingleStepDurationPredictor()`.

**6.50.5.7** `double SingleStepDurationPredictor::smoothingFactor` [private]

**Todo**

documentation

Definition at line 80 of file SingleStepDuration.h.

Referenced by `SingleStepDurationPredictor()`.

**6.50.5.8** `unsigned int SingleStepDurationPredictor::smoothingWindowWidth` [private]

**Todo**

documentation

Definition at line 79 of file SingleStepDuration.h.

Referenced by `SingleStepDurationPredictor()`.

**6.50.5.9** `const unsigned SingleStepDurationPredictor::startfrequencieslength = 20` [static, private]

**Todo**

documentation

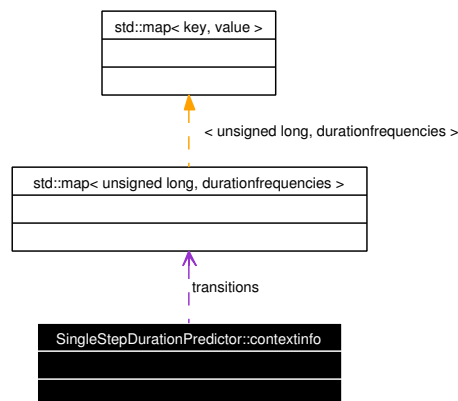
Definition at line 54 of file SingleStepDuration.h.

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

- [SingleStepDuration.h](#)
- [SingleStepDuration.cpp](#)

## 6.51 SingleStepDurationPredictor::contextinfo Struct Reference

Collaboration diagram for SingleStepDurationPredictor::contextinfo:



### [NOHEADER]

- [nextcontexts transitions](#)
- unsigned long **entries**

### 6.51.1 Detailed Description

#### Todo

documentation

Definition at line 61 of file SingleStepDuration.h.

### 6.51.2 Member Data Documentation

#### 6.51.2.1 [nextcontexts SingleStepDurationPredictor::contextinfo::transitions](#)

#### Todo

documentation

Definition at line 64 of file SingleStepDuration.h.

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

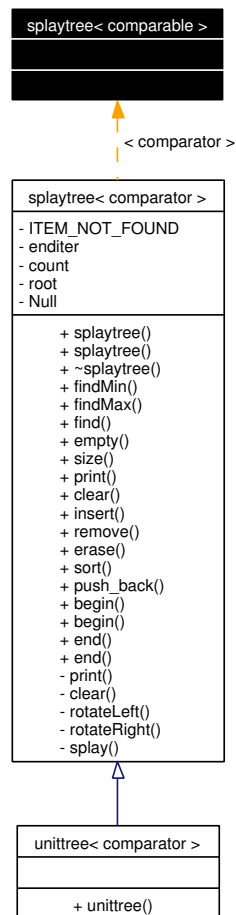
- [SingleStepDuration.h](#)



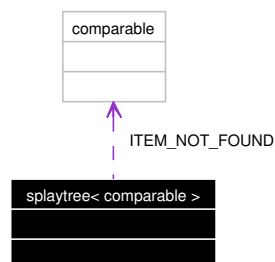
## 6.52 splaytree< comparable > Class Template Reference

```
#include <splaytree.h>
```

Inheritance diagram for splaytree< comparable >:



Collaboration diagram for splaytree< comparable >:



**[NOHEADER]**

- typedef [splaytree\\_iterator](#)< comparable > [iterator](#)
- typedef [splaytree\\_iterator](#)< comparable > [const\\_iterator](#)
- typedef size\_t [size\\_type](#)
- [splaytree](#) (const comparable &notFound)
- [splaytree](#) (const [splaytree](#) &rhs)
- virtual [~splaytree](#) ()
- const comparable & [findMin](#) ()
- const comparable & [findMax](#) ()
- const comparable & [find](#) (const comparable &x)
- bool [empty](#) () const
- [size\\_type](#) [size](#) () const
- void [print](#) () const
- void [clear](#) ()
- void [insert](#) (const comparable &x)
- void [remove](#) (const comparable &x)
- void [erase](#) ([iterator](#) iter)
- void [sort](#) ()
- void [push\\_back](#) (const comparable &x)
- [iterator](#) [begin](#) ()
- [iterator](#) [end](#) ()
- [const\\_iterator](#) [begin](#) () const
- [const\\_iterator](#) [end](#) () const

**[NOHEADER]**

- void **rotateLeft** ([node](#)< comparable > \*&k2) const
- void **rotateRight** ([node](#)< comparable > \*&k1) const
- void **splay** (const comparable &x, [node](#)< comparable > \*&t) const
- void **clear** ([node](#)< comparable > \*n)
- void **print** ([node](#)< comparable > \*n) const
- const comparable [ITEM\\_NOT\\_FOUND](#)
- [iterator](#) **enditer**
- [size\\_type](#) **count**
- [node](#)< comparable > \* **root**
- [node](#)< comparable > \* **Null**

**6.52.1 Detailed Description**

**template<class comparable> class splaytree< comparable >**

**Todo**

documentation

Definition at line 48 of file splaytree.h.

## 6.52.2 Member Typedef Documentation

**6.52.2.1** `template<class comparable> typedef splaytree\_iterator<comparable> splaytree<comparable >::const_iterator`

**Todo**

documentation

Definition at line 53 of file splaytree.h.

**6.52.2.2** `template<class comparable> typedef splaytree\_iterator<comparable> splaytree<comparable >::iterator`

**Todo**

documentation

Definition at line 52 of file splaytree.h.

Referenced by `splaytree< comparator >::begin()`, `splaytree< comparable >::sort()`, and `splaytree< comparable >::splaytree()`.

**6.52.2.3** `template<class comparable> typedef size_t splaytree< comparable >::size_type`

**Todo**

documentation

Definition at line 54 of file splaytree.h.

## 6.52.3 Constructor & Destructor Documentation

**6.52.3.1** `template<class comparable> splaytree< comparable >::splaytree (const comparable &notFound) [explicit]`

**Todo**

documentation

Definition at line 180 of file splaytree.h.

References `splaytree< comparable >::iterator`.

**6.52.3.2** `template<class comparable> splaytree< comparable >::splaytree (const splaytree<comparable > &rhs)`

**Todo**

documentation

**6.52.3.3** `template<class comparable> virtual splaytree< comparable >::~splaytree () [inline, virtual]`

**Todo**

documentation

Definition at line 58 of file splaytree.h.

## 6.52.4 Member Function Documentation

**6.52.4.1** `template<class comparable> const\_iterator splaytree< comparable >::begin \(\) const [inline]`

[Todo](#)

documentation

Definition at line 80 of file splaytree.h.

**6.52.4.2** `template<class comparable> iterator splaytree< comparable >::begin \(\) [inline]`

[Todo](#)

documentation

Definition at line 77 of file splaytree.h.

Referenced by `splaytree< comparable >::sort()`.

**6.52.4.3** `template<class comparable> void splaytree< comparable >::clear \(\) [inline]`

[Todo](#)

documentation

Definition at line 69 of file splaytree.h.

Referenced by `splaytree< comparator >::clear()`, `splaytree< comparable >::sort()`, and `splaytree< comparator >::~~splaytree()`.

**6.52.4.4** `template<class comparable> bool splaytree< comparable >::empty \(\) const [inline]`

[Todo](#)

documentation

Definition at line 64 of file splaytree.h.

**6.52.4.5** `template<class comparable> const\_iterator splaytree< comparable >::end \(\) const [inline]`

[Todo](#)

documentation

Definition at line 81 of file splaytree.h.

**6.52.4.6** `template<class comparable> iterator splaytree< comparable >::end \(\) [inline]`

[Todo](#)

documentation

Definition at line 78 of file splaytree.h.

Referenced by splaytree< comparable >::sort().

**6.52.4.7** `template<class comparable> void splaytree< comparable >::erase (iterator iter)`  
[inline]

**Todo**

documentation

Definition at line 72 of file splaytree.h.

**6.52.4.8** `template<class comparable> const comparable & splaytree< comparable >::find (const comparable & x)`

**Todo**

documentation

Definition at line 326 of file splaytree.h.

References splaytree< comparable >::ITEM\_NOT\_FOUND.

Referenced by splaytree< comparable >::findMax(), and splaytree< comparable >::findMin().

**6.52.4.9** `template<class comparable> const comparable & splaytree< comparable >::findMax ()`

**Todo**

documentation

Definition at line 337 of file splaytree.h.

References node< comparable >::element, splaytree< comparable >::find(), and node< comparable >::right.

**6.52.4.10** `template<class comparable> const comparable & splaytree< comparable >::findMin ()`

**Todo**

documentation

Definition at line 349 of file splaytree.h.

References node< comparable >::element, splaytree< comparable >::find(), and node< comparable >::left.

**6.52.4.11** `template<class comparable> void splaytree< comparable >::insert (const comparable & x)`

**Todo**

documentation

Definition at line 241 of file splaytree.h.

References `node< comparable >::element`, `node< comparable >::left`, `node< comparable >::parent`, and `node< comparable >::right`.

Referenced by `splaytree< comparator >::push_back()`, and `splaytree< comparable >::sort()`.

**6.52.4.12** `template<class comparable> void splaytree< comparable >::print () const` [inline]

**Todo**

documentation

Definition at line 67 of file `splaytree.h`.

Referenced by `splaytree< comparator >::print()`.

**6.52.4.13** `template<class comparable> void splaytree< comparable >::push_back (const comparable & x)` [inline]

**Todo**

documentation

Definition at line 75 of file `splaytree.h`.

**6.52.4.14** `template<class comparable> void splaytree< comparable >::remove (const comparable & x)`

**Todo**

documentation

Definition at line 278 of file `splaytree.h`.

References `node< comparable >::right`.

Referenced by `splaytree< comparator >::erase()`.

**6.52.4.15** `template<class comparable> size_type splaytree< comparable >::size () const` [inline]

**Todo**

documentation

Definition at line 65 of file `splaytree.h`.

**6.52.4.16** `template<class comparable> void splaytree< comparable >::sort ()`

**Todo**

implement splaytree sorting

Definition at line 305 of file `splaytree.h`.

References `splaytree< comparable >::begin()`, `splaytree< comparable >::clear()`, `node< comparable >::element`, `splaytree< comparable >::end()`, `splaytree< comparable >::insert()`, and `splaytree< comparable >::iterator`.

## 6.52.5 Member Data Documentation

**6.52.5.1** `template<class comparable> const comparable splaytree< comparable >::ITEM\_NOT\_FOUND [private]`

### [Todo](#)

documentation

Definition at line 89 of file splaytree.h.

Referenced by `splaytree< comparable >::find()`.

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

- [splaytree.h](#)

## 6.53 splaytree\_iterator< comparable > Class Template Reference

```
#include <splaytree.h>
```

### Public Member Functions

- [splaytree\\_iterator](#) ()
- [splaytree\\_iterator](#) (const [node](#)< comparable > \*root, const [node](#)< comparable > \*n)
- bool [operator!=](#) (const [splaytree\\_iterator](#)< comparable > iter)
- comparable [operator \\*](#) ()
- [splaytree\\_iterator](#)< comparable > \* [operator++](#) (int)
- [splaytree\\_iterator](#)< comparable > \* [operator++](#) ()

### Private Attributes

- long [iterations](#)
- [node](#)< comparable > \* [current](#)
- [node](#)< comparable > \* [Null](#)

#### 6.53.1 Detailed Description

```
template<class comparable> class splaytree_iterator< comparable >
```

**Todo**

documentation

Definition at line 129 of file splaytree.h.

#### 6.53.2 Constructor & Destructor Documentation

**6.53.2.1** `template<class comparable> splaytree\_iterator< comparable >::splaytree\_iterator ()`  
[inline]

**Todo**

documentation

Definition at line 133 of file splaytree.h.

**6.53.2.2** `template<class comparable> splaytree\_iterator< comparable >::splaytree\_iterator (const node< comparable > * root, const node< comparable > * n) [inline]`

**Todo**

documentation

Definition at line 134 of file splaytree.h.

References [splaytree\\_iterator](#)< comparable >::current, [splaytree\\_iterator](#)< comparable >::iterations, and [splaytree\\_iterator](#)< comparable >::Null.



### 6.53.3 Member Function Documentation

**6.53.3.1** `template<class comparable> comparable splaytree_iterator< comparable >::operator *()` `[inline]`

**Todo**

documentation

Definition at line 146 of file splaytree.h.

References splaytree\_iterator< comparable >::current.

**6.53.3.2** `template<class comparable> bool splaytree_iterator< comparable >::operator!=(const splaytree_iterator< comparable > iter)` `[inline]`

**Todo**

documentation

Definition at line 145 of file splaytree.h.

References splaytree\_iterator< comparable >::current.

**6.53.3.3** `template<class comparable> splaytree_iterator<comparable>* splaytree_iterator< comparable >::operator++()` `[inline]`

**Todo**

documentation

Definition at line 148 of file splaytree.h.

References splaytree\_iterator< comparable >::current, splaytree\_iterator< comparable >::iterations, and splaytree\_iterator< comparable >::Null.

**6.53.3.4** `template<class comparable> splaytree_iterator<comparable>* splaytree_iterator< comparable >::operator++(int)` `[inline]`

**Todo**

documentation

Definition at line 147 of file splaytree.h.

### 6.53.4 Member Data Documentation

**6.53.4.1** `template<class comparable> node<comparable>* splaytree_iterator< comparable >::current` `[private]`

**Todo**

documentation

Definition at line 174 of file splaytree.h.

Referenced by splaytree\_iterator< comparable >::operator \*(), splaytree\_iterator< comparable >::operator!==(const splaytree\_iterator< comparable > iter), splaytree\_iterator< comparable >::operator++(), and splaytree\_iterator< comparable >::splaytree\_iterator().

**6.53.4.2** `template<class comparable> long splaytree\_iterator< comparable >::iterations`  
`[private]`

**Todo**

documentation

Definition at line 173 of file `splaytree.h`.

Referenced by `splaytree_iterator< comparable >::operator++()`, and `splaytree_iterator< comparable >::splaytree_iterator()`.

**6.53.4.3** `template<class comparable> node<comparable>* splaytree\_iterator< comparable >::Null`  
`[private]`

**Todo**

documentation

Definition at line 175 of file `splaytree.h`.

Referenced by `splaytree_iterator< comparable >::operator++()`, and `splaytree_iterator< comparable >::splaytree_iterator()`.

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

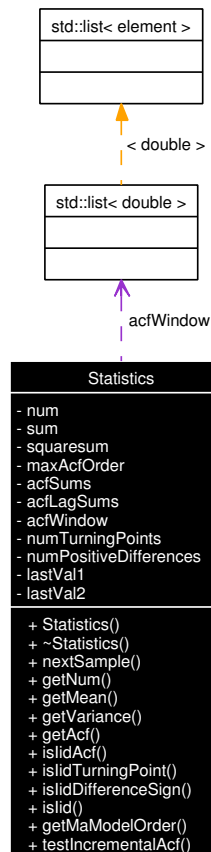
- [splaytree.h](#)

## 6.54 Statistics Class Reference

This class can be used to compute statistics on a `_single_`, numerical continuous variable.

```
#include <Statistics.h>
```

Collaboration diagram for Statistics:



### Public Member Functions

- `Statistics` (unsigned int `maxAcfOrder`, bool `useExactCalculation=true`)  
*If `useExactCalculation` is true, the autocorrelation values will be exact.*
- void `nextSample` (double value)  
*Update the internal helper variables with a new sample.*
- unsigned long `getNum` () const  
*The number of samples.*
- double `getMean` () const  
*The (sample) mean of all sample values.*
- double `getVariance` () const

*The (sample) variance of all sample values.*

- double [getAcf](#) (unsigned int lag) const

*The (sample) autocorrelation of all samples values at the given lag.*

- bool [isIidAcf](#) () const

*Test the time series if it can be regarded as iid noise - independent and identically distributed random variables.*

- bool [isIidTurningPoint](#) () const

*Test the time series if it can be regarded as iid noise - independent and identically distributed random variables.*

- bool [isIidDifferenceSign](#) () const

*Test the time series if it can be regarded as iid noise - independent and identically distributed random variables.*

- bool [isIid](#) () const

*Test the time series if it can be regarded as iid noise - independent and identically distributed random variables.*

- unsigned int [getMaModelOrder](#) () const

*Try to estimate the order  $q$  of an  $MA(q)$  model, under the proposition that the time series indeed can be expressed by an MA model.*

## Static Public Member Functions

- void [testIncrementalAcf](#) (unsigned int [maxAcfOrder](#), double \*values, unsigned int len, double accuracy)

*This can be used for validating the computed ACF values.*

## Private Attributes

- unsigned long [num](#)

*The number of samples until now.*

- double [sum](#)

*The sum of all values until now.*

- double [squaresum](#)

*The square sum of all values until now.*

- const unsigned int [maxAcfOrder](#)

*The maximum lag of the autocorrelation function.*

- double \* [acfSums](#)

*All sums for the autocorrelation function values, from 0 lag to maximum lag.*

- double \* [acfLagSums](#)  
*More helper sums for the autocorrelation function: the sums lagging behind.*
- [list](#) < double > [acfWindow](#)  
*The sliding window for computing the autocorrelation function.*
- unsigned long [numTurningPoints](#)  
*The number of turning points in the series.*
- unsigned long [numPositiveDifferences](#)  
*The number of times the differenced series is positive.*
- double [lastVal1](#)  
*Two two second last values, needed for the turning point and difference sign tests.*
- double [lastVal2](#)  
*Two two second last values, needed for the turning point and difference sign tests.*

### 6.54.1 Detailed Description

This class can be used to compute statistics on a `_single_`, numerical continuous variable.

All computations are performance incrementally with runtime performance in mind.

Definition at line 31 of file `Statistics.h`.

### 6.54.2 Constructor & Destructor Documentation

#### 6.54.2.1 `Statistics::Statistics (unsigned int maxAcfOrder, bool useExactCalculation = true)`

If `useExactCalculation` is true, the autocorrelation values will be exact.

If set to false, an approximation is used, which should be good enough until lag at about 10, and the object will only use 2/3 of the memory on the heap ( $2 * \text{sizeof}(\text{double}) * \text{maxAcfOrder}$  instead of  $3 * \text{sizeof}(\text{double}) * \text{maxAcfOrder}$ ). Please be aware that even with `useExactCalculation`, the ACF will still be off a bit, but at least be exact to an accuracy of  $10^{-3}$  until lag 128.

Setting `maxAcfOrder` to 0 will disable computation of the ACF and thus reduce the runtime complexity of `nextSample` significantly. Definition at line 27 of file `Statistics.cpp`.

References `acfLagSums`, `acfSums`, `num`, `numPositiveDifferences`, `numTurningPoints`, `squaresum`, and `sum`.

### 6.54.3 Member Function Documentation

#### 6.54.3.1 `double Statistics::getAcf (unsigned int lag) const`

The (sample) autocorrelation of all samples values at the given lag.

This function thus represents the (sample) autocorrelation function (ACF). Please see "Introduction to Time Series and Forecasting", page 59 for details. Definition at line 115 of file `Statistics.cpp`.

References `acfLagSums`, `acfSums`, `getMean()`, `getVariance()`, `maxAcfOrder`, `num`, and `sum`.

Referenced by `getMaModelOrder()`, `isIdAcf()`, and `testIncrementalAcf()`.

### 6.54.3.2 unsigned int Statistics::getMaModelOrder () const

Try to estimate the order  $q$  of an  $MA(q)$  model, under the proposition that the time series indeed can be expressed by an MA model.

This method basically returns the highest lag for which the sample ACF value (computed with `getAcf(lag)`) is greater than  $\pm 1.96/\sqrt{\text{getNum()}}$ . It is an implementation of the method described in "Introduction to Time Series and Forecasting", page 94.

**Returns:**

The estimated order of an MA model, at maximum the value of `maxAcfOrder` specified in the constructor (which can also be 0 if the ACF calculation was disabled in that way).

**See also:**

[getAcf](#)

Definition at line 209 of file `Statistics.cpp`.

References `getAcf()`, `maxAcfOrder`, and `num`.

Referenced by `main()`.

### 6.54.3.3 bool Statistics::isIid () const

Test the time series if it can be regarded as *iid* noise - independent and identically distributed random variables.

This test combines all implemented specific tests.

**Returns:**

*true* if the sequence is iid according to the all tests (none returned *false*), *false* otherwise.

Definition at line 204 of file `Statistics.cpp`.

References `isIidAcf()`, `isIidDifferenceSign()`, and `isIidTurningPoint()`.

### 6.54.3.4 bool Statistics::isIidAcf () const

Test the time series if it can be regarded as *iid* noise - independent and identically distributed random variables.

This method uses the sample autocorrelation function (whose values are available with `getAcf`) and checks if all values are within the bounds  $\pm 1.96/\sqrt{\text{getNum()}}$ . This corresponds to a probability of 95% that the sequence is iid according to this test.

**Returns:**

*true* if all values are within the bounds, *false* otherwise.

Definition at line 166 of file `Statistics.cpp`.

References `getAcf()`, `maxAcfOrder`, and `num`.

Referenced by `isIid()`.

### 6.54.3.5 bool Statistics::isIidDifferenceSign () const

Test the time series if it can be regarded as *iid* noise - independent and identically distributed random variables.

This method uses the difference sign test, described in "Introduction to Time Series and Forecasting", page 37f.

If this test returns with *false*, it indicates the presence of an increasing (or decreasing) trend in the data.

#### Returns:

*true* if the sequence is iid according to the test, *false* otherwise.

Definition at line 194 of file Statistics.cpp.

References num, and numPositiveDifferences.

Referenced by isIid(), and main().

### 6.54.3.6 bool Statistics::isIidTurningPoint () const

Test the time series if it can be regarded as *iid* noise - independent and identically distributed random variables.

This method uses the turning point test, described in "Introduction to Time Series and Forecasting", page 36f.

#### Returns:

*true* if the sequence is iid according to the test, *false* otherwise.

Definition at line 184 of file Statistics.cpp.

References num, and numTurningPoints.

Referenced by isIid(), and main().

## 6.54.4 Member Data Documentation

### 6.54.4.1 unsigned long Statistics::numPositiveDifferences [private]

The number of times the differenced series is positive.

#### See also:

[isIidDifferenceSign](#)

Definition at line 56 of file Statistics.h.

Referenced by isIidDifferenceSign(), nextSample(), and Statistics().

### 6.54.4.2 unsigned long Statistics::numTurningPoints [private]

The number of turning points in the series.

#### See also:

[isIidTurningPoint](#)

Definition at line 52 of file Statistics.h.

Referenced by `isIidTurningPoint()`, `nextSample()`, and `Statistics()`.

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

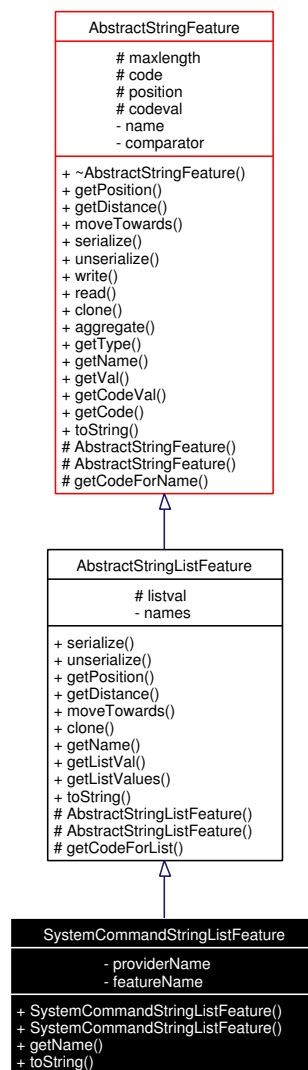
- [Statistics.h](#)
- [Statistics.cpp](#)



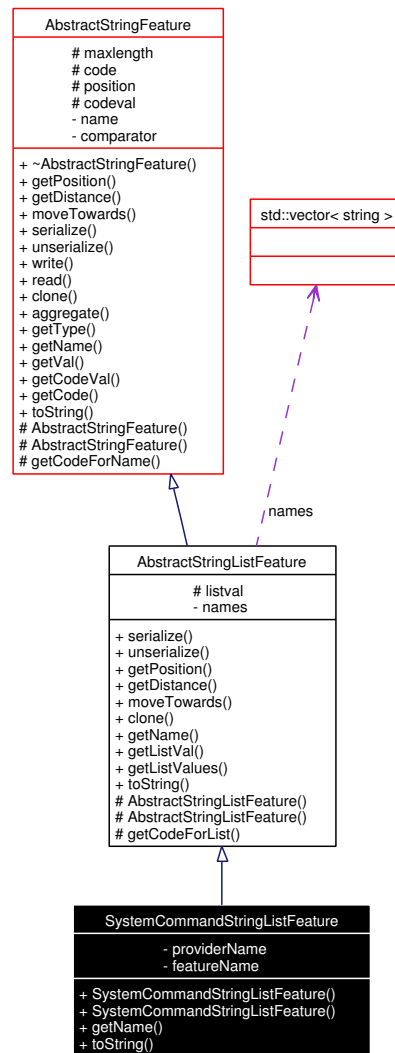
## 6.55 SystemCommandStringListFeature Class Reference

```
#include <SystemCommandStringList.h>
```

Inheritance diagram for SystemCommandStringListFeature:



Collaboration diagram for SystemCommandStringListFeature:



## Public Types

- enum **ComparatorType** { **None**, **Levenshtein** }

*The distance metric to use.*

- enum **FeatureType** {  
**boolean**, **nominal**, **ordinal**, **numerical\_discrete**,  
**numerical\_continuous** }

*Possible feature types.*

## Public Member Functions

- SystemCommandStringListFeature** (**stringcode** \*code, long \*maxlen, const **stringvector** \*names, const string **providerName**, const string **featureName**)

- [SystemCommandStringListFeature](#) ([stringcode](#) \*code, long \*maxlen, const string [providerName](#), const string [featureName](#))
- virtual const string [getName](#) () const  
*Query a features name.*
- virtual string [toString](#) () const  
*Get feature as string.*
- virtual string [serialize](#) () const  
*Serialize a samples data to a string.*
- virtual void [unserialize](#) (string value)  
*Unserialize a samples data from a string.*
- virtual double [getPosition](#) () const  
*Query a features position.*
- virtual double [getDistance](#) ([Feature](#) \*f) const  
*Calculates the distance between two features.*
- virtual void [moveTowards](#) ([Feature](#) \*f, double factor)  
*Move feature.*
- virtual [Feature](#) \* [clone](#) () const  
*Clone a feature.*
- const bit\_vector & [getListVal](#) () const  
*Query the list of values.*
- const [stringvector](#) & [getListValues](#) () const  
*Query the list of values.*
- virtual [featureparams](#) [write](#) () const  
*Externalize feature.*
- virtual void [read](#) ([featureparams](#) \*param)  
*Load feature from persistant data.*
- virtual void [aggregate](#) ([aggregatelist](#) samples)  
*Aggregate a sample values from other sources.*
- virtual [FeatureType](#) [getType](#) () const  
*Query a features type.*
- const string & [getVal](#) () const  
*Query the string values.*
- const unsigned long [getCodeVal](#) () const

- const [stringcode](#) \* [getCode](#) ()
- void [invalidate](#) ()  
*Invalidate feature.*
- void [validate](#) ()  
*Set the validation flag to true.*
- bool [isValid](#) ()  
*Query validation flag.*
- bool [isExternalizable](#) ()  
*Query externalization flag.*

## Protected Member Functions

- bit\_vector [getCodeForList](#) (const [stringvector](#) \*[names](#))  
*Get a code value for a given stringvector.*
- unsigned long [getCodeForName](#) (const string &[name](#))  
*Get a code value for a given string.*

## Protected Attributes

- bit\_vector [listval](#)  
*The coded value of the feature.*
- long \* [maxlength](#)  
*A reference to the static, persistant maximum length of strings seen so far.*
- [stringcode](#) \* [code](#)  
*A reference to the static, persistant code table of strings seen so far.*
- char \* [position](#)  
*Only for internal usage in [moveTowards](#) and [getDistance](#).*
- unsigned long [codeval](#)  
*The coded value of the feature.*
- const bool [externalize](#)  
*Externalization flag.*

## Private Attributes

- const string [providerName](#)
- const string [featureName](#)

## Related Functions

(Note that these are not member functions.)

- long [levenshtein](#) (char \*\*x, const char \*t, double \*factor)  
*Modified Levenshtein algorithm.*

### 6.55.1 Detailed Description

[Todo](#)

documentation

Definition at line 33 of file SystemCommandStringList.h.

### 6.55.2 Constructor & Destructor Documentation

**6.55.2.1** SystemCommandStringListFeature::SystemCommandStringListFeature ([stringcode](#) \*code, long \*maxlen, const [stringvector](#) \*names, const string providerName, const string featureName) [inline]

[Todo](#)

documentation

Definition at line 43 of file SystemCommandStringList.h.

References [stringcode](#), and [stringvector](#).

**6.55.2.2** SystemCommandStringListFeature::SystemCommandStringListFeature ([stringcode](#) \*code, long \*maxlen, const string providerName, const string featureName) [inline]

[Todo](#)

documentation

Definition at line 46 of file SystemCommandStringList.h.

References [stringcode](#).

### 6.55.3 Member Function Documentation

**6.55.3.1** void AbstractStringFeature::aggregate ([aggregatelist](#) samples) [virtual, inherited]

Aggregate a sample values from other sources.

**Parameters:**

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 421 of file AbstractString.cpp.

References [aggregatelist](#).

### 6.55.3.2 **Feature** \* **AbstractStringListFeature::clone () const** [virtual, inherited]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Reimplemented from [AbstractStringFeature](#).

Definition at line 581 of file [AbstractString.cpp](#).

References [AbstractStringListFeature::AbstractStringListFeature\(\)](#), [AbstractStringListFeature::listval](#), and [AbstractStringListFeature::names](#).

### 6.55.3.3 **bit\_vector** **AbstractStringListFeature::getCodeForList (const [stringvector](#) \* *names*)** [protected, inherited]

Get a code value for a given stringvector.

The function composes a `bit_vector` wherein for each string in the stringvector the bit on the position of the strings code value is set to *true*. The remaining bits are initialized to *false*.

#### Parameters:

*names* The stringvector to code

#### Returns:

A code value as `bit_vector`

Definition at line 450 of file [AbstractString.cpp](#).

References [AbstractStringFeature::getCodeForName\(\)](#), and [stringvector](#).

Referenced by [AbstractStringListFeature::AbstractStringListFeature\(\)](#), and [WlanPeersFeature::WlanPeersFeature\(\)](#).

### 6.55.3.4 **unsigned long** **AbstractStringFeature::getCodeForName (const [string](#) & *name*)** [protected, inherited]

Get a code value for a given string.

A helper function to return the respective code for a given string. If the string is encountered the first time a new code is allocated.

#### Parameters:

*name* The string to code

#### Returns:

A code value

Definition at line 303 of file [AbstractString.cpp](#).

References [AbstractStringFeature::code](#), [PersistantFeature::invalidate\(\)](#), and [AbstractStringFeature::maxlength](#).

Referenced by [AbstractStringFeature::AbstractStringFeature\(\)](#), and [AbstractStringListFeature::getCodeForList\(\)](#).

**6.55.3.5** `double AbstractStringListFeature::getDistance (Feature *f) const` [virtual, inherited]

Calculates the distance between two features.

**Parameters:**

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Reimplemented from [AbstractStringFeature](#).

Definition at line 518 of file `AbstractString.cpp`.

References `AbstractStringListFeature::listval`.

**6.55.3.6** `const bit_vector& AbstractStringListFeature::getListVal () const` [inline, inherited]

Query the list of values.

**Returns:**

Values list

Definition at line 244 of file `AbstractString.h`.

**6.55.3.7** `const stringvector& AbstractStringListFeature::getListValues () const` [inline, inherited]

Query the list of values.

**Returns:**

Values list

Definition at line 253 of file `AbstractString.h`.

References `stringvector`.

Referenced by `Java_at_jku_intelligence_samples_StringListSample_nativeGetValues()`, `WlanPeersFeature::toString()`, `toString()`, `BluetoothPeersFeature::toString()`, and `AbstractStringListFeature::toString()`.

**6.55.3.8** `const string SystemCommandStringListFeature::getName () const` [virtual]

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Reimplemented from [AbstractStringListFeature](#).

Definition at line 29 of file `SystemCommandStringList.cpp`.

References `featureName`, and `providerName`.

**6.55.3.9 double AbstractStringListFeature::getPosition () const** [virtual, inherited]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Reimplemented from [AbstractStringFeature](#).

Definition at line 498 of file AbstractString.cpp.

References AbstractStringListFeature::listval.

**6.55.3.10 virtual FeatureType AbstractStringFeature::getType () const** [inline, virtual, inherited]

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 100 of file AbstractString.h.

**6.55.3.11 const string& AbstractStringFeature::getVal () const** [inline, inherited]

Query the string values.

**Returns:**

Values list

Definition at line 108 of file AbstractString.h.

References AbstractStringFeature::name.

Referenced by Java\_at\_jku\_intelligence\_samples\_StringSample\_nativeGetVal().

**6.55.3.12 void PersistentFeature::invalidate ()** [inline, inherited]

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by AbstractStringFeature::getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature::NumericalDiscreteFeature(), TimeFeature::TimeFeature(), NumericalContinuousFeature::unserialize(), and NumericalDiscreteFeature::unserialize().



**6.55.3.13 bool Feature::isExternalizable ()** [inline, inherited]

Query externalization flag.

**Returns:**

*true* if the feature is externalizeable ( i.e. it has persistend data that should be stored across restarts),  
*false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.55.3.14 bool PersistentFeature::isValid ()** [inline, inherited]

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

**6.55.3.15 void AbstractStringListFeature::moveTowards (Feature \**f*, double *factor*)**  
[virtual, inherited]

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

**Parameters:**

*f* [Feature](#) used for distance measurement.

*factor* Distance weight.

Reimplemented from [AbstractStringFeature](#).

Definition at line 546 of file AbstractString.cpp.

References AbstractStringListFeature::listval, and rand\_double.

**6.55.3.16 void AbstractStringFeature::read (featureparams \**param*)** [virtual, inherited]

Load feature from persistent data.

Initializes the persistent feature data from the given representation.

**Parameters:**

*param* Persistent feature data.

**See also:**

[write](#)

Implements [PersistantFeature](#).

Definition at line 296 of file AbstractString.cpp.

References AbstractStringFeature::code, and featureparams.

#### 6.55.3.17 string AbstractStringListFeature::serialize () const [virtual, inherited]

Serialize a samples data to a string.

##### Returns:

String representation of the samples data.

Reimplemented from [AbstractStringFeature](#).

Definition at line 470 of file AbstractString.cpp.

References AbstractStringListFeature::listval.

Referenced by WlanPeersFeature::toString(), BluetoothPeersFeature::toString(), and AbstractStringListFeature::toString().

#### 6.55.3.18 string SystemCommandStringListFeature::toString () const [virtual]

Get feature as string.

##### Note:

This is only for testing.

##### Returns:

[Feature](#) as string.

Reimplemented from [AbstractStringListFeature](#).

Definition at line 37 of file SystemCommandStringList.cpp.

References featureName, and AbstractStringListFeature::getListValues().

#### 6.55.3.19 void AbstractStringListFeature::unserialize (string value) [virtual, inherited]

Unserialize a samples data from a string.

##### Parameters:

*value* String representation of the samples data.

Reimplemented from [AbstractStringFeature](#).

Definition at line 480 of file AbstractString.cpp.

References AbstractStringListFeature::listval.

#### 6.55.3.20 featureparams AbstractStringFeature::write () const [virtual, inherited]

Externalize feature.

**Returns:**

Persistent feature data.

**See also:**

[read](#)

Implements [PersistentFeature](#).

Definition at line 283 of file AbstractString.cpp.

References AbstractStringFeature::code, and featureparams.

## 6.55.4 Friends And Related Function Documentation

### 6.55.4.1 long levenshtein (char \*\* *x*, const char \* *t*, double \* *factor*) [related, inherited]

Modified Levenshtein algorithm.

This function implements a slightly modified version of the Levenshtein algorithm to not only calculate the distance of two strings but also modify the string *x* towards the string *t* with the propability *factor*. In other words, every transformation operation found by the algorithm is performed on the string *x* with the propability *factor*.

**Parameters:**

*x* Input string.

*t* String to compare the input string to.

*factor* Propability with witch transformations are performed. Has to be a value out of the intervall  $[0; 1]$ .

**Returns:**

The levenshtein distance between *x* and *t*.

**Todo**

alloc once

Definition at line 46 of file AbstractString.cpp.

References rand\_double.

Referenced by AbstractStringFeature::getDistance(), and AbstractStringFeature::moveTowards().

## 6.55.5 Member Data Documentation

### 6.55.5.1 unsigned long [AbstractStringFeature::codeval](#) [protected, inherited]

The coded value of the feature.

**See also:**

[name](#)

[code](#)

Definition at line 144 of file AbstractString.h.

Referenced by AbstractStringFeature::AbstractStringFeature(), AbstractStringFeature::clone(), AbstractStringFeature::getCodeVal(), AbstractStringFeature::getDistance(), AbstractStringFeature::moveTowards(), AbstractStringFeature::serialize(), and AbstractStringFeature::unserialize().

**6.55.5.2** `const bool Feature::externalize` [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to `PersistentFeature`, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**

[PersistentFeature](#)

Definition at line 187 of file `Feature.h`.

**6.55.5.3** `const string SystemCommandStringListFeature::featureName` [private]

**Todo**

documentation

Definition at line 39 of file `SystemCommandStringList.h`.

Referenced by `getName()`, and `toString()`.

**6.55.5.4** `bit_vector AbstractStringListFeature::listval` [protected, inherited]

The coded value of the feature.

**See also:**

[names](#)

[code](#)

Definition at line 180 of file `AbstractString.h`.

Referenced by `AbstractStringListFeature::AbstractStringListFeature()`, `AbstractStringListFeature::clone()`, `AbstractStringListFeature::getDistance()`, `AbstractStringListFeature::getPosition()`, `AbstractStringListFeature::moveTowards()`, `AbstractStringListFeature::serialize()`, and `AbstractStringListFeature::unserialize()`.

**6.55.5.5** `const string SystemCommandStringListFeature::providerName` [private]

**Todo**

documentation

Definition at line 37 of file `SystemCommandStringList.h`.

Referenced by `getName()`.

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

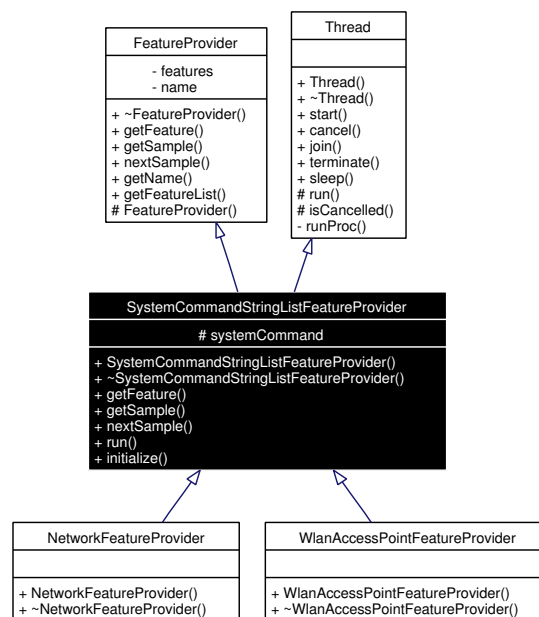
- [SystemCommandStringList.h](#)
- [SystemCommandStringList.cpp](#)

## 6.56 SystemCommandStringListFeatureProvider Class Reference

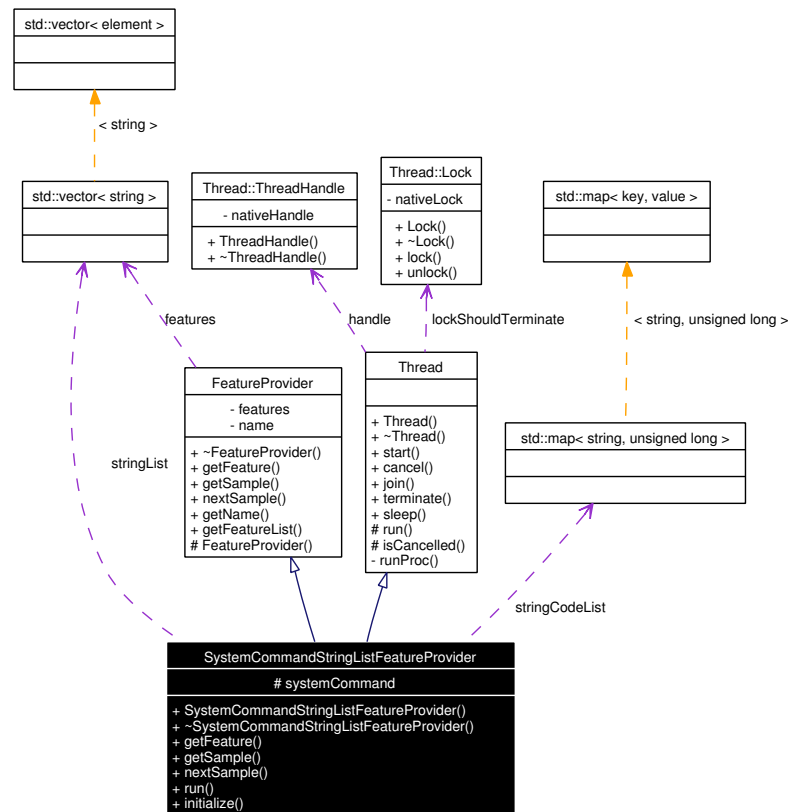
This class returns a feature out of any system command that can return a newline-separated list of strings.

```
#include <SystemCommandStringList.h>
```

Inheritance diagram for SystemCommandStringListFeatureProvider:



Collaboration diagram for SystemCommandStringListFeatureProvider:



## [NOHEADER]

- string **executeSystemCommand** ()
- const string **providerName**
- const string **featureName**
- **stringcode** \* **stringCodeList**
- long \* **maxlen**
- int **delay**
- Lock **lockStringList**

## Public Member Functions

- **SystemCommandStringListFeatureProvider** (string featureProviderName, string featureName, string **systemCommand**, **stringvector** \*featureList, **stringcode** \*stringCodeList, long \*maxlen, **featureparams** &params)

*this->start has to be called in the subclass constructor*

- virtual **~SystemCommandStringListFeatureProvider** () throw ()

*Destructor.*

- virtual **Feature** \* **getFeature** (string **name**) const

*Query feature instance.*

- virtual [Feature](#) \* [getSample](#) (string [name](#)) const  
*Query sample instance.*
- virtual void [nextSample](#) (clock\_t checkpoint)  
*Prepare sample.*
- virtual void [run](#) () throw ()  
*[Thread](#) main worker function.*
- string [getName](#) () const  
*Query provider name.*
- [stringvector](#) \* [getFeatureList](#) () const  
*Query list of provided features.*
- void [start](#) ()  
*Start thread execution.*
- void [cancel](#) ()  
*End thread execution.*
- void [join](#) ()  
*Join thread.*
- void [terminate](#) ()  
*Terminate thread execution.*

## Static Public Member Functions

- void [initialize](#) (string [name](#))
- void [sleep](#) (unsigned milliseconds)  
*sleep function*

## Protected Member Functions

- bool [isCancelled](#) ()

## Protected Attributes

- string [systemCommand](#)  
*Warning: change with care and only when necessary !*
- [stringvector](#) [stringList](#)
- bool [stringListValid](#)

### 6.56.1 Detailed Description

This class returns a feature out of any system command that can return a newline-separated list of strings.  
Definition at line 58 of file SystemCommandStringList.h.

### 6.56.2 Constructor & Destructor Documentation

#### 6.56.2.1 SystemCommandStringListFeatureProvider::SystemCommandStringListFeatureProvider (string *featureProviderName*, string *featureName*, string *systemCommand*, [stringvector](#) \* *featureList*, [stringcode](#) \* *stringCodeList*, long \* *maxlen*, [featureparams](#) & *params*)

this->start has to be called in the subclass constructor

#### [Todo](#)

check parameter

Definition at line 53 of file SystemCommandStringList.cpp.

References [featureparams](#), [stringcode](#), [stringListValid](#), [stringvector](#), and [SYSTEMCOMMAND\\_DELAY](#).

### 6.56.3 Member Function Documentation

#### 6.56.3.1 [Feature](#) \* SystemCommandStringListFeatureProvider::getFeature (string *name*) const [virtual]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

#### Parameters:

*name* Name of the requested feature.

#### Returns:

An instance of the requested feature.

Implements [FeatureProvider](#).

Definition at line 86 of file SystemCommandStringList.cpp.

References [providerName](#).

#### 6.56.3.2 [stringvector](#)\* FeatureProvider::getFeatureList () const [inline, inherited]

Query list of provided features.

#### Returns:

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by [FeatureContainer::loadFeature\(\)](#).



### 6.56.3.3 string FeatureProvider::getName () const [inline, inherited]

Query provider name.

**Returns:**

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

### 6.56.3.4 [Feature](#) \* SystemCommandStringListFeatureProvider::getSample (string name) const [virtual]

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the sample.

**See also:**

[nextSample](#)

Implements [FeatureProvider](#).

Definition at line 95 of file SystemCommandStringList.cpp.

References [providerName](#), [stringList](#), and [stringListValid](#).

### 6.56.3.5 void SystemCommandStringListFeatureProvider::initialize (string name) [static]

**Todo**

documentation

### 6.56.3.6 bool Thread::isCancelled () [protected, inherited]

**Returns:**

*true* if the thread is cancelled, *false* if the thread is still running

Definition at line 43 of file Thread.cpp.

References [Thread::cancelled](#).

Referenced by [GSMFeatureProvider::nextSample\(\)](#), [WlanLinuxFeatureProvider::run\(\)](#), [run\(\)](#), and [BluetoothLinuxFeatureProvider::run\(\)](#).

### 6.56.3.7 void SystemCommandStringListFeatureProvider::nextSample (clock\_t *checkpoint*) [virtual]

Prepare sample.

Prepare the feature provider to return the next sample from the sensor when [getSample\(\)](#) is called.

#### Parameters:

*checkpoint* This parameter tells the feature extractor that it must not return sensordata sampled earlier than the given clock value.

#### See also:

[getSample](#)

Implements [FeatureProvider](#).

Definition at line 106 of file SystemCommandStringList.cpp.

### 6.56.3.8 void Thread::sleep (unsigned *milliseconds*) [static, inherited]

sleep function

#### Parameters:

*milliseconds* Timeout in milliseconds

Definition at line 80 of file ThreadPosix.cpp.

Referenced by Main(), GSMFeatureProvider::readLine(), WlanLinuxFeatureProvider::run(), run(), Scanner::run(), and BluetoothLinuxFeatureProvider::run().

## 6.56.4 Member Data Documentation

### 6.56.4.1 const string SystemCommandStringListFeatureProvider::providerName [private]

#### Todo

documentation

Definition at line 63 of file SystemCommandStringList.h.

Referenced by getFeature(), and getSample().

### 6.56.4.2 stringvector SystemCommandStringListFeatureProvider::stringList [protected]

#### Todo

documentation

Definition at line 102 of file SystemCommandStringList.h.

Referenced by getSample(), and run().

**6.56.4.3**   **bool** [SystemCommandStringListFeatureProvider::stringListValid](#)   [protected]**Todo**

documentation

Definition at line 103 of file SystemCommandStringList.h.

Referenced by [getSample\(\)](#), [run\(\)](#), and [SystemCommandStringListFeatureProvider\(\)](#).

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

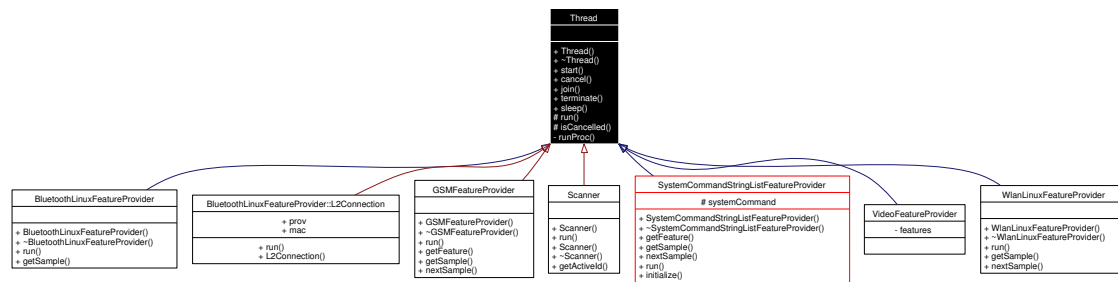
- [SystemCommandStringList.h](#)
- [SystemCommandStringList.cpp](#)

## 6.57 Thread Class Reference

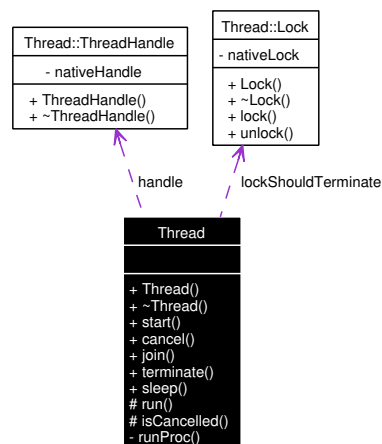
Platform independent thread class.

```
#include <Thread.h>
```

Inheritance diagram for Thread:



Collaboration diagram for Thread:



### Public Member Functions

- **Thread** ()  
*Constructor.*
- virtual **~Thread** ()  
*Destructor.*
- void **start** ()  
*Start thread execution.*
- void **cancel** ()  
*End thread execution.*

- void [join](#) ()  
*Join thread.*
- void [terminate](#) ()  
*Terminate thread execution.*

## Static Public Member Functions

- void [sleep](#) (unsigned milliseconds)  
*sleep function*

## Protected Member Functions

- virtual void [run](#) ()=0  
*Thread main worker function.*
- bool [isCancelled](#) ()

## Static Private Member Functions

- void \* [runProc](#) (void \*userData)  
*Worker function.*

## Private Attributes

- bool [cancelled](#)  
*Thread properties.*
- bool [started](#)  
*Thread properties.*
- [ThreadHandle](#) [handle](#)  
*Thread properties.*
- [Lock](#) [lockShouldTerminate](#)  
*Thread properties.*

### 6.57.1 Detailed Description

Platform independent thread class.

Definition at line 54 of file Thread.h.

## 6.57.2 Member Function Documentation

### 6.57.2.1 `bool Thread::isCancelled ()` [protected]

**Returns:**

*true* if the thread is cancelled, *false* if the thread is still running

Definition at line 43 of file Thread.cpp.

References cancelled.

Referenced by GSMFeatureProvider::nextSample(), WlanLinuxFeatureProvider::run(), SystemCommandStringListFeatureProvider::run(), and BluetoothLinuxFeatureProvider::run().

### 6.57.2.2 `void * Thread::runProc (void * userData)` [static, private]

Worker function.

**Parameters:**

*userData* Thread object

**Returns:**

Status code

Definition at line 29 of file ThreadPosix.cpp.

References run(), and THREAD.

Referenced by start().

### 6.57.2.3 `void Thread::sleep (unsigned milliseconds)` [static]

sleep function

**Parameters:**

*milliseconds* Timeout in milliseconds

Definition at line 80 of file ThreadPosix.cpp.

Referenced by Main(), GSMFeatureProvider::readLine(), WlanLinuxFeatureProvider::run(), SystemCommandStringListFeatureProvider::run(), Scanner::run(), and BluetoothLinuxFeatureProvider::run().

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

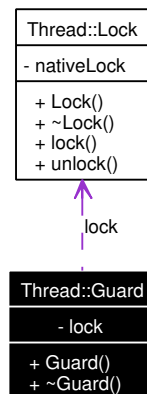
- [Thread.h](#)
- [Thread.cpp](#)
- [ThreadPosix.cpp](#)
- [ThreadSymbian.cpp](#)
- [ThreadWindows.cpp](#)

## 6.58 Thread::Guard Class Reference

[Guard.](#)

```
#include <Thread.h>
```

Collaboration diagram for Thread::Guard:



### Public Member Functions

- [Guard \(Lock &lock\)](#)

*Constructor.*

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

*Destructor.*

### Private Attributes

- [Lock & lock](#)

*lock instance*

### 6.58.1 Detailed Description

[Guard.](#)

Definition at line 97 of file Thread.h.

### 6.58.2 Constructor & Destructor Documentation

#### 6.58.2.1 Thread::Guard::Guard (Lock & lock)

Constructor.

**Parameters:**

*lock* [Lock](#) instance

Definition at line 49 of file Thread.cpp.

References [Thread::Lock::lock\(\)](#).

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

- [Thread.h](#)
- [Thread.cpp](#)



## 6.59 Thread::Lock Class Reference

[Lock.](#)

```
#include <Thread.h>
```

### Public Member Functions

- [Lock](#) ()  
*Constructor.*
- [~Lock](#) ()  
*Destructor.*
- void [lock](#) ()  
*Lock.*
- void [unlock](#) ()  
*Unlock.*

### Private Attributes

- void \* [nativeLock](#)  
*handle*

### 6.59.1 Detailed Description

[Lock.](#)

Definition at line 76 of file Thread.h.

### 6.59.2 Constructor & Destructor Documentation

#### 6.59.2.1 Thread::Lock::Lock ()

Constructor.

**Todo**

leaking 24 bytes

Definition at line 48 of file ThreadPosix.cpp.

References [nativeLock](#).

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

- [Thread.h](#)
- [ThreadPosix.cpp](#)
- [ThreadSymbian.cpp](#)
- [ThreadWindows.cpp](#)

## 6.60 Thread::ThreadHandle Class Reference

Thread handle

```
#include <Thread.h>
```

### Public Member Functions

- [ThreadHandle \(\)](#)

### Private Attributes

- void \* [nativeHandle](#)  
*handle*

#### 6.60.1 Detailed Description

Thread handle

Definition at line 60 of file Thread.h.

#### 6.60.2 Constructor & Destructor Documentation

##### 6.60.2.1 Thread::ThreadHandle::ThreadHandle ()

Definition at line 38 of file ThreadPosix.cpp.

References [nativeHandle](#).

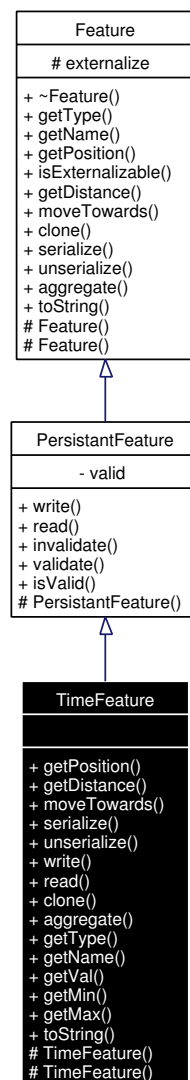
The documentation for this class was generated from the following files:

- [Thread.h](#)
- [ThreadPosix.cpp](#)
- [ThreadSymbian.cpp](#)
- [ThreadWindows.cpp](#)

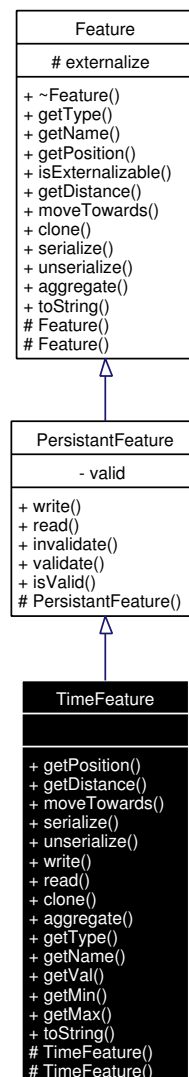
## 6.61 TimeFeature Class Reference

```
#include <TimeFeature.h>
```

Inheritance diagram for TimeFeature:



Collaboration diagram for TimeFeature:



## Public Types

- enum [FeatureKind](#) {  
**Timestamp, Second, Minute, Hour,**  
**Day, Weekday, Yearday, Month,**  
**Year }**  
*Possible kinds of time features.*
- enum [FeatureType](#) {  
**boolean, nominal, ordinal, numerical\_discrete,**  
**numerical\_continuous }**  
*Possible feature types.*

## Public Member Functions

- virtual double [getPosition](#) () const  
*Query a features position.*
- virtual double [getDistance](#) (Feature \*f) const  
*Calculates the distance between two features.*
- virtual void [moveTowards](#) (Feature \*f, double factor)  
*Move feature.*
- virtual string [serialize](#) () const  
*Serialize a samples data to a string.*
- virtual void [unserialize](#) (string value)  
*Unserialize a samples data from a string.*
- virtual [featureparams write](#) () const  
*Externalize feature.*
- virtual void [read](#) (featureparams \*param)  
*Load feature from persistant data.*
- virtual Feature \* [clone](#) () const  
*Clone a feature.*
- virtual void [aggregate](#) (aggregatelist samples)  
*Aggregate a sample values from other sources.*
- virtual FeatureType [getType](#) () const  
*Query a features type.*
- virtual const string [getName](#) () const  
*Query a features name.*
- virtual time\_t [getVal](#) () const
- const time\_t [getMin](#) () const
- const time\_t [getMax](#) () const
- virtual string [toString](#) () const  
*This is only for testing.*
- void [invalidate](#) ()  
*Invalidate feature.*
- void [validate](#) ()  
*Set the validation flag to true.*
- bool [isValid](#) ()  
*Query validation flag.*

- bool [isExternalizable](#) ()  
*Query externalization flag.*

## Protected Member Functions

- [TimeFeature](#) ([FeatureKind](#) kind, time\_t \*[minval](#), time\_t \*[maxval](#))
- [TimeFeature](#) ([FeatureKind](#) kind, time\_t \*[minval](#), time\_t \*[maxval](#), time\_t [timestamp](#))

## Protected Attributes

- const bool [externalize](#)  
*Externalization flag.*

## Private Attributes

- time\_t \* [minval](#)
- time\_t \* [maxval](#)
- time\_t [timestamp](#)
- [FeatureKind](#) [feature](#)

### 6.61.1 Detailed Description

#### [Todo](#)

documentation

Definition at line 32 of file TimeFeature.h.

### 6.61.2 Constructor & Destructor Documentation

#### 6.61.2.1 [TimeFeature::TimeFeature](#) ([FeatureKind](#) kind, time\_t \* *minval*, time\_t \* *maxval*) [protected]

#### [Todo](#)

documentation

Definition at line 37 of file Time.cpp.

#### 6.61.2.2 [TimeFeature::TimeFeature](#) ([FeatureKind](#) kind, time\_t \* *minval*, time\_t \* *maxval*, time\_t *timestamp*) [protected]

#### [Todo](#)

documentation

Definition at line 47 of file Time.cpp.

References [PersistantFeature::invalidate\(\)](#).

### 6.61.3 Member Function Documentation

#### 6.61.3.1 void TimeFeature::aggregate ([aggregatelist](#) *samples*) [virtual]

Aggregate a sample values from other sources.

**Parameters:**

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 323 of file Time.cpp.

References [feature](#).

#### 6.61.3.2 [Feature](#) \* TimeFeature::clone () const [virtual]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Definition at line 318 of file Time.cpp.

References [aggregatelist](#).

#### 6.61.3.3 double TimeFeature::getDistance ([Feature](#) \* *f*) const [virtual]

Calculates the distance between two features.

**Parameters:**

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Definition at line 63 of file Time.cpp.

References [timestamp](#).

#### 6.61.3.4 const string TimeFeature::getName () const [virtual]

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Implements [Feature](#).

Definition at line 327 of file Time.cpp.

**6.61.3.5 double TimeFeature::getPosition () const** [virtual]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Definition at line 205 of file Time.cpp.

References feature, maxval, minval, and timestamp.

**6.61.3.6 virtual FeatureType TimeFeature::getType () const** [inline, virtual]

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 59 of file TimeFeature.h.

**6.61.3.7 virtual time\_t TimeFeature::getVal () const** [inline, virtual]**Returns:**

The value of the feature.

Definition at line 63 of file TimeFeature.h.

Referenced by evaluate().

**6.61.3.8 void PersistantFeature::invalidate ()** [inline, inherited]

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistant feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by AbstractStringFeature::getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature::NumericalDiscreteFeature(), TimeFeature(), NumericalContinuousFeature::unserialize(), and NumericalDiscreteFeature::unserialize().

**6.61.3.9 bool Feature::isExternalizable ()** [inline, inherited]

Query externalization flag.



**Returns:**

*true* if the feature is externalizeable ( i.e. it has persistend data that should be stored across restarts),  
*false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.61.3.10 bool PersistentFeature::isValid () [inline, inherited]**

Query validation flag.

**Returns:**

*true* if the features persistant data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

**6.61.3.11 void TimeFeature::moveTowards (Feature \**f*, double *factor*) [virtual]**

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample s.

**Parameters:**

*f* Feature used for distance measurement.

*factor* Distance weight.

Implements Feature.

Definition at line 253 of file Time.cpp.

**6.61.3.12 void TimeFeature::read (featureparams \**param*) [virtual]**

Load feature from persistant data.

Initializes the persistant feature data from the given representation.

**Parameters:**

*param* Persistant feature data.

**See also:**

[write](#)

Implements PersistentFeature.

Definition at line 162 of file Time.cpp.

References minval.

**6.61.3.13** `string TimeFeature::serialize () const` [virtual]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [Feature](#).

Definition at line 302 of file Time.cpp.

**6.61.3.14** `void TimeFeature::unserialize (string value)` [virtual]

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Definition at line 310 of file Time.cpp.

**6.61.3.15** `featureparams TimeFeature::write () const` [virtual]

Externalize feature.

**Returns:**

Persistent feature data.

**See also:**

[read](#)

Implements [PersistentFeature](#).

Definition at line 183 of file Time.cpp.

**6.61.4 Member Data Documentation****6.61.4.1** `const bool Feature::externalize` [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to [PersistentFeature](#), because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**

[PersistentFeature](#)

Definition at line 187 of file Feature.h.

**6.61.4.2 FeatureKind TimeFeature::feature** [private]**Todo**

documentation

Definition at line 80 of file TimeFeature.h.

Referenced by aggregate(), getPosition(), and toString().

**6.61.4.3 time\_t\* TimeFeature::maxval** [private]**Todo**

documentation

Definition at line 77 of file TimeFeature.h.

Referenced by getPosition().

**6.61.4.4 time\_t\* TimeFeature::minval** [private]**Todo**

documentation

Definition at line 76 of file TimeFeature.h.

Referenced by getPosition(), and read().

**6.61.4.5 time\_t TimeFeature::timestamp** [private]**Todo**

documentation

Definition at line 79 of file TimeFeature.h.

Referenced by getDistance(), and getPosition().

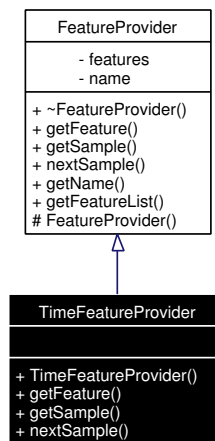
The documentation for this class was generated from the following files:

- [TimeFeature.h](#)
- [Time.cpp](#)

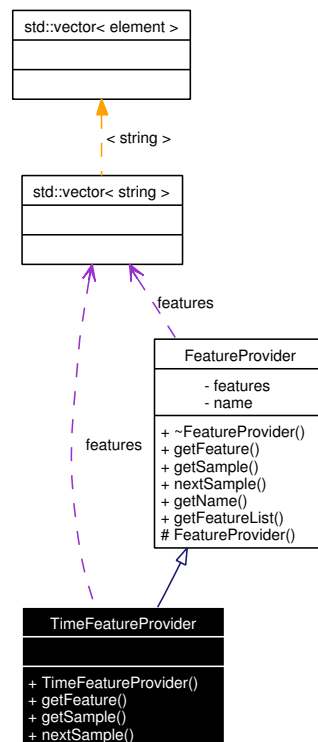
## 6.62 TimeFeatureProvider Class Reference

```
#include <TimeFeature.h>
```

Inheritance diagram for TimeFeatureProvider:



Collaboration diagram for TimeFeatureProvider:



## Public Member Functions

- [TimeFeatureProvider](#) ([providerparams](#) &[params](#))
- virtual [Feature](#) \* [getFeature](#) (string [name](#)) const  
*Query feature instance.*
- virtual [Feature](#) \* [getSample](#) (string [name](#)) const  
*Query sample instance.*
- virtual void [nextSample](#) (clock\_t [checkpoint](#))  
*Prepare sample.*
- string [getName](#) () const  
*Query provider name.*
- [stringvector](#) \* [getFeatureList](#) () const  
*Query list of provided features.*

## Private Attributes

- time\_t [timeFeature\\_maxval](#)
- time\_t [timeFeature\\_minval](#)
- [stringvector](#) [features](#)
- [map](#)< string, [TimeFeature::FeatureKind](#) > [featuremap](#)
- time\_t [timestamp](#)

### 6.62.1 Detailed Description

#### Todo

documentation

Definition at line 87 of file [TimeFeature.h](#).

### 6.62.2 Constructor & Destructor Documentation

#### 6.62.2.1 [TimeFeatureProvider::TimeFeatureProvider](#) ([providerparams](#) & [params](#))

#### Todo

documentation

Definition at line 411 of file [Time.cpp](#).

### 6.62.3 Member Function Documentation

#### 6.62.3.1 [Feature](#) \* [TimeFeatureProvider::getFeature](#) (string [name](#)) const [virtual]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the requested feature.

Implements [FeatureProvider](#).

Definition at line 441 of file Time.cpp.

**6.62.3.2 [stringvector](#)\* FeatureProvider::getFeatureList () const** [inline, inherited]

Query list of provided features.

**Returns:**

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by FeatureContainer::loadFeature().

**6.62.3.3 [string](#) FeatureProvider::getName () const** [inline, inherited]

Query provider name.

**Returns:**

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

**6.62.3.4 [Feature](#)\* TimeFeatureProvider::getSample (string *name*) const** [virtual]

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the sample.

**See also:**

[nextSample](#)

Implements [FeatureProvider](#).

Definition at line 451 of file Time.cpp.

**6.62.3.5 void TimeFeatureProvider::nextSample (clock\_t *checkpoint*)** [virtual]

Prepare sample.

Prepare the feature provider to return the next sample from the sensor when [getSample\(\)](#) is called.

**Parameters:**

*checkpoint* This parameter tells the feature extractor that it must not return sensor data sampled earlier than the given clock value.

**See also:**

[getSample](#)

Implements [FeatureProvider](#).

Definition at line 461 of file Time.cpp.

**6.62.4 Member Data Documentation****6.62.4.1 map<string, TimeFeature::FeatureKind> TimeFeatureProvider::featuremap**  
[private]**Todo**

documentation

Definition at line 96 of file TimeFeature.h.

**6.62.4.2 stringvector TimeFeatureProvider::features** [private]**Todo**

documentation

Reimplemented from [FeatureProvider](#).

Definition at line 95 of file TimeFeature.h.

**6.62.4.3 time\_t TimeFeatureProvider::timeFeature\_maxval** [private]**Todo**

documentation

Definition at line 93 of file TimeFeature.h.

**6.62.4.4 time\_t TimeFeatureProvider::timeFeature\_minval** [private]**Todo**

documentation

Definition at line 94 of file TimeFeature.h.

**6.62.4.5**   `time_t TimeFeatureProvider::timestamp`   [private]

**Todo**

documentation

Definition at line 98 of file TimeFeature.h.

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

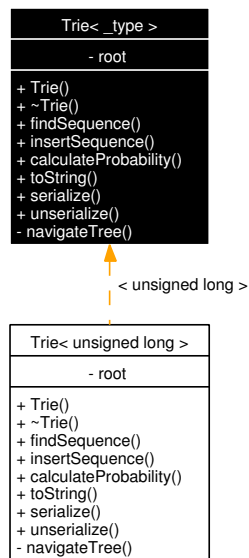
- [TimeFeature.h](#)
- [Time.cpp](#)



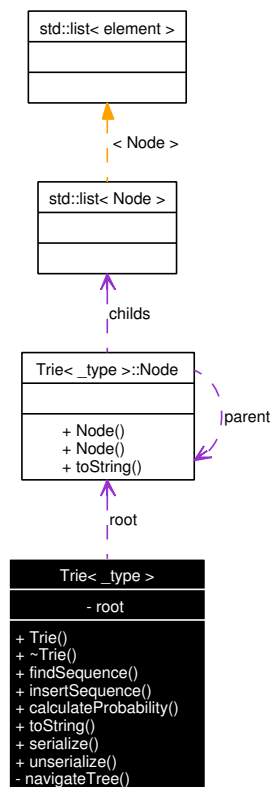
## 6.63 Trie<\_type> Class Template Reference

```
#include <trie.h>
```

Inheritance diagram for Trie<\_type>:



Collaboration diagram for Trie<\_type>:



## Public Member Functions

- [Trie](#) ()
- [~Trie](#) ()
- bool [findSequence](#) (const [list](#)< \_type > \*sequence) const
- void [insertSequence](#) (const [list](#)< \_type > \*sequence, bool updateFrequency=false)
- double [calculateProbability](#) (const [list](#)< \_type > \*context, const \_type nextValue) const
- string [toString](#) () const
- string [serialize](#) () const
- void [unserialize](#) (string data)

## Private Member Functions

- const [Node](#) \* [navigateTree](#) (const [list](#)< \_type > \*sequence) const

## Private Attributes

- [Node](#) root

## 6.63.1 Detailed Description

`template<typename _type> class Trie<_type>`

**Todo**

documentation

Definition at line 35 of file trie.h.

## 6.63.2 Constructor & Destructor Documentation

**6.63.2.1** `template<typename _type> Trie<_type>::Trie ()`

**Todo**

documentation

Definition at line 129 of file trie.h.

**6.63.2.2** `template<typename _type> Trie<_type>::~Trie ()`

**Todo**

documentation

Definition at line 132 of file trie.h.

## 6.63.3 Member Function Documentation

**6.63.3.1** `template<typename _type> double Trie<_type>::calculateProbability (const list<_type> * context, const _type nextValue) const`

**Todo**

documentation

Definition at line 167 of file trie.h.

References `Trie<_type>::Node::childs`, `Trie<_type>::Node::frequency`, `Trie<_type>::navigateTree()`, `Trie<_type>::Node::parent`, and `Trie<_type>::root`.

Referenced by `ActiveLezi::getNextContext()`.

**6.63.3.2** `template<typename _type> bool Trie<_type>::findSequence (const list<_type> * sequence) const`

**Todo**

documentation

Definition at line 135 of file trie.h.

References `Trie<_type>::navigateTree()`.

Referenced by `ActiveLezi::getContextTrajectory()`.

**6.63.3.3** `template<typename _type> void Trie<_type>::insertSequence (const list<_type> * sequence, bool updateFrequency = false)`

**Todo**

documentation

Definition at line 141 of file trie.h.

References `Trie<_type>::Node::childs`, `Trie<_type>::Node::frequency`, and `Trie<_type>::root`.

**6.63.3.4** `template<typename _type> const Node* Trie<_type>::navigateTree (const list<_type> * sequence) const` [`inline`, `private`]

**Todo**

documentation

Definition at line 107 of file trie.h.

Referenced by `Trie<_type>::calculateProbability()`, and `Trie<_type>::findSequence()`.

**6.63.3.5** `template<typename _type> string Trie<_type>::serialize () const`

**Todo**

documentation

Definition at line 219 of file trie.h.

**6.63.3.6** `template<typename _type> string Trie<_type>::toString () const`

**Todo**

documentation

Definition at line 213 of file trie.h.

References `Trie<_type>::root`, and `Trie<_type>::Node::toString()`.

**6.63.3.7** `template<typename _type> void Trie<_type>::unserialize (string data)`

**Todo**

documentation

Definition at line 226 of file trie.h.

## 6.63.4 Member Data Documentation

**6.63.4.1** `template<typename _type> Node Trie<_type>::root` [`private`]

**Todo**

documentation

Definition at line 78 of file trie.h.

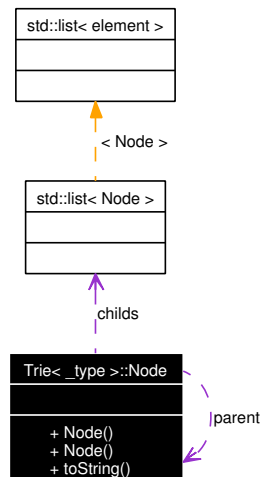
Referenced by `Trie<_type>::calculateProbability()`, `Trie<_type>::insertSequence()`, and `Trie<_type>::toString()`.

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

- [trie.h](#)

## 6.64 Trie< \_type >::Node Class Reference

Collaboration diagram for Trie< \_type >::Node:



### Public Member Functions

- [Node](#) ()
- [Node](#) (const \_type val, const [Node](#) \*parent)
- string [toString](#) (unsigned int level) const

### Public Attributes

- const \_type [value](#)
- const [Node](#) \* [parent](#)
- unsigned long [frequency](#)
- [list](#)< [Node](#) > [childs](#)

#### 6.64.1 Detailed Description

```
template<typename _type> class Trie< _type >::Node
```

**Todo**

documentation

Definition at line 40 of file trie.h.

#### 6.64.2 Constructor & Destructor Documentation

**6.64.2.1** template<typename \_type> [Trie](#)< \_type >::Node::Node () [inline]

**Todo**

documentation

Definition at line 53 of file trie.h.

References `Trie< _type >::Node::frequency`, `Trie< _type >::Node::parent`, and `Trie< _type >::Node::value`.

**6.64.2.2** `template<typename _type> Trie< _type >::Node::Node (const _type val, const Node * parent) [inline]`

**Todo**

documentation

Definition at line 58 of file trie.h.

References `Trie< _type >::Node::frequency`, and `Trie< _type >::Node::value`.

### 6.64.3 Member Function Documentation

**6.64.3.1** `template<typename _type> string Trie< _type >::Node::toString (unsigned int level) const [inline]`

**Todo**

documentation

Definition at line 63 of file trie.h.

References `Trie< _type >::Node::childs`, `Trie< _type >::Node::frequency`, and `Trie< _type >::Node::value`.

Referenced by `Trie< _type >::toString()`.

### 6.64.4 Member Data Documentation

**6.64.4.1** `template<typename _type> list<Node> Trie< _type >::Node::childs`

**Todo**

documentation

Definition at line 48 of file trie.h.

Referenced by `Trie< _type >::calculateProbability()`, `Trie< _type >::insertSequence()`, and `Trie< _type >::Node::toString()`.

**6.64.4.2** `template<typename _type> unsigned long Trie< _type >::Node::frequency`

**Todo**

documentation

Definition at line 47 of file trie.h.

Referenced by `Trie< _type >::calculateProbability()`, `Trie< _type >::insertSequence()`, `Trie< _type >::Node::Node()`, and `Trie< _type >::Node::toString()`.

**6.64.4.3** `template<typename _type> const Node* Trie<_type>::Node::parent`**Todo**

documentation

Definition at line 46 of file trie.h.

Referenced by `Trie<_type>::calculateProbability()`, and `Trie<_type>::Node::Node()`.

**6.64.4.4** `template<typename _type> const _type Trie<_type>::Node::value`**Todo**

documentation

Definition at line 45 of file trie.h.

Referenced by `Trie<_type>::Node::Node()`, and `Trie<_type>::Node::toString()`.

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

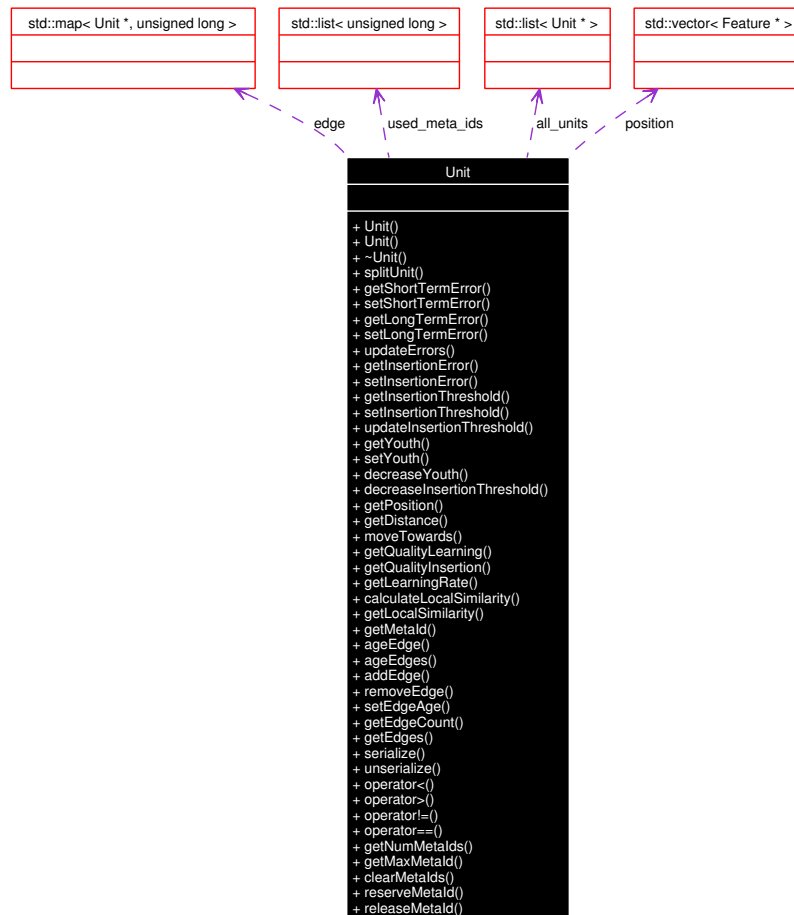
- [trie.h](#)



## 6.65 Unit Class Reference

```
#include <Unit.h>
```

Collaboration diagram for Unit:



### [NOHEADER]

- void **insertEdge** (**Unit** \*u)
- void **eraseEdge** (**Unit** \*u)
- void **movePositionTowards** (const **featurevector** \*sample, bool isWinner)
- void **setMetaId** (unsigned long id)
- void **updateMetaId** ()
- bool **isConnectedHelper** (**Unit** \*u) const
- bool **isConnected** (**Unit** \*u) const
- void **updateRadius** ()
- **unit\_list** const \*const **all\_units**
- unsigned long **meta\_id**
- **featurevector** **position**
- **edge\_map** **edge**
- double **error\_short**

- double **error\_long**
- double **error\_insert**
- double **insertion\_threshold**
- double **youth**
- double **radius**
- bool **traversing**
- double **helper\_local\_similarity**
- unsigned long **getNextMetaId** ()
- [list](#)< unsigned long > **used\_meta\_ids**

## Public Member Functions

- [Unit](#) ([unit\\_list](#) const \*const units, const [featurevector](#) pos)
- [Unit](#) ([unit\\_list](#) const \*const units, const [featurevector](#) pos, unsigned long id)
- virtual [~Unit](#) ()

*Destructor.*

- [Unit](#) \* [splitUnit](#) ()
- double [getShortTermError](#) () const
- void [setShortTermError](#) (double error)
- double [getLongTermError](#) () const
- void [setLongTermError](#) (double error)
- void [updateErrors](#) (const [featurevector](#) \*sample)
- double [getInsertionError](#) () const
- void [setInsertionError](#) (double error)
- double [getInsertionThreshold](#) () const
- void [setInsertionThreshold](#) (double threshold)
- void [updateInsertionThreshold](#) ()
- double [getYouth](#) () const
- void [setYouth](#) (double youth)
- void [decreaseYouth](#) ()
- void [decreaseInsertionThreshold](#) ()
- const [featurevector](#) \* [getPosition](#) () const
- double [getDistance](#) (const [featurevector](#) \*sample) const
- void [moveTowards](#) (const [featurevector](#) \*sample)
- double [getQualityLearning](#) () const
- double [getQualityInsertion](#) () const
- double [getLearningRate](#) (bool isWinner) const
- void [calculateLocalSimilarity](#) () const
- double [getLocalSimilarity](#) () const
- unsigned long [getMetaId](#) () const
- void [ageEdge](#) ([Unit](#) \*u)
- void [ageEdges](#) ()
- void [addEdge](#) ([Unit](#) \*u)
- void [removeEdge](#) ([Unit](#) \*u)
- void [setEdgeAge](#) ([Unit](#) \*u, unsigned long age)
- [edge\\_map](#)::size\_type [getEdgeCount](#) () const
- [edge\\_map](#) \* [getEdges](#) ()
- string [serialize](#) () const
- size\_t [unserialize](#) (const [featurevector](#) &features, const string data)

- bool `operator<` (`Unit *u`)
- bool `operator>` (`Unit *u`)
- bool `operator!=` (`Unit *u`)
- bool `operator==` (`Unit *u`)

## Static Public Member Functions

- `size_t` `getNumMetaIds` ()
- `unsigned long` `getMaxMetaId` ()
- void `clearMetaIds` ()
- void `reserveMetaId` (`unsigned long meta_id`)
- void `releaseMetaId` (`unsigned long meta_id`)

### 6.65.1 Detailed Description

#### Todo

documentation

Definition at line 65 of file Unit.h.

### 6.65.2 Constructor & Destructor Documentation

#### 6.65.2.1 `Unit::Unit (unit_list const *const units, const featurevector pos)`

##### Todo

documentation

Definition at line 29 of file Unit.cpp.

References `featurevector`, and `unit_list`.

Referenced by `splitUnit()`.

#### 6.65.2.2 `Unit::Unit (unit_list const *const units, const featurevector pos, unsigned long id)`

##### Todo

documentation

Definition at line 34 of file Unit.cpp.

References `featurevector`, and `unit_list`.

### 6.65.3 Member Function Documentation

#### 6.65.3.1 `void Unit::addEdge (Unit * u)`

##### Todo

documentation

Definition at line 469 of file Unit.cpp.

References `getMetaId()`, `insertEdge()`, `meta_id`, `releaseMetaId()`, and `setMetaId()`.

Referenced by `splitUnit()`.

**6.65.3.2 void Unit::ageEdge (Unit \* *u*)****Todo**

documentation

Definition at line 441 of file Unit.cpp.

**6.65.3.3 void Unit::ageEdges ()****Todo**

documentation

Definition at line 445 of file Unit.cpp.

References removeEdge(), and unit\_list.

**6.65.3.4 void Unit::calculateLocalSimilarity () const****Todo**

documentation

Definition at line 411 of file Unit.cpp.

References getDistance(), getEdgeCount(), and getLocalSimilarity().

Referenced by getLocalSimilarity(), and moveTowards().

**6.65.3.5 void Unit::clearMetaIds () [static]****Todo**

documentation

Definition at line 716 of file Unit.cpp.

**6.65.3.6 void Unit::decreaseInsertionThreshold ()****Todo**

documentation

Definition at line 266 of file Unit.cpp.

References getQualityLearning().

**6.65.3.7 void Unit::decreaseYouth ()****Todo**

documentation

Definition at line 257 of file Unit.cpp.

**6.65.3.8 double Unit::getDistance (const [featurevector](#) \* *sample*) const****Todo**

documentation

**Todo**

implement the distance from heterogenous kohonen !

Definition at line 294 of file Unit.cpp.

References [featurevector](#).

Referenced by [calculateLocalSimilarity\(\)](#), and [updateErrors\(\)](#).

**6.65.3.9 [edge\\_map::size\\_type](#) Unit::getEdgeCount () const****Todo**

documentation

Definition at line 525 of file Unit.cpp.

Referenced by [calculateLocalSimilarity\(\)](#), [removeEdge\(\)](#), and [splitUnit\(\)](#).

**6.65.3.10 [edge\\_map](#) \* Unit::getEdges ()****Todo**

documentation

Definition at line 529 of file Unit.cpp.

References [edge\\_map](#).

**6.65.3.11 double Unit::getInsertionError () const****Todo**

documentation

Definition at line 205 of file Unit.cpp.

**6.65.3.12 double Unit::getInsertionThreshold () const****Todo**

documentation

Definition at line 217 of file Unit.cpp.

Referenced by [splitUnit\(\)](#).

**6.65.3.13 double Unit::getLearningRate (bool *isWinner*) const****Todo**

documentation

**Todo**

implement getActivation()

Definition at line 394 of file Unit.cpp.

References getQualityLearning().

**6.65.3.14 double Unit::getLocalSimilarity () const****Todo**

documentation

Definition at line 425 of file Unit.cpp.

References calculateLocalSimilarity().

Referenced by calculateLocalSimilarity().

**6.65.3.15 double Unit::getLongTermError () const****Todo**

documentation

Definition at line 161 of file Unit.cpp.

Referenced by splitUnit().

**6.65.3.16 unsigned long Unit::getMaxMetaId () [static]****Todo**

documentation

Definition at line 708 of file Unit.cpp.

**6.65.3.17 unsigned long Unit::getMetaId () const****Todo**

documentation

Definition at line 167 of file Unit.cpp.

Referenced by addEdge().

**6.65.3.18 size\_t Unit::getNumMetaIds () [static]****Todo**

documentation

Definition at line 703 of file Unit.cpp.

**6.65.3.19** `const featurevector * Unit::getPosition () const`**Todo**

documentation

Definition at line 285 of file Unit.cpp.

References [featurevector](#).Referenced by [splitUnit\(\)](#).**6.65.3.20** `double Unit::getQualityInsertion () const`**Todo**

documentation

Definition at line 386 of file Unit.cpp.

**6.65.3.21** `double Unit::getQualityLearning () const`**Todo**

documentation

Definition at line 376 of file Unit.cpp.

Referenced by [decreaseInsertionThreshold\(\)](#), and [getLearningRate\(\)](#).**6.65.3.22** `double Unit::getShortTermError () const`**Todo**

documentation

Definition at line 149 of file Unit.cpp.

Referenced by [splitUnit\(\)](#).**6.65.3.23** `double Unit::getYouth () const`**Todo**

documentation

Definition at line 245 of file Unit.cpp.

**6.65.3.24** `void Unit::moveTowards (const featurevector * sample)`**Todo**

documentation

Definition at line 337 of file Unit.cpp.

References [calculateLocalSimilarity\(\)](#), and [featurevector](#).

**6.65.3.25** `bool Unit::operator!=(Unit * u)` [inline]

**Todo**

documentation

Definition at line 207 of file Unit.h.

**6.65.3.26** `bool Unit::operator<(Unit * u)`

**Todo**

documentation

Definition at line 289 of file Unit.cpp.

References radius.

**6.65.3.27** `bool Unit::operator==(Unit * u)` [inline]

**Todo**

documentation

Definition at line 209 of file Unit.h.

**6.65.3.28** `bool Unit::operator>(Unit * u)` [inline]

**Todo**

documentation

Definition at line 205 of file Unit.h.

**6.65.3.29** `void Unit::releaseMetaId(unsigned long meta_id)` [static]

**Todo**

documentation

Definition at line 697 of file Unit.cpp.

Referenced by addEdge().

**6.65.3.30** `void Unit::removeEdge(Unit * u)`

**Todo**

documentation

Definition at line 497 of file Unit.cpp.

References eraseEdge(), getEdgeCount(), meta\_id, and updateMetaId().

Referenced by ageEdges().



**6.65.3.31 void Unit::reserveMetaId (unsigned long *meta\_id*) [static]****Todo**

documentation

Definition at line 680 of file Unit.cpp.

**6.65.3.32 string Unit::serialize () const****Todo**

documentation

Definition at line 533 of file Unit.cpp.

References serializeFeatureVector().

**6.65.3.33 void Unit::setEdgeAge (Unit \* *u*, unsigned long *age*)****Todo**

documentation

Definition at line 520 of file Unit.cpp.

References edge.

**6.65.3.34 void Unit::setInsertionError (double *error*)****Todo**

documentation

Definition at line 211 of file Unit.cpp.

Referenced by splitUnit().

**6.65.3.35 void Unit::setInsertionThreshold (double *threshold*)****Todo**

documentation

Definition at line 223 of file Unit.cpp.

Referenced by splitUnit().

**6.65.3.36 void Unit::setLongTermError (double *error*)****Todo**

documentation

Definition at line 172 of file Unit.cpp.

Referenced by splitUnit().

**6.65.3.37 void Unit::setShortTermError (double *error*)****Todo**

documentation

Definition at line 155 of file Unit.cpp.

Referenced by splitUnit().

**6.65.3.38 void Unit::setYouth (double *youth*)****Todo**

documentation

Definition at line 251 of file Unit.cpp.

Referenced by splitUnit().

**6.65.3.39 Unit \* Unit::splitUnit ()****Todo**

documentation

Definition at line 73 of file Unit.cpp.

References addEdge(), all\_units, eraseEdge(), featurevector, getEdgeCount(), getInsertionThreshold(), getLongTermError(), getPosition(), getShortTermError(), Feature::moveTowards(), setInsertionError(), setInsertionThreshold(), setLongTermError(), setShortTermError(), setYouth(), Unit(), updateInsertionThreshold(), and updateRadius().

**6.65.3.40 size\_t Unit::unserialize (const [featurevector](#) & *features*, const string *data*)****Todo**

documentation

Definition at line 564 of file Unit.cpp.

References featurevector, strsplit(), and unserializeFeatureVector().

**6.65.3.41 void Unit::updateErrors (const [featurevector](#) \* *sample*)****Todo**

documentation

Definition at line 189 of file Unit.cpp.

References featurevector, and getDistance().

**6.65.3.42 void Unit::updateInsertionThreshold ()****Todo**

documentation

Definition at line 229 of file Unit.cpp.

Referenced by splitUnit().

## 6.65.4 Member Data Documentation

### 6.65.4.1 [unit\\_list](#) const\* const [Unit::all\\_units](#) [private]

#### **Todo**

documentation

Definition at line 72 of file Unit.h.

Referenced by splitUnit().

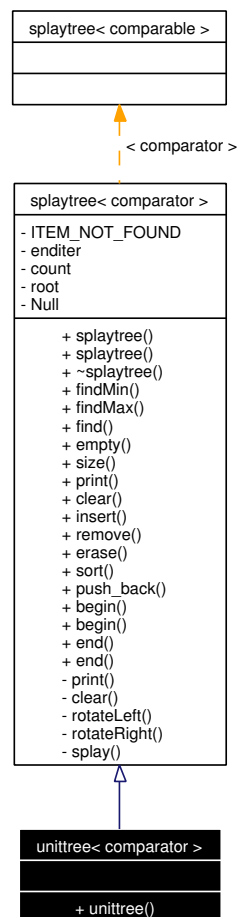
The documentation for this class was generated from the following files:

- [Unit.h](#)
- [Unit.cpp](#)

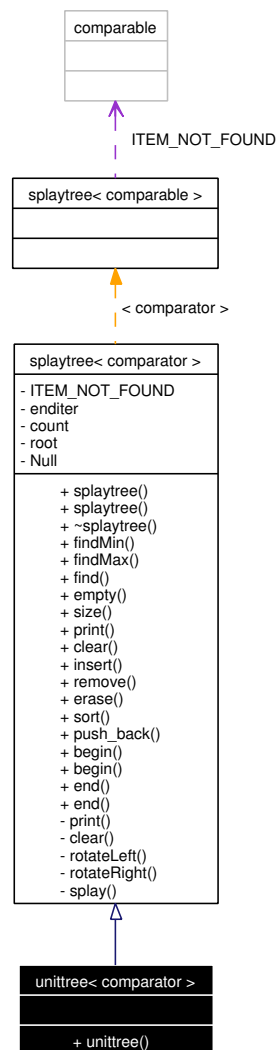
## 6.66 unittree< comparator > Class Template Reference

```
#include <Unit.h>
```

Inheritance diagram for unittree< comparator >:



Collaboration diagram for unittree< comparator >:



## Public Types

- typedef `splaytree_iterator< comparator >` `iterator`
- typedef `splaytree_iterator< comparator >` `const_iterator`
- typedef `size_t` `size_type`

## Public Member Functions

- const comparator & **findMin** ()
- const comparator & **findMax** ()
- const comparator & **find** (const comparator &x)
- bool **empty** () const
- `size_type` **size** () const
- void **print** () const
- void **clear** ()
- void **insert** (const comparator &x)

- void **remove** (const comparator &x)
- void **erase** ([iterator](#) iter)
- void **sort** ()
- void **push\_back** (const comparator &x)
- [iterator](#) **begin** ()
- [const\\_iterator](#) **begin** () const
- [iterator](#) **end** ()
- [const\\_iterator](#) **end** () const

### 6.66.1 Detailed Description

**template**<class comparator> class **unittree**< comparator >

[Todo](#)

documentation

Definition at line 46 of file Unit.h.

### 6.66.2 Member Typedef Documentation

**6.66.2.1** typedef [splaytree\\_iterator](#)<comparator > [splaytree](#)< comparator >::[iterator](#)  
[inherited]

[Todo](#)

documentation

Definition at line 52 of file splaytree.h.

Referenced by [splaytree](#)< comparator >::[begin](#)(), [splaytree](#)< comparator >::[end](#)(), and [splaytree](#)< comparator >::[erase](#)().

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

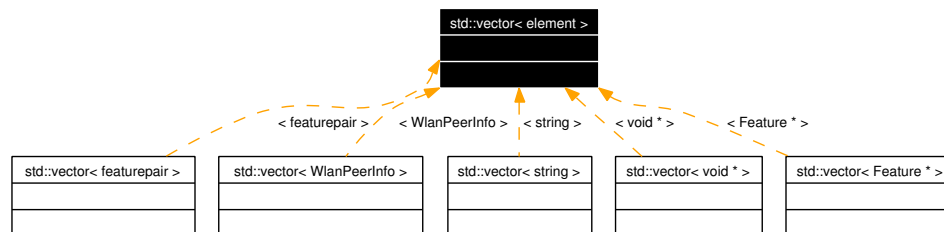
- [Unit.h](#)

## 6.67 std::vector< element > Class Template Reference

STL vector template.

```
#include <doxygen.h>
```

Inheritance diagram for std::vector< element >:



### 6.67.1 Detailed Description

**template<class element> class std::vector< element >**

STL vector template.

Definition at line 32 of file doxygen.h.

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

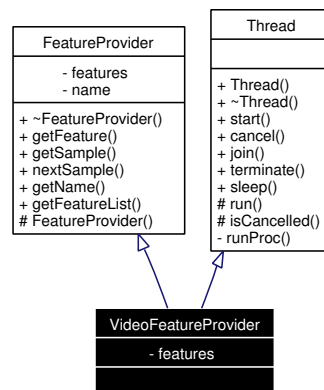
- [doxygen.h](#)

## 6.68 VideoFeatureProvider Class Reference

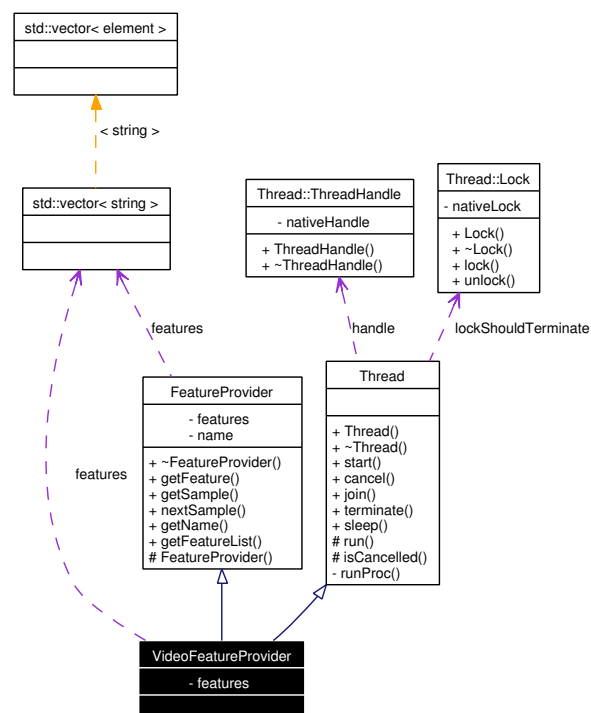
Provider for the video features.

```
#include <Video.h>
```

Inheritance diagram for VideoFeatureProvider:



Collaboration diagram for VideoFeatureProvider:



[NOHEADER]

- [VideoFeatureProvider](#) ([providerparams](#) &[params](#))



- virtual `~VideoFeatureProvider ()`
- virtual `Feature * getFeature (string name) const`
- virtual `Feature * getSample (string name) const`
- virtual void `nextSample (clock_t checkpoint)`
- virtual void `run ()`
- char \* `v4l_start ()`
- string `deviceName`

*Feature properties.*

- int `height`
- int `width`
- int `frequency`
- double `videoMotionFeature_minval`
- double `videoMotionFeature_maxval`
- double `curMotion`
- Lock `lockCurMotion`

## Public Member Functions

- string `getName () const`  
*Query provider name.*
- `stringvector * getFeatureList () const`  
*Query list of provided features.*
- void `start ()`  
*Start thread execution.*
- void `cancel ()`  
*End thread execution.*
- void `join ()`  
*Join thread.*
- void `terminate ()`  
*Terminate thread execution.*

## Static Public Member Functions

- void `sleep (unsigned milliseconds)`  
*sleep function*

## Protected Member Functions

- bool `isCancelled ()`

## Private Attributes

- [stringvector features](#)

*List of features.*

### 6.68.1 Detailed Description

Provider for the video features.

Definition at line 50 of file video.h.

### 6.68.2 Constructor & Destructor Documentation

#### 6.68.2.1 VideoFeatureProvider::VideoFeatureProvider ([providerparams](#) & *params*)

##### Todo

symbian stl fix  
check parameter !!  
check parameter !!  
check parameter !!  
check parameter !!

Definition at line 92 of file Video.cpp.

References [deviceName](#), [frequency](#), [height](#), [videoMotionFeature\\_maxval](#), [videoMotionFeature\\_minval](#), and [width](#).

#### 6.68.2.2 VideoFeatureProvider::~~VideoFeatureProvider () [virtual]

##### Todo

symbian stl fix  
check parameter !!  
check parameter !!  
check parameter !!  
check parameter !!

Definition at line 161 of file Video.cpp.

### 6.68.3 Member Function Documentation

#### 6.68.3.1 [Feature](#) \* VideoFeatureProvider::getFeature (string *name*) const [virtual]

##### Todo

symbian stl fix  
check parameter !!  
check parameter !!  
check parameter !!  
check parameter !!

Implements [FeatureProvider](#).

Definition at line 52 of file Video.cpp.

**6.68.3.2** [stringvector\\*](#) **FeatureProvider::getFeatureList () const** [inline, inherited]

Query list of provided features.

**Returns:**

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by FeatureContainer::loadFeature().

**6.68.3.3** [string](#) **FeatureProvider::getName () const** [inline, inherited]

Query provider name.

**Returns:**

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

**6.68.3.4** [Feature](#) \* **VideoFeatureProvider::getSample (string name) const** [virtual]**Todo**

- symbian stl fix
- check parameter !!
- check parameter !!
- check parameter !!
- check parameter !!

Implements [FeatureProvider](#).

Definition at line 61 of file Video.cpp.

**6.68.3.5** [bool](#) **Thread::isCancelled ()** [protected, inherited]**Returns:**

*true* if the thread is cancelled, *false* if the thread is still running

Definition at line 43 of file Thread.cpp.

References Thread::cancelled.

Referenced by GSMFeatureProvider::nextSample(), WlanLinuxFeatureProvider::run(), SystemCommandStringListFeatureProvider::run(), and BluetoothLinuxFeatureProvider::run().

**6.68.3.6** [void](#) **VideoFeatureProvider::nextSample (clock\_t checkpoint)** [virtual]**Todo**

- symbian stl fix
- check parameter !!
- check parameter !!
- check parameter !!
- check parameter !!

Implements [FeatureProvider](#).

Definition at line 71 of file Video.cpp.

References features, and providerparams.

#### 6.68.3.7 void VideoFeatureProvider::run () [virtual]

##### Todo

- symbian stl fix
- check parameter !!
- check parameter !!
- check parameter !!
- check parameter !!

Implements [Thread](#).

Definition at line 170 of file Video.cpp.

#### 6.68.3.8 void Thread::sleep (unsigned *milliseconds*) [static, inherited]

sleep function

##### Parameters:

*milliseconds* Timeout in milliseconds

Definition at line 80 of file ThreadPosix.cpp.

Referenced by Main(), GSMFeatureProvider::readLine(), WlanLinuxFeatureProvider::run(), System-CommandStringListFeatureProvider::run(), Scanner::run(), and BluetoothLinuxFeatureProvider::run().

#### 6.68.3.9 char \* VideoFeatureProvider::v4l\_start () [private]

##### Todo

- symbian stl fix
- check parameter !!
- check parameter !!
- check parameter !!
- check parameter !!

Definition at line 180 of file Video.cpp.

### 6.68.4 Member Data Documentation

#### 6.68.4.1 double VideoFeatureProvider::curMotion [private]

##### Todo

- symbian stl fix
- check parameter !!
- check parameter !!
- check parameter !!
- check parameter !!

Definition at line 80 of file video.h.

**6.68.4.2** int **VideoFeatureProvider::frequency** [private]**Todo**

- symbian stl fix
- check parameter !!
- check parameter !!
- check parameter !!
- check parameter !!

Definition at line 75 of file video.h.

Referenced by VideoFeatureProvider().

**6.68.4.3** int **VideoFeatureProvider::height** [private]**Todo**

- symbian stl fix
- check parameter !!
- check parameter !!
- check parameter !!
- check parameter !!

Definition at line 75 of file video.h.

Referenced by VideoFeatureProvider().

**6.68.4.4** Lock **VideoFeatureProvider::lockCurMotion** [mutable, private]**Todo**

- symbian stl fix
- check parameter !!
- check parameter !!
- check parameter !!
- check parameter !!

Definition at line 81 of file video.h.

**6.68.4.5** double **VideoFeatureProvider::videoMotionFeature\_maxval** [private]**Todo**

- symbian stl fix
- check parameter !!
- check parameter !!
- check parameter !!
- check parameter !!

Definition at line 78 of file video.h.

Referenced by VideoFeatureProvider().

**6.68.4.6** double **VideoFeatureProvider::videoMotionFeature\_minval** [private]**Todo**

- symbian stl fix

```
check parameter !!  
check parameter !!  
check parameter !!  
check parameter !!
```

Definition at line 77 of file video.h.

Referenced by VideoFeatureProvider().

#### 6.68.4.7 int [VideoFeatureProvider::width](#) [private]

##### [Todo](#)

```
symbian stl fix  
check parameter !!  
check parameter !!  
check parameter !!  
check parameter !!
```

Definition at line 75 of file video.h.

Referenced by VideoFeatureProvider().

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

- [video.h](#)
- [Video.cpp](#)

## 6.69 VideoFeatureProvider::SampleData Struct Reference

### Public Member Functions

- [SampleData \(\)](#)  
*Constructor.*

### Public Attributes

- double **average**
- unsigned long **average\_counter**

#### 6.69.1 Detailed Description

##### **Todo**

documentation

Definition at line 56 of file video.h.

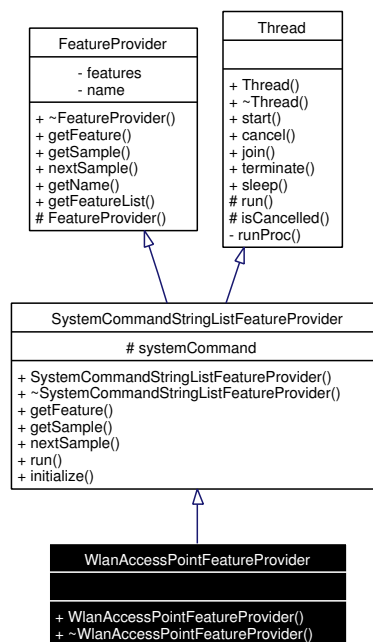
The documentation for this struct was generated from the following file:

- [video.h](#)

## 6.70 WlanAccessPointFeatureProvider Class Reference

```
#include <WlanAP.h>
```

Inheritance diagram for WlanAccessPointFeatureProvider:



Collaboration diagram for WlanAccessPointFeatureProvider:





- string getName () const

*Query provider name.*

- `stringvector * getFeatureList ()` const

*Query list of provided features.*

- void `start ()`

*Start thread execution.*

- void `cancel ()`

*End thread execution.*

- void `join ()`

*Join thread.*

- void `terminate ()`

*Terminate thread execution.*

## Static Public Member Functions

- void `initialize` (string `name`)
- void `sleep` (unsigned milliseconds)

*sleep function*

## Protected Member Functions

- bool `isCancelled ()`

## Protected Attributes

- string `systemCommand`

*Warning: change with care and only when necessary !*

- `stringvector` `stringList`
- bool `stringListValid`

- `stringvector` `stringList`
- bool `stringListValid`

## Private Attributes

- `stringvector` `features`
- `stringcode` `clientsFeature_code`
- long `clientsFeature_maxlen`

## 6.70.1 Detailed Description

### Todo

documentation

Definition at line 32 of file WlanAP.h.

## 6.70.2 Constructor & Destructor Documentation

### 6.70.2.1 WlanAccessPointFeatureProvider::WlanAccessPointFeatureProvider (*featureparams* & *params*)

#### Todo

documentation

#### Todo

check parameter !!

Definition at line 43 of file WlanAP.cpp.

References `DEFAULT_ADDRESS`, `SYSTEM_COMMAND_1_CISCO`, `SYSTEM_COMMAND_1_LINKSYS`, `SYSTEM_COMMAND_2_CISCO`, `SYSTEM_COMMAND_2_LINKSYS`, and `SYSTEM_COMMAND_3_LINKSYS`.

## 6.70.3 Member Function Documentation

### 6.70.3.1 *Feature* \* SystemCommandStringListFeatureProvider::getFeature (string *name*) const [virtual, inherited]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

#### Parameters:

*name* Name of the requested feature.

#### Returns:

An instance of the requested feature.

Implements [FeatureProvider](#).

Definition at line 86 of file SystemCommandStringList.cpp.

References `SystemCommandStringListFeatureProvider::providerName`.

### 6.70.3.2 *stringvector*\* FeatureProvider::getFeatureList () const [inline, inherited]

Query list of provided features.

#### Returns:

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by `FeatureContainer::loadFeature()`.

**6.70.3.3** `string FeatureProvider::getName () const` [inline, inherited]

Query provider name.

**Returns:**

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

**6.70.3.4** `Feature * SystemCommandStringListFeatureProvider::getSample (string name) const` [virtual, inherited]

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the sample.

**See also:**

[nextSample](#)

Implements [FeatureProvider](#).

Definition at line 95 of file SystemCommandStringList.cpp.

References [SystemCommandStringListFeatureProvider::providerName](#), [SystemCommandStringListFeatureProvider::stringList](#), and [SystemCommandStringListFeatureProvider::stringListValid](#).

**6.70.3.5** `void SystemCommandStringListFeatureProvider::initialize (string name)` [static, inherited]**Todo**

documentation

**6.70.3.6** `bool Thread::isCancelled ()` [protected, inherited]**Returns:**

*true* if the thread is cancelled, *false* if the thread is still running

Definition at line 43 of file Thread.cpp.

References [Thread::cancelled](#).

Referenced by [GSMFeatureProvider::nextSample\(\)](#), [WlanLinuxFeatureProvider::run\(\)](#), [SystemCommandStringListFeatureProvider::run\(\)](#), and [BluetoothLinuxFeatureProvider::run\(\)](#).

### 6.70.3.7 void SystemCommandStringListFeatureProvider::nextSample (clock\_t *checkpoint*) [virtual, inherited]

Prepare sample.

Prepare the feature provider to return the next sample from the sensor when [getSample\(\)](#) is called.

#### Parameters:

***checkpoint*** This parameter tells the feature extractor that it must not return sensordata sampled earlier than the given clock value.

#### See also:

[getSample](#)

Implements [FeatureProvider](#).

Definition at line 106 of file SystemCommandStringList.cpp.

### 6.70.3.8 void Thread::sleep (unsigned *milliseconds*) [static, inherited]

sleep function

#### Parameters:

***milliseconds*** Timeout in milliseconds

Definition at line 80 of file ThreadPosix.cpp.

Referenced by [Main\(\)](#), [GSMFeatureProvider::readLine\(\)](#), [WlanLinuxFeatureProvider::run\(\)](#), [SystemCommandStringListFeatureProvider::run\(\)](#), [Scanner::run\(\)](#), and [BluetoothLinuxFeatureProvider::run\(\)](#).

## 6.70.4 Member Data Documentation

### 6.70.4.1 stringcode WlanAccessPointFeatureProvider::clientsFeature\_code [private]

#### Todo

documentation

Definition at line 39 of file WlanAP.h.

### 6.70.4.2 long WlanAccessPointFeatureProvider::clientsFeature\_maxlen [private]

#### Todo

documentation

Definition at line 40 of file WlanAP.h.

### 6.70.4.3 stringvector WlanAccessPointFeatureProvider::features [private]

#### Todo

documentation

Reimplemented from [FeatureProvider](#).

Definition at line 37 of file WlanAP.h.

**6.70.4.4** [stringvector SystemCommandStringListFeatureProvider::stringList](#) [protected, inherited]

**Todo**

documentation

Definition at line 102 of file SystemCommandStringList.h.

Referenced by SystemCommandStringListFeatureProvider::getSample(), and SystemCommandStringListFeatureProvider::run().

**6.70.4.5** [bool SystemCommandStringListFeatureProvider::stringListValid](#) [protected, inherited]

**Todo**

documentation

Definition at line 103 of file SystemCommandStringList.h.

Referenced by SystemCommandStringListFeatureProvider::getSample(), SystemCommandStringListFeatureProvider::run(), and SystemCommandStringListFeatureProvider::SystemCommandStringListFeatureProvider().

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

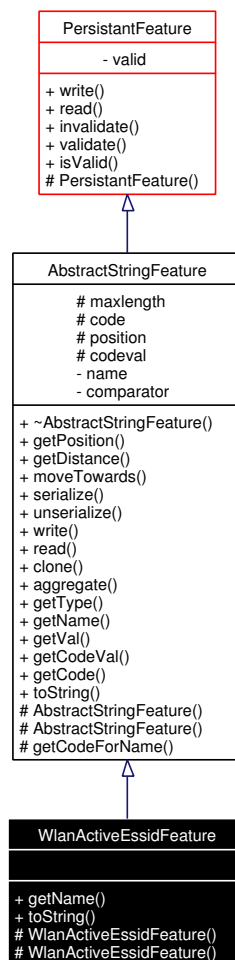
- [WlanAP.h](#)
- [WlanAP.cpp](#)

## 6.71 WlanActiveEssidFeature Class Reference

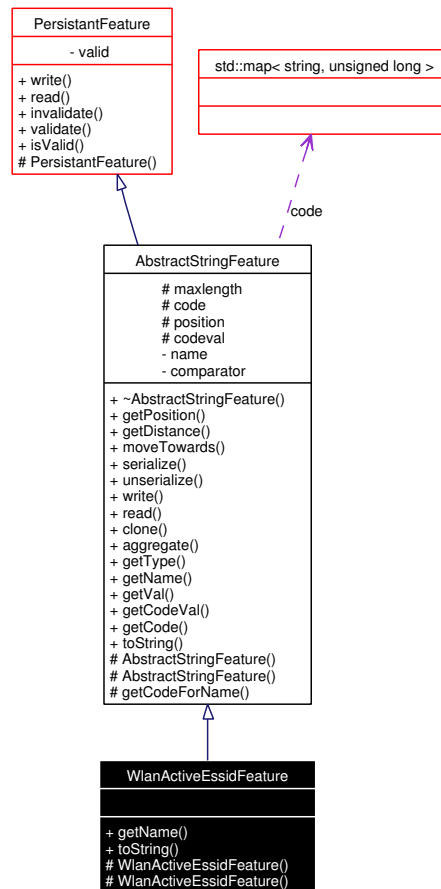
This feature encapsulates the ESSID which is currently set for a WLAN network interface or the ESSID which it is currently bound to (if not set explicitly).

```
#include <Wlan.h>
```

Inheritance diagram for WlanActiveEssidFeature:



Collaboration diagram for WlanActiveEssidFeature:



## Public Types

- enum [ComparatorType](#) { **None**, **Levenshtein** }

*The distance metric to use.*

- enum [FeatureType](#) {  
**boolean**, **nominal**, **ordinal**, **numerical\_discrete**,  
**numerical\_continuous** }

*Possible feature types.*

## Public Member Functions

- virtual const string [getName](#) () const  
*Query a features name.*
- virtual string [toString](#) () const  
*Get feature as string.*
- virtual double [getPosition](#) () const



*Query a features position.*

- virtual double `getDistance` (`Feature *f`) const  
*Calculates the distance between two features.*
- virtual void `moveTowards` (`Feature *f`, double factor)  
*Move feature.*
- virtual string `serialize` () const  
*Serialize a samples data to a string.*
- virtual void `unserialize` (string value)  
*Unserialize a samples data from a string.*
- virtual `featureparams write` () const  
*Externalize feature.*
- virtual void `read` (`featureparams *param`)  
*Load feature from persistant data.*
- virtual `Feature * clone` () const  
*Clone a feature.*
- virtual void `aggregate` (`aggregatelist` samples)  
*Aggregate a sample values from other sources.*
- virtual `FeatureType getType` () const  
*Query a features type.*
- const string & `getVal` () const  
*Query the string values.*
- const unsigned long `getCodeVal` () const
- const `stringcode * getCode` ()
- void `invalidate` ()  
*Invalidate feature.*
- void `validate` ()  
*Set the validation flag to true.*
- bool `isValid` ()  
*Query validation flag.*
- bool `isExternalizable` ()  
*Query externalization flag.*

## Protected Member Functions

- [WlanActiveEssidFeature](#) ([stringcode](#) \*code, long \*maxlen)  
*This constructor should not be used to construct feature objects.*
- [WlanActiveEssidFeature](#) ([stringcode](#) \*code, long \*maxlen, const string essid)
- unsigned long [getCodeForName](#) (const string &name)  
*Get a code value for a given string.*

## Protected Attributes

- long \* [maxlength](#)  
*A reference to the static, persistant maximum length of strings seen so far.*
- [stringcode](#) \* code  
*A reference to the static, persistant code table of strings seen so far.*
- char \* [position](#)  
*Only for internal usage in moveTowards and getDistance.*
- unsigned long [codeval](#)  
*The coded value of the feature.*
- const bool [externalize](#)  
*Externalization flag.*

## Related Functions

(Note that these are not member functions.)

- long [levenshtein](#) (char \*\*x, const char \*t, double \*factor)  
*Modified Levenshtein algorithm.*

### 6.71.1 Detailed Description

This feature encapsulates the ESSID which is currently set for a WLAN network interface or the ESSID which it is currently bound to (if not set explicitly).

It is based on the [AbstractStringFeature](#) and uses the Levenshtein distance.

Definition at line 35 of file Wlan.h.

### 6.71.2 Constructor & Destructor Documentation

#### 6.71.2.1 [WlanActiveEssidFeature::WlanActiveEssidFeature](#) ([stringcode](#) \* code, long \* maxlen) [inline, protected]

This constructor should not be used to construct feature objects.

They are only created by the respective feature provider. Definition at line 44 of file Wlan.h.

References [stringcode](#).

**6.71.2.2 WlanActiveEssidFeature::WlanActiveEssidFeature** ([stringcode](#) \* *code*, long \* *maxlen*, const string *ssid*) [inline, protected]

#### Todo

[documentation](#)

Definition at line 46 of file Wlan.h.

References [stringcode](#).

### 6.71.3 Member Function Documentation

**6.71.3.1 void AbstractStringFeature::aggregate** ([aggregatelist](#) *samples*) [virtual, inherited]

Aggregate a sample values from other sources.

#### Parameters:

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 421 of file AbstractString.cpp.

References [aggregatelist](#).

**6.71.3.2 Feature \* AbstractStringFeature::clone () const** [virtual, inherited]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 403 of file AbstractString.cpp.

References [AbstractStringFeature::AbstractStringFeature\(\)](#), [AbstractStringFeature::code](#), [AbstractStringFeature::codeval](#), [AbstractStringFeature::comparator](#), [AbstractStringFeature::maxlength](#), [AbstractStringFeature::name](#), and [AbstractStringFeature::position](#).

**6.71.3.3 unsigned long AbstractStringFeature::getCodeForName** (const string & *name*) [protected, inherited]

Get a code value for a given string.

A helper function to return the respective code for a given string. If the string is encountered the first time a new code is allocated.

**Parameters:**

*name* The string to code

**Returns:**

A code value

Definition at line 303 of file AbstractString.cpp.

References AbstractStringFeature::code, PersistentFeature::invalidate(), and AbstractStringFeature::maxlength.

Referenced by AbstractStringFeature::AbstractStringFeature(), and AbstractStringListFeature::getCodeForList().

#### 6.71.3.4 double AbstractStringFeature::getDistance (Feature \*f) const [virtual, inherited]

Calculates the distance between two features.

**Parameters:**

*f* Feature used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements Feature.

Reimplemented in AbstractStringListFeature.

Definition at line 337 of file AbstractString.cpp.

References AbstractStringFeature::code, AbstractStringFeature::codeval, AbstractStringFeature::comparator, AbstractStringFeature::levenshtein(), AbstractStringFeature::maxlength, and AbstractStringFeature::position.

#### 6.71.3.5 virtual const string WlanActiveEssidFeature::getName () const [inline, virtual]

Query a features name.

**Returns:**

Name of the Feature in the format "Featureprovider.Feature"

Reimplemented from AbstractStringFeature.

Definition at line 49 of file Wlan.h.

Referenced by toString().

#### 6.71.3.6 double AbstractStringFeature::getPosition () const [virtual, inherited]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 323 of file `AbstractString.cpp`.

References `AbstractStringFeature::maxLength`, and `AbstractStringFeature::position`.

**6.71.3.7** `virtual FeatureType AbstractStringFeature::getType () const` `[inline, virtual, inherited]`

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 100 of file `AbstractString.h`.

**6.71.3.8** `const string& AbstractStringFeature::getVal () const` `[inline, inherited]`

Query the string values.

**Returns:**

Values list

Definition at line 108 of file `AbstractString.h`.

References `AbstractStringFeature::name`.

Referenced by `Java_at_jku_intelligence_samples_StringSample_nativeGetVal()`.

**6.71.3.9** `void PersistentFeature::invalidate ()` `[inline, inherited]`

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file `Feature.h`.

Referenced by `AbstractStringFeature::getCodeForName()`, `NumericalContinuousFeature::NumericalContinuousFeature()`, `NumericalDiscreteFeature::NumericalDiscreteFeature()`, `TimeFeature::TimeFeature()`, `NumericalContinuousFeature::unserialize()`, and `NumericalDiscreteFeature::unserialize()`.

**6.71.3.10** `bool Feature::isExternalizable ()` `[inline, inherited]`

Query externalization flag.

**Returns:**

*true* if the feature is externalizable ( i.e. it has persistent data that should be stored across restarts), *false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

#### 6.71.3.11 bool PersistentFeature::isValid () [inline, inherited]

Query validation flag.

##### Returns:

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

#### 6.71.3.12 void AbstractStringFeature::moveTowards (Feature \*f, double factor) [virtual, inherited]

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample s.

##### Parameters:

*f* [Feature](#) used for distance measurement.

*factor* Distance weight.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 371 of file AbstractString.cpp.

References [AbstractStringFeature::code](#), [AbstractStringFeature::codeval](#), [AbstractStringFeature::comparator](#), [AbstractStringFeature::levenshtein\(\)](#), [AbstractStringFeature::position](#), and [rand\\_double](#).

#### 6.71.3.13 void AbstractStringFeature::read (featureparams \*param) [virtual, inherited]

Load feature from persistent data.

Initializes the persistent feature data from the given representation.

##### Parameters:

*param* Persistent feature data.

##### See also:

[write](#)

Implements [PersistentFeature](#).

Definition at line 296 of file AbstractString.cpp.

References [AbstractStringFeature::code](#), and [featureparams](#).

**6.71.3.14** `string AbstractStringFeature::serialize () const` [virtual, inherited]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 228 of file `AbstractString.cpp`.

References `AbstractStringFeature::codeval`, `AbstractStringFeature::comparator`, and `AbstractStringFeature::position`.

Referenced by `WlanActiveMacAddressFeature::toString()`, `toString()`, `AbstractStringFeature::toString()`, and `toupperstr()`.

**6.71.3.15** `string WlanActiveEssidFeature::toString () const` [virtual]

Get feature as string.

**Note:**

This is only for testing.

**Returns:**

[Feature](#) as string.

Reimplemented from [AbstractStringFeature](#).

Definition at line 29 of file `Wlan.cpp`.

References `getName()`, and `AbstractStringFeature::serialize()`.

**6.71.3.16** `void AbstractStringFeature::unserialize (string value)` [virtual, inherited]

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 251 of file `AbstractString.cpp`.

References `AbstractStringFeature::code`, `AbstractStringFeature::codeval`, `AbstractStringFeature::comparator`, `AbstractStringFeature::maxlength`, and `AbstractStringFeature::position`.

**6.71.3.17** `featureparams AbstractStringFeature::write () const` [virtual, inherited]

Externalize feature.

**Returns:**

Persistent feature data.

**See also:**

[read](#)

Implements [PersistentFeature](#).

Definition at line 283 of file AbstractString.cpp.

References AbstractStringFeature::code, and featureparams.

## 6.71.4 Friends And Related Function Documentation

### 6.71.4.1 long levenshtein (char \*\* *x*, const char \* *t*, double \* *factor*) [related, inherited]

Modified Levenshtein algorithm.

This function implements a slightly modified version of the Levenshtein algorithm to not only calculate the distance of two strings but also modify the string *x* towards the string *t* with the propability *factor*. In other words, every transformation operation found by the algorithm is performed on the string *x* with the propability *factor*.

**Parameters:**

*x* Input string.

*t* String to compare the input string to.

*factor* Propability with witch transformations are performed. Has to be a value out of the intervall  $[0; 1]$ .

**Returns:**

The levenshtein distance between *x* and *t*.

**Todo**

alloc once

Definition at line 46 of file AbstractString.cpp.

References rand\_double.

Referenced by AbstractStringFeature::getDistance(), and AbstractStringFeature::moveTowards().

## 6.71.5 Member Data Documentation

### 6.71.5.1 unsigned long [AbstractStringFeature::codeval](#) [protected, inherited]

The coded value of the feature.

**See also:**

[name](#)

[code](#)

Definition at line 144 of file AbstractString.h.

Referenced by AbstractStringFeature::AbstractStringFeature(), AbstractStringFeature::clone(), AbstractStringFeature::getCodeVal(), AbstractStringFeature::getDistance(), AbstractStringFeature::moveTowards(), AbstractStringFeature::serialize(), and AbstractStringFeature::unserialize().



**6.71.5.2** `const bool Feature::externalize` [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to `PersistentFeature`, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**

[PersistentFeature](#)

Definition at line 187 of file `Feature.h`.

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

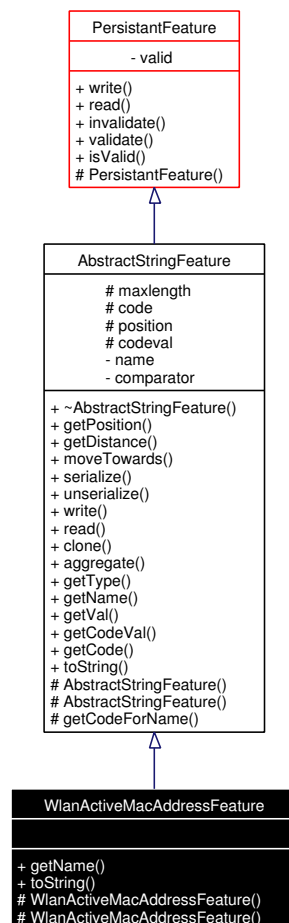
- [Wlan.h](#)
- [Wlan.cpp](#)

## 6.72 WlanActiveMacAddressFeature Class Reference

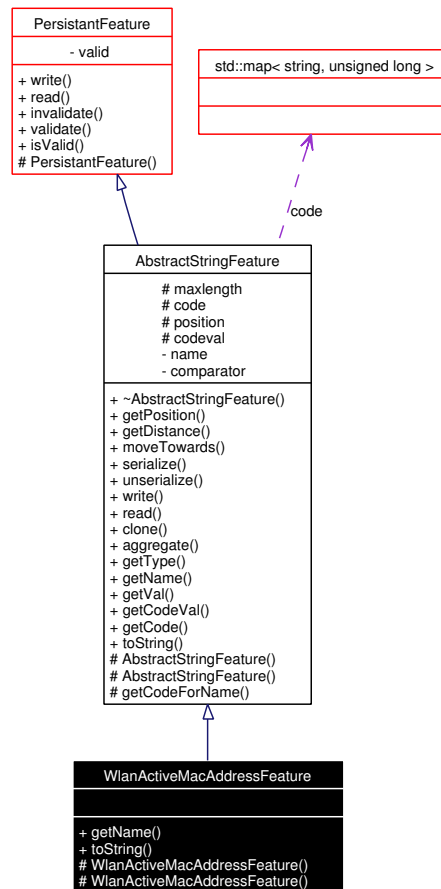
This feature encapsulates the ESSID which is currently set for a WLAN network interface or the ESSID which it is currently bound to (if not set explicitly).

```
#include <Wlan.h>
```

Inheritance diagram for WlanActiveMacAddressFeature:



Collaboration diagram for WlanActiveMacAddressFeature:



## Public Types

- enum [ComparatorType](#) { **None**, **Levenshtein** }  
*The distance metric to use.*
- enum [FeatureType](#) {  
**boolean**, **nominal**, **ordinal**, **numerical\_discrete**,  
**numerical\_continuous** }  
*Possible feature types.*

## Public Member Functions

- virtual const string [getName](#) () const  
*Query a features name.*
- virtual string [toString](#) () const  
*This is only for testing.*
- virtual double [getPosition](#) () const

*Query a features position.*

- virtual double `getDistance` (`Feature *f`) const  
*Calculates the distance between two features.*
- virtual void `moveTowards` (`Feature *f`, double factor)  
*Move feature.*
- virtual string `serialize` () const  
*Serialize a samples data to a string.*
- virtual void `unserialize` (string value)  
*Unserialize a samples data from a string.*
- virtual `featureparams write` () const  
*Externalize feature.*
- virtual void `read` (`featureparams *param`)  
*Load feature from persistant data.*
- virtual `Feature * clone` () const  
*Clone a feature.*
- virtual void `aggregate` (`aggregatelist` samples)  
*Aggregate a sample values from other sources.*
- virtual `FeatureType getType` () const  
*Query a features type.*
- const string & `getVal` () const  
*Query the string values.*
- const unsigned long `getCodeVal` () const
- const `stringcode * getCode` ()
- void `invalidate` ()  
*Invalidate feature.*
- void `validate` ()  
*Set the validation flag to true.*
- bool `isValid` ()  
*Query validation flag.*
- bool `isExternalizable` ()  
*Query externalization flag.*

## Protected Member Functions

- [WlanActiveMacAddressFeature](#) ([stringcode](#) \*code, long \*maxlen)  
*This constructor should not be used to construct feature objects.*
- [WlanActiveMacAddressFeature](#) ([stringcode](#) \*code, long \*maxlen, const string mac)  
*This constructor should not be used to construct feature objects.*
- unsigned long [getCodeForName](#) (const string &name)  
*Get a code value for a given string.*

## Protected Attributes

- long \* [maxlength](#)  
*A reference to the static, persistent maximum length of strings seen so far.*
- [stringcode](#) \* code  
*A reference to the static, persistent code table of strings seen so far.*
- char \* [position](#)  
*Only for internal usage in moveTowards and getDistance.*
- unsigned long [codeval](#)  
*The coded value of the feature.*
- const bool [externalize](#)  
*Externalization flag.*

## Related Functions

(Note that these are not member functions.)

- long [levenshtein](#) (char \*\*x, const char \*t, double \*factor)  
*Modified Levenshtein algorithm.*

### 6.72.1 Detailed Description

This feature encapsulates the ESSID which is currently set for a WLAN network interface or the ESSID which it is currently bound to (if not set explicitly).

It is based on the [AbstractStringFeature](#) and uses the Levenshtein distance.

Definition at line 61 of file Wlan.h.

## 6.72.2 Constructor & Destructor Documentation

### 6.72.2.1 `WlanActiveMacAddressFeature::WlanActiveMacAddressFeature (stringcode * code, long * maxlen) [inline, protected]`

This constructor should not be used to construct feature objects.

They are only created by the respective feature provider. Definition at line 70 of file Wlan.h.

References [stringcode](#).

### 6.72.2.2 `WlanActiveMacAddressFeature::WlanActiveMacAddressFeature (stringcode * code, long * maxlen, const string mac) [inline, protected]`

This constructor should not be used to construct feature objects.

They are only created by the respective feature provider. Definition at line 76 of file Wlan.h.

References [stringcode](#).

## 6.72.3 Member Function Documentation

### 6.72.3.1 `void AbstractStringFeature::aggregate (aggregatelist samples) [virtual, inherited]`

Aggregate a sample values from other sources.

#### Parameters:

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 421 of file AbstractString.cpp.

References [aggregatelist](#).

### 6.72.3.2 `Feature * AbstractStringFeature::clone () const [virtual, inherited]`

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 403 of file AbstractString.cpp.

References [AbstractStringFeature::AbstractStringFeature\(\)](#), [AbstractStringFeature::code](#), [AbstractStringFeature::codeval](#), [AbstractStringFeature::comparator](#), [AbstractStringFeature::maxlength](#), [AbstractStringFeature::name](#), and [AbstractStringFeature::position](#).

### 6.72.3.3 `unsigned long AbstractStringFeature::getCodeForName (const string & name) [protected, inherited]`

Get a code value for a given string.

A helper function to return the respective code for a given string. If the string is encountered the first time a new code is allocated.

**Parameters:**

*name* The string to code

**Returns:**

A code value

Definition at line 303 of file AbstractString.cpp.

References AbstractStringFeature::code, PersistentFeature::invalidate(), and AbstractStringFeature::maxlength.

Referenced by AbstractStringFeature::AbstractStringFeature(), and AbstractStringListFeature::getCodeForList().

#### 6.72.3.4 double AbstractStringFeature::getDistance ([Feature](#) \**f*) const [virtual, inherited]

Calculates the distance between two features.

**Parameters:**

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 337 of file AbstractString.cpp.

References AbstractStringFeature::code, AbstractStringFeature::codeval, AbstractStringFeature::comparator, AbstractStringFeature::levenshtein(), AbstractStringFeature::maxlength, and AbstractStringFeature::position.

#### 6.72.3.5 virtual const string WlanActiveMacAddressFeature::getName () const [inline, virtual]

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Reimplemented from [AbstractStringFeature](#).

Definition at line 79 of file Wlan.h.

Referenced by toString().

**6.72.3.6 double AbstractStringFeature::getPosition () const** [virtual, inherited]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 323 of file AbstractString.cpp.

References AbstractStringFeature::maxlength, and AbstractStringFeature::position.

**6.72.3.7 virtual FeatureType AbstractStringFeature::getType () const** [inline, virtual, inherited]

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 100 of file AbstractString.h.

**6.72.3.8 const string& AbstractStringFeature::getVal () const** [inline, inherited]

Query the string values.

**Returns:**

Values list

Definition at line 108 of file AbstractString.h.

References AbstractStringFeature::name.

Referenced by Java\_at\_jku\_intelligence\_samples\_StringSample\_nativeGetVal().

**6.72.3.9 void PersistentFeature::invalidate ()** [inline, inherited]

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by AbstractStringFeature::getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature::NumericalDiscreteFeature(), TimeFeature::TimeFeature(), NumericalContinuousFeature::unserialize(), and NumericalDiscreteFeature::unserialize().



**6.72.3.10 bool Feature::isExternalizable ()** [inline, inherited]

Query externalization flag.

**Returns:**

*true* if the feature is externalizeable ( i.e. it has persistend data that should be stored across restarts),  
*false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.72.3.11 bool PersistentFeature::isValid ()** [inline, inherited]

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

**6.72.3.12 void AbstractStringFeature::moveTowards (Feature \**f*, double *factor*)** [virtual, inherited]

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample s.

**Parameters:**

*f* [Feature](#) used for distance measurement.

*factor* Distance weight.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 371 of file AbstractString.cpp.

References [AbstractStringFeature::code](#), [AbstractStringFeature::codeval](#), [AbstractStringFeature::comparator](#), [AbstractStringFeature::levenshtein\(\)](#), [AbstractStringFeature::position](#), and [rand\\_double](#).

**6.72.3.13 void AbstractStringFeature::read (featureparams \**param*)** [virtual, inherited]

Load feature from persistent data.

Initializes the persistent feature data from the given representation.

**Parameters:**

*param* Persistent feature data.

**See also:**[write](#)

Implements [PersistantFeature](#).

Definition at line 296 of file AbstractString.cpp.

References AbstractStringFeature::code, and featureparams.

**6.72.3.14 string AbstractStringFeature::serialize () const** [virtual, inherited]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 228 of file AbstractString.cpp.

References AbstractStringFeature::codeval, AbstractStringFeature::comparator, and AbstractStringFeature::position.

Referenced by toString(), WlanActiveEssidFeature::toString(), AbstractStringFeature::toString(), and toupperstr().

**6.72.3.15 void AbstractStringFeature::unserialize (string value)** [virtual, inherited]

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Reimplemented in [AbstractStringListFeature](#).

Definition at line 251 of file AbstractString.cpp.

References AbstractStringFeature::code, AbstractStringFeature::codeval, AbstractStringFeature::comparator, AbstractStringFeature::maxlength, and AbstractStringFeature::position.

**6.72.3.16 featureparams AbstractStringFeature::write () const** [virtual, inherited]

Externalize feature.

**Returns:**

Persistant feature data.

**See also:**[read](#)

Implements [PersistantFeature](#).

Definition at line 283 of file AbstractString.cpp.

References AbstractStringFeature::code, and featureparams.

## 6.72.4 Friends And Related Function Documentation

### 6.72.4.1 `long levenshtein (char ** x, const char * t, double * factor)` [related, inherited]

Modified Levenshtein algorithm.

This function implements a slightly modified version of the Levenshtein algorithm to not only calculate the distance of two strings but also modify the string *x* towards the string *t* with the propability *factor*. In other words, every transformation operation found by the algorithm is performed on the string *x* with the propability *factor*.

#### Parameters:

*x* Input string.

*t* String to compare the input string to.

*factor* Propability with witch transformations are performed. Has to be a value out of the intervall  $[0; 1]$ .

#### Returns:

The levenshtein distance between *x* and *t*.

#### Todo

alloc once

Definition at line 46 of file AbstractString.cpp.

References `rand_double`.

Referenced by `AbstractStringFeature::getDistance()`, and `AbstractStringFeature::moveTowards()`.

## 6.72.5 Member Data Documentation

### 6.72.5.1 `unsigned long AbstractStringFeature::codeval` [protected, inherited]

The coded value of the feature.

#### See also:

[name](#)

[code](#)

Definition at line 144 of file AbstractString.h.

Referenced by `AbstractStringFeature::AbstractStringFeature()`, `AbstractStringFeature::clone()`, `AbstractStringFeature::getCodeVal()`, `AbstractStringFeature::getDistance()`, `AbstractStringFeature::moveTowards()`, `AbstractStringFeature::serialize()`, and `AbstractStringFeature::unserialize()`.

### 6.72.5.2 `const bool Feature::externalize` [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistant features, the object should be cast to `PersistentFeature`, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**[PersistantFeature](#)

Definition at line 187 of file Feature.h.

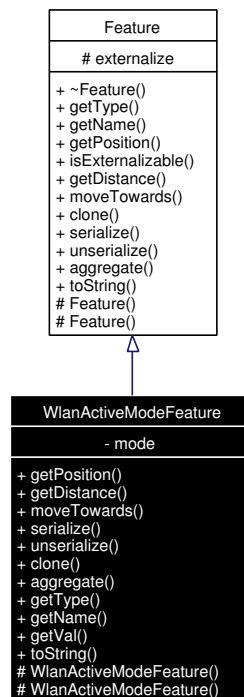
The documentation for this class was generated from the following files:

- [Wlan.h](#)
- [Wlan.cpp](#)

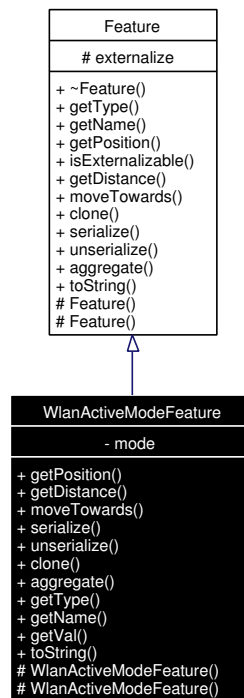
## 6.73 WlanActiveModeFeature Class Reference

```
#include <Wlan.h>
```

Inheritance diagram for WlanActiveModeFeature:



Collaboration diagram for WlanActiveModeFeature:



## Public Types

- enum [FeatureKind](#) {  
**Unknown, Master, Managed, AdHoc,**  
**Last** }
  - enum [FeatureType](#) {  
**boolean, nominal, ordinal, numerical\_discrete,**  
**numerical\_continuous** }
- Possible feature types.*

## Public Member Functions

- virtual double [getPosition](#) () const  
*Query a features position.*
- virtual double [getDistance](#) ([Feature](#) \*f) const  
*Calculates the distance between two features.*
- virtual void [moveTowards](#) ([Feature](#) \*f, double factor)  
*Move feature.*
- virtual string [serialize](#) () const  
*Serialize a samples data to a string.*

- virtual void [unserialize](#) (string value)  
*Unserialize a samples data from a string.*
- virtual [Feature](#) \* [clone](#) () const  
*Clone a feature.*
- void [aggregate](#) ([aggregatelist](#) samples)  
*Aggregate a sample values from other sources.*
- virtual [FeatureType](#) [getType](#) () const  
*Query a features type.*
- virtual const string [getName](#) () const  
*Query a features name.*
- const [FeatureKind](#) [getVal](#) () const
- virtual string [toString](#) () const  
*This is only for testing.*
- bool [isExternalizable](#) ()  
*Query externalization flag.*

## Protected Member Functions

- [WlanActiveModeFeature](#) ()  
*This constructor should not be used to construct feature objects.*
- [WlanActiveModeFeature](#) ([FeatureKind](#) mode)  
*This constructor should not be used to construct feature objects.*

## Protected Attributes

- const bool [externalize](#)  
*Externalization flag.*

## Private Attributes

- [FeatureKind](#) mode

### 6.73.1 Detailed Description

#### [Todo](#)

documentation

Definition at line 90 of file Wlan.h.

## 6.73.2 Member Enumeration Documentation

### 6.73.2.1 enum [WlanActiveModeFeature::FeatureKind](#)

#### [Todo](#)

documentation

Definition at line 96 of file Wlan.h.

Referenced by `unserialize()`, and `WlanActiveModeFeature()`.

## 6.73.3 Constructor & Destructor Documentation

### 6.73.3.1 `WlanActiveModeFeature::WlanActiveModeFeature ()` `[protected]`

This constructor should not be used to construct feature objects.

They are only created by the respective feature provider. Definition at line 59 of file Wlan.cpp.

References `FeatureKind`, `mode`, and `rand_double`.

Referenced by `clone()`.

### 6.73.3.2 `WlanActiveModeFeature::WlanActiveModeFeature (FeatureKind mode)` `[protected]`

This constructor should not be used to construct feature objects.

They are only created by the respective feature provider. Definition at line 64 of file Wlan.cpp.

## 6.73.4 Member Function Documentation

### 6.73.4.1 `void WlanActiveModeFeature::aggregate (aggregatelist samples)` `[virtual]`

Aggregate a sample values from other sources.

#### Parameters:

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 135 of file Wlan.cpp.

References `aggregatelist`.

### 6.73.4.2 `Feature * WlanActiveModeFeature::clone () const` `[virtual]`

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Definition at line 130 of file Wlan.cpp.

References `mode`, and `WlanActiveModeFeature()`.



**6.73.4.3 double WlanActiveModeFeature::getDistance (Feature \*f) const** [virtual]

Calculates the distance between two features.

**Parameters:**

*f* Feature used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements Feature.

Definition at line 76 of file Wlan.cpp.

References mode.

**6.73.4.4 virtual const string WlanActiveModeFeature::getName () const** [inline, virtual]

Query a features name.

**Returns:**

Name of the Feature in the format "Featureprovider.Feature"

Implements Feature.

Definition at line 123 of file Wlan.h.

**6.73.4.5 double WlanActiveModeFeature::getPosition () const** [virtual]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements Feature.

Definition at line 84 of file Wlan.cpp.

References mode.

**6.73.4.6 virtual FeatureType WlanActiveModeFeature::getType () const** [inline, virtual]

Query a features type.

**Returns:**

The type of the Feature.

Implements Feature.

Definition at line 122 of file Wlan.h.

**6.73.4.7 bool Feature::isExternalizable () [inline, inherited]**

Query externalization flag.

**Returns:**

*true* if the feature is externalizeable ( i.e. it has persistend data that should be stored across restarts),  
*false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.73.4.8 void WlanActiveModeFeature::moveTowards (Feature \**f*, double *factor*) [virtual]**

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample s.

**Parameters:**

*f* Feature used for distance measurement.

*factor* Distance weight.

Implements Feature.

Definition at line 69 of file Wlan.cpp.

References mode, and rand\_double.

**6.73.4.9 string WlanActiveModeFeature::serialize () const [virtual]**

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements Feature.

Definition at line 89 of file Wlan.cpp.

References mode.

**6.73.4.10 void WlanActiveModeFeature::unserialize (string *value*) [virtual]**

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements Feature.

Definition at line 97 of file Wlan.cpp.

References FeatureKind, and mode.

## 6.73.5 Member Data Documentation

### 6.73.5.1 `const bool Feature::externalize` [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to `PersistentFeature`, because only subclasses of this type are (by policy) allowed to set this variable to true.

See also:

[PersistentFeature](#)

Definition at line 187 of file `Feature.h`.

### 6.73.5.2 `FeatureKind WlanActiveModeFeature::mode` [private]

**Todo**

documentation

Definition at line 134 of file `Wlan.h`.

Referenced by `clone()`, `getDistance()`, `getPosition()`, `moveTowards()`, `serialize()`, `toString()`, `unserialize()`, and `WlanActiveModeFeature()`.

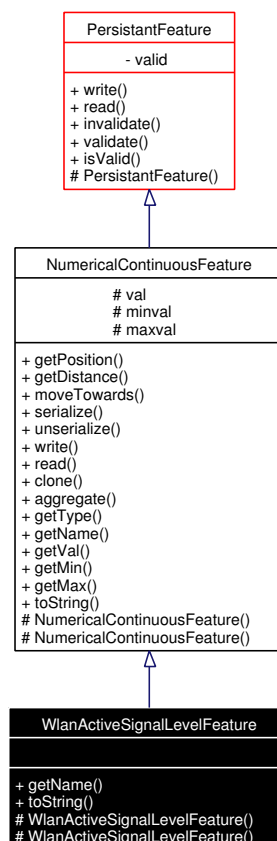
The documentation for this class was generated from the following files:

- [Wlan.h](#)
- [Wlan.cpp](#)

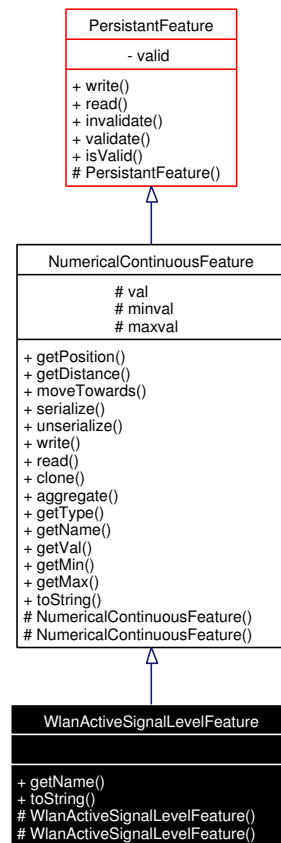
## 6.74 WlanActiveSignalLevelFeature Class Reference

```
#include <Wlan.h>
```

Inheritance diagram for WlanActiveSignalLevelFeature:



Collaboration diagram for WlanActiveSignalLevelFeature:



## Public Types

- enum `FeatureType` {  
     **boolean**, **nominal**, **ordinal**, **numerical\_discrete**,  
     **numerical\_continuous** }  
*Possible feature types.*

## Public Member Functions

- virtual const string `getName ()` const  
*Query a features name.*
- virtual string `toString ()` const  
*This is only for testing.*
- virtual double `getPosition ()` const  
*Query a features position.*
- virtual double `getDistance (Feature *f)` const  
*Calculates the distance between two features.*

- virtual void [moveTowards](#) ([Feature](#) \*f, double factor)

*Move feature.*

- virtual string [serialize](#) () const

*Serialize a samples data to a string.*

- virtual void [unserialize](#) (string value)

*Unserialize a samples data from a string.*

- virtual [featureparams](#) [write](#) () const

*Externalize feature.*

- virtual void [read](#) ([featureparams](#) \*param)

*Load feature from persistant data.*

- virtual [Feature](#) \* [clone](#) () const

*Clone a feature.*

- virtual void [aggregate](#) ([aggregatelist](#) samples)

*Aggregate a sample values from other sources.*

- virtual [FeatureType](#) [getType](#) () const

*Query a features type.*

- const double [getVal](#) () const

- const double [getMin](#) () const

- const double [getMax](#) () const

- void [invalidate](#) ()

*Invalidate feature.*

- void [validate](#) ()

*Set the validation flag to true.*

- bool [isValid](#) ()

*Query validation flag.*

- bool [isExternalizable](#) ()

*Query externalization flag.*

## Protected Member Functions

- [WlanActiveSignalLevelFeature](#) (double \*minval, double \*maxval)

*This constructor should not be used to construct feature objects.*

- [WlanActiveSignalLevelFeature](#) (double \*minval, double \*maxval, const double level)

*This constructor should not be used to construct feature objects.*

## Protected Attributes

- double [val](#)  
*The feature value.*
- double \* [minval](#)  
*A reference to the static, persistant minimum value seen so far.*
- double \* [maxval](#)  
*A reference to the static, persistant maximum value seen so far.*
- const bool [externalize](#)  
*Externalization flag.*

### 6.74.1 Detailed Description

#### [Todo](#)

documentation

Definition at line 140 of file Wlan.h.

### 6.74.2 Constructor & Destructor Documentation

#### 6.74.2.1 WlanActiveSignalLevelFeature::WlanActiveSignalLevelFeature (double \* [minval](#), double \* [maxval](#)) [[inline](#), [protected](#)]

This constructor should not be used to construct feature objects.

They are only created by the respective feature provider. Definition at line 149 of file Wlan.h.

#### 6.74.2.2 WlanActiveSignalLevelFeature::WlanActiveSignalLevelFeature (double \* [minval](#), double \* [maxval](#), const double [level](#)) [[inline](#), [protected](#)]

This constructor should not be used to construct feature objects.

They are only created by the respective feature provider. Definition at line 155 of file Wlan.h.

### 6.74.3 Member Function Documentation

#### 6.74.3.1 void NumericalContinuousFeature::aggregate ([aggregatelist](#) [samples](#)) [[virtual](#), [inherited](#)]

Aggregate a sample values from other sources.

##### Parameters:

*[samples](#)* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 309 of file Numerical.cpp.

References [aggregatelist](#).

**6.74.3.2** [Feature](#) \* **NumericalContinuousFeature::clone () const** [virtual, inherited]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Definition at line 304 of file Numerical.cpp.

References [NumericalContinuousFeature::maxval](#), [NumericalContinuousFeature::minval](#), [NumericalContinuousFeature::NumericalContinuousFeature\(\)](#), and [NumericalContinuousFeature::val](#).

**6.74.3.3** **double NumericalContinuousFeature::getDistance ([Feature](#) \**f*) const** [virtual, inherited]

Calculates the distance between two features.

**Parameters:**

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Definition at line 220 of file Numerical.cpp.

References [NumericalContinuousFeature::getPosition\(\)](#), [NumericalContinuousFeature::maxval](#), and [NumericalContinuousFeature::minval](#).

**6.74.3.4** **virtual const string WlanActiveSignalLevelFeature::getName () const** [inline, virtual]

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Reimplemented from [NumericalContinuousFeature](#).

Definition at line 158 of file Wlan.h.

**6.74.3.5** **double NumericalContinuousFeature::getPosition () const** [virtual, inherited]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.



Implements [Feature](#).

Definition at line 204 of file Numerical.cpp.

References `NumericalContinuousFeature::maxval`, `NumericalContinuousFeature::minval`, and `NumericalContinuousFeature::val`.

Referenced by `NumericalContinuousFeature::getDistance()`.

**6.74.3.6** `virtual FeatureType NumericalContinuousFeature::getType () const` [`inline`, `virtual`, `inherited`]

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 183 of file Numerical.h.

**6.74.3.7** `const double NumericalContinuousFeature::getVal () const` [`inline`, `inherited`]

**Returns:**

The value of the feature.

Definition at line 187 of file Numerical.h.

Referenced by `getProvider()`, `Java_at_jku_intelligence_samples_NumericalContinuousSample_nativeGetVal()`, `toString()`, and `NumericalContinuousFeature::toString()`.

**6.74.3.8** `void PersistentFeature::invalidate ()` [`inline`, `inherited`]

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by `AbstractStringFeature::getCodeForName()`, `NumericalContinuousFeature::NumericalContinuousFeature()`, `NumericalDiscreteFeature::NumericalDiscreteFeature()`, `TimeFeature::TimeFeature()`, `NumericalContinuousFeature::unserialize()`, and `NumericalDiscreteFeature::unserialize()`.

**6.74.3.9** `bool Feature::isExternalizable ()` [`inline`, `inherited`]

Query externalization flag.

**Returns:**

*true* if the feature is externalizable ( i.e. it has persistent data that should be stored across restarts), *false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by `FeatureContainer::FeatureContainer()`, and `FeatureContainer::nextSample()`.

**6.74.3.10 bool PersistentFeature::isValid ()** [inline, inherited]

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

**6.74.3.11 void NumericalContinuousFeature::moveTowards (Feature \* *f*, double *factor*)**  
[virtual, inherited]

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

**Parameters:**

*f* Feature used for distance measurement.

*factor* Distance weight.

Implements Feature.

Definition at line 227 of file Numerical.cpp.

References NumericalContinuousFeature::maxval, NumericalContinuousFeature::minval, and NumericalContinuousFeature::val.

**6.74.3.12 void NumericalContinuousFeature::read (featureparams \* *param*)** [virtual, inherited]

Load feature from persistent data.

Initializes the persistent feature data from the given representation.

**Parameters:**

*param* Persistent feature data.

**See also:**

[write](#)

Implements PersistentFeature.

Definition at line 291 of file Numerical.cpp.

References featureparams, NumericalContinuousFeature::maxval, and NumericalContinuousFeature::minval.

**6.74.3.13 string NumericalContinuousFeature::serialize () const** [virtual, inherited]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [Feature](#).

Definition at line 246 of file Numerical.cpp.

References NumericalContinuousFeature::val.

**6.74.3.14 void NumericalContinuousFeature::unserialize (string *value*)** [virtual, inherited]

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Definition at line 254 of file Numerical.cpp.

References PersistentFeature::invalidate(), NumericalContinuousFeature::maxval, NumericalContinuousFeature::minval, and NumericalContinuousFeature::val.

**6.74.3.15 [featureparams](#) NumericalContinuousFeature::write () const** [virtual, inherited]

Externalize feature.

**Returns:**

Persistent feature data.

**See also:**

[read](#)

Implements [PersistentFeature](#).

Definition at line 277 of file Numerical.cpp.

References [featureparams](#), NumericalContinuousFeature::maxval, and NumericalContinuousFeature::minval.

## 6.74.4 Member Data Documentation

**6.74.4.1 const bool [Feature::externalize](#)** [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to PersistentFeature, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**[PersistantFeature](#)

Definition at line 187 of file Feature.h.

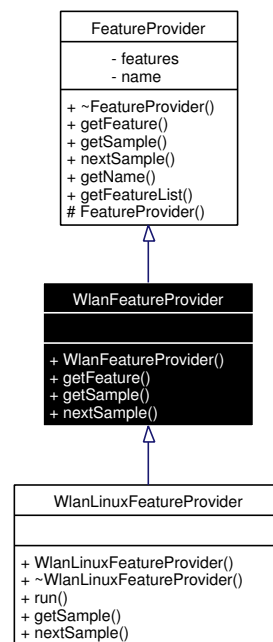
The documentation for this class was generated from the following files:

- [Wlan.h](#)
- [Wlan.cpp](#)

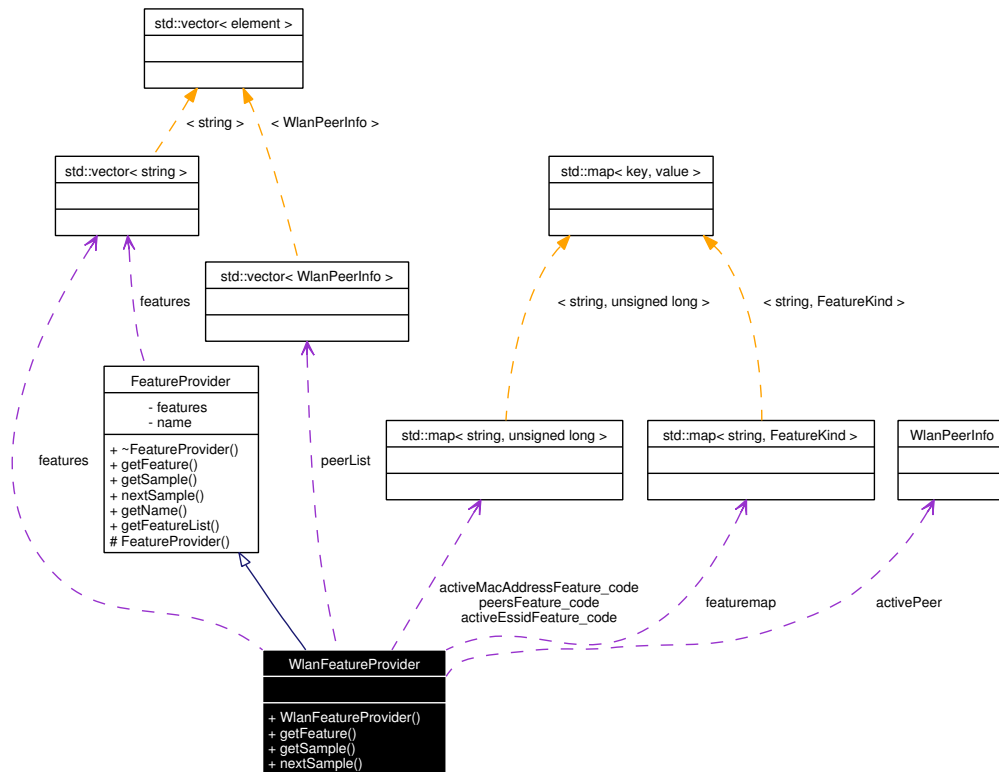
## 6.75 WlanFeatureProvider Class Reference

```
#include <Wlan.h>
```

Inheritance diagram for WlanFeatureProvider:



Collaboration diagram for WlanFeatureProvider:



## Public Types

- enum `FeatureKind` {  
`ActiveEssid`, `ActiveMode`, `ActiveSignalLevel`, `ActiveMacAddress`,  
`Peers`, `NumPeers` }

## Public Member Functions

- `WlanFeatureProvider` (`providerparams` &`params`)
- virtual `Feature *` `getFeature` (`string name`) const  
*Query feature instance.*
- virtual `Feature *` `getSample` (`string name`) const  
*Query sample instance.*
- virtual void `nextSample` (`clock_t` checkpoint)
- string `getName` () const  
*Query provider name.*
- `stringvector *` `getFeatureList` () const  
*Query list of provided features.*

## Protected Attributes

- [WlanPeerInfo](#) activePeer
- [vector< WlanPeerInfo >](#) peerList
- [bool](#) activePeerValid
- [bool](#) peerListValid

## Private Attributes

- [stringvector](#) features
- [map< string, FeatureKind >](#) featuremap
- [stringcode](#) activeEssidFeature\_code
- [long](#) activeEssidFeature\_maxlen
- [stringcode](#) activeMacAddressFeature\_code
- [long](#) activeMacAddressFeature\_maxlen
- [double](#) activeSignalLevelFeature\_minval
- [double](#) activeSignalLevelFeature\_maxval
- [stringcode](#) peersFeature\_code
- [long](#) peersFeature\_maxlen
- [long](#) numPeersFeature\_minval
- [long](#) numPeersFeature\_maxval

### 6.75.1 Detailed Description

#### [Todo](#)

documentation

Definition at line 241 of file Wlan.h.

### 6.75.2 Member Enumeration Documentation

#### 6.75.2.1 [enum WlanFeatureProvider::FeatureKind](#)

#### [Todo](#)

documentation

Definition at line 245 of file Wlan.h.

### 6.75.3 Constructor & Destructor Documentation

#### 6.75.3.1 [WlanFeatureProvider::WlanFeatureProvider](#) ([providerparams](#) & *params*)

#### [Todo](#)

documentation

Definition at line 171 of file Wlan.cpp.

References [activeEssidFeature\\_maxlen](#), [activeMacAddressFeature\\_maxlen](#), [activePeerValid](#), [activeSignalLevelFeature\\_maxval](#), [activeSignalLevelFeature\\_minval](#), [featuremap](#), [features](#), [numPeersFeature\\_maxval](#), [numPeersFeature\\_minval](#), [peerListValid](#), [peersFeature\\_maxlen](#), and [providerparams](#).

## 6.75.4 Member Function Documentation

### 6.75.4.1 **Feature** \* WlanFeatureProvider::getFeature (string *name*) const [virtual]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the requested feature.

Implements [FeatureProvider](#).

Definition at line 260 of file Wlan.cpp.

References activeEssidFeature\_code, activeEssidFeature\_maxlen, activeMacAddressFeature\_code, activeMacAddressFeature\_maxlen, activeSignalLevelFeature\_maxval, activeSignalLevelFeature\_minval, featuremap, numPeersFeature\_maxval, numPeersFeature\_minval, peersFeature\_code, and peersFeature\_maxlen.

### 6.75.4.2 **stringvector**\* FeatureProvider::getFeatureList () const [inline, inherited]

Query list of provided features.

**Returns:**

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by FeatureContainer::loadFeature().

### 6.75.4.3 **string** FeatureProvider::getName () const [inline, inherited]

Query provider name.

**Returns:**

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

### 6.75.4.4 **Feature** \* WlanFeatureProvider::getSample (string *name*) const [virtual]

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

**Parameters:**

*name* Name of the requested feature.



**Returns:**

An instance of the sample.

**See also:**

[nextSample](#)

Implements [FeatureProvider](#).

Reimplemented in [WlanLinuxFeatureProvider](#).

Definition at line 209 of file Wlan.cpp.

References [activeEssidFeature\\_code](#), [activeEssidFeature\\_maxlen](#), [activeMacAddressFeature\\_code](#), [activeMacAddressFeature\\_maxlen](#), [activePeer](#), [activePeerValid](#), [activeSignalLevelFeature\\_maxval](#), [activeSignalLevelFeature\\_minval](#), [WlanPeerInfo::essid](#), [featuremap](#), [WlanPeerInfo::mac](#), [WlanPeerInfo::mode](#), [numPeersFeature\\_maxval](#), [numPeersFeature\\_minval](#), [peerList](#), [peerListValid](#), [peersFeature\\_code](#), [peersFeature\\_maxlen](#), and [WlanPeerInfo::signalLevel](#).

**6.75.4.5 void WlanFeatureProvider::nextSample (clock\_t *checkpoint*)** [virtual]**Todo**

clean up

Implements [FeatureProvider](#).

Reimplemented in [WlanLinuxFeatureProvider](#).

Definition at line 56 of file WlanLinux.cpp.

**6.75.5 Member Data Documentation****6.75.5.1 stringcode WlanFeatureProvider::activeEssidFeature\_code** [private]**Todo**

documentation

Definition at line 261 of file Wlan.h.

Referenced by [getFeature\(\)](#), and [getSample\(\)](#).

**6.75.5.2 long WlanFeatureProvider::activeEssidFeature\_maxlen** [private]**Todo**

documentation

Definition at line 262 of file Wlan.h.

Referenced by [getFeature\(\)](#), [getSample\(\)](#), and [WlanFeatureProvider\(\)](#).

**6.75.5.3 stringcode WlanFeatureProvider::activeMacAddressFeature\_code** [private]**Todo**

documentation

Definition at line 263 of file Wlan.h.

Referenced by [getFeature\(\)](#), and [getSample\(\)](#).

**6.75.5.4** long **WlanFeatureProvider::activeMacAddressFeature\_maxlen** [private]**Todo**

documentation

Definition at line 264 of file Wlan.h.

Referenced by getFeature(), getSample(), and WlanFeatureProvider().

**6.75.5.5** **WlanPeerInfo** **WlanFeatureProvider::activePeer** [protected]**Todo**

documentation

Definition at line 277 of file Wlan.h.

Referenced by getSample().

**6.75.5.6** bool **WlanFeatureProvider::activePeerValid** [protected]**Todo**

documentation

Definition at line 279 of file Wlan.h.

Referenced by getSample(), and WlanFeatureProvider().

**6.75.5.7** double **WlanFeatureProvider::activeSignalLevelFeature\_maxval** [private]**Todo**

documentation

Definition at line 266 of file Wlan.h.

Referenced by getFeature(), getSample(), and WlanFeatureProvider().

**6.75.5.8** double **WlanFeatureProvider::activeSignalLevelFeature\_minval** [private]**Todo**

documentation

Definition at line 265 of file Wlan.h.

Referenced by getFeature(), getSample(), and WlanFeatureProvider().

**6.75.5.9** **map**<string, **FeatureKind**> **WlanFeatureProvider::featuremap** [mutable, private]**Todo**

documentation

Definition at line 259 of file Wlan.h.

Referenced by getFeature(), getSample(), and WlanFeatureProvider().

**6.75.5.10** [stringvector WlanFeatureProvider::features](#) [private]**Todo**

documentation

Reimplemented from [FeatureProvider](#).

Definition at line 257 of file Wlan.h.

Referenced by WlanFeatureProvider().

**6.75.5.11** [long WlanFeatureProvider::numPeersFeature\\_maxval](#) [private]**Todo**

documentation

Definition at line 270 of file Wlan.h.

Referenced by getFeature(), getSample(), and WlanFeatureProvider().

**6.75.5.12** [long WlanFeatureProvider::numPeersFeature\\_minval](#) [private]**Todo**

documentation

Definition at line 269 of file Wlan.h.

Referenced by getFeature(), getSample(), and WlanFeatureProvider().

**6.75.5.13** [vector<WlanPeerInfo> WlanFeatureProvider::peerList](#) [protected]**Todo**

documentation

Definition at line 278 of file Wlan.h.

Referenced by getSample().

**6.75.5.14** [bool WlanFeatureProvider::peerListValid](#) [protected]**Todo**

documentation

Definition at line 279 of file Wlan.h.

Referenced by getSample(), and WlanFeatureProvider().

**6.75.5.15** [stringcode WlanFeatureProvider::peersFeature\\_code](#) [private]**Todo**

documentation

Definition at line 267 of file Wlan.h.

Referenced by getFeature(), and getSample().

**6.75.5.16** long [WlanFeatureProvider::peersFeature\\_maxlen](#) [private]

**Todo**

documentation

Definition at line 268 of file Wlan.h.

Referenced by [getFeature\(\)](#), [getSample\(\)](#), and [WlanFeatureProvider\(\)](#).

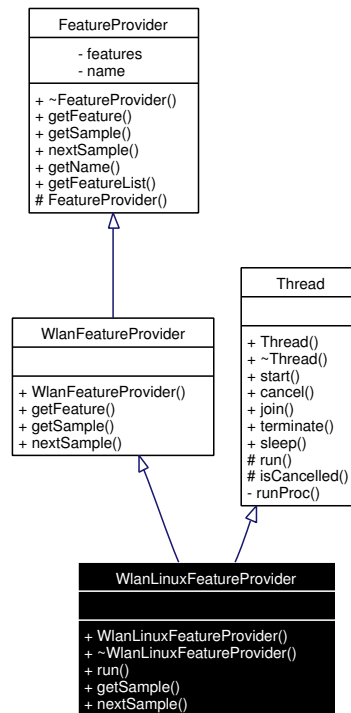
The documentation for this class was generated from the following files:

- [Wlan.h](#)
- [Wlan.cpp](#)
- [WlanLinux.cpp](#)
- [WlanSymbolAPI.cpp](#)
- [WlanWindows.cpp](#)

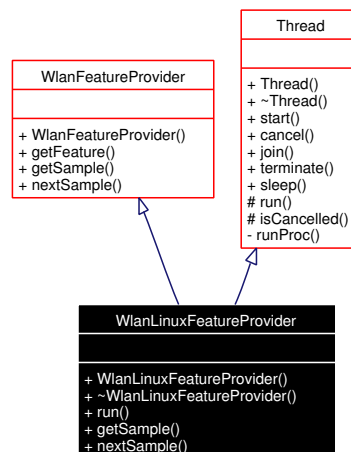
## 6.76 WlanLinuxFeatureProvider Class Reference

```
#include <WlanLinux.h>
```

Inheritance diagram for WlanLinuxFeatureProvider:



Collaboration diagram for WlanLinuxFeatureProvider:



### Public Types

- enum [FeatureKind](#) {

ActiveEssid, ActiveMode, ActiveSignalLevel, ActiveMacAddress,  
Peers, NumPeers }

## Public Member Functions

- [WlanLinuxFeatureProvider](#) (featureparams &params)
- virtual [~WlanLinuxFeatureProvider](#) () throw ()

*Destructor.*

- virtual void [run](#) () throw ()

*Thread main worker function.*

- virtual [Feature](#) \* [getSample](#) (string [name](#)) const

*Query sample instance.*

- virtual void [nextSample](#) (clock\_t checkpoint)

- virtual [Feature](#) \* [getFeature](#) (string [name](#)) const

*Query feature instance.*

- string [getName](#) () const

*Query provider name.*

- [stringvector](#) \* [getFeatureList](#) () const

*Query list of provided features.*

- void [start](#) ()

*Start thread execution.*

- void [cancel](#) ()

*End thread execution.*

- void [join](#) ()

*Join thread.*

- void [terminate](#) ()

*Terminate thread execution.*

## Static Public Member Functions

- void [sleep](#) (unsigned milliseconds)

*sleep function*

## Protected Member Functions

- bool [isCancelled](#) ()

## Protected Attributes

- [WlanPeerInfo](#) `activePeer`
- `vector< WlanPeerInfo >` `peerList`
- `bool` `activePeerValid`
- `bool` `peerListValid`

## Private Member Functions

- `bool` `initSocket` (string `interfaceName`)
- `void` `closeSocket` ()
- `bool` `getEssid` (string &`essid`)
- `bool` `getApAddress` (string &`apAddress`)
- `bool` `getApList` (`vector< WlanPeerInfo >` &`apList`, `bool` `changeEssid`)
- `bool` `getMode` ([WlanActiveModeFeature::FeatureKind](#) &`mode`)
- `bool` `getApSignalLevel` (double &`level`)

## Private Attributes

- string `ifname`
- int `wsocket`
- int `delay`
- Lock `lockApList`

### 6.76.1 Detailed Description

#### [Todo](#)

documentation

Definition at line 34 of file `WlanLinux.h`.

### 6.76.2 Member Enumeration Documentation

**6.76.2.1** enum [WlanFeatureProvider::FeatureKind](#) [inherited]

#### [Todo](#)

documentation

Definition at line 245 of file `Wlan.h`.

### 6.76.3 Constructor & Destructor Documentation

**6.76.3.1** [WlanLinuxFeatureProvider::WlanLinuxFeatureProvider](#) ([featureparams](#) & *params*)

#### [Todo](#)

documentation

Definition at line 111 of file `WlanLinux.cpp`.

References `WLAN_INTERFACENAME`, and `WLAN_SCANDELAY`.

## 6.76.4 Member Function Documentation

### 6.76.4.1 void WlanLinuxFeatureProvider::closeSocket () [private]

#### Todo

documentation

Definition at line 182 of file WlanLinux.cpp.

### 6.76.4.2 bool WlanLinuxFeatureProvider::getApAddress (string & *apAddress*) [private]

#### Todo

documentation

Definition at line 231 of file WlanLinux.cpp.

References ifname.

### 6.76.4.3 bool WlanLinuxFeatureProvider::getApList (vector< WlanPeerInfo > & *apList*, bool *changeEssid*) [private]

#### Todo

documentation

Definition at line 276 of file WlanLinux.cpp.

References getEssid(), ifname, and setEssid().

Referenced by run().

### 6.76.4.4 bool WlanLinuxFeatureProvider::getApSignalLevel (double & *level*) [private]

#### Todo

documentation

Definition at line 248 of file WlanLinux.cpp.

References decodeSignalLevel(), ifname, and wlsocket.

### 6.76.4.5 bool WlanLinuxFeatureProvider::getEssid (string & *essid*) [private]

#### Todo

documentation

Definition at line 186 of file WlanLinux.cpp.

References ifname.

Referenced by getApList().



#### 6.76.4.6 **Feature** \* WlanFeatureProvider::getFeature (string *name*) const [virtual, inherited]

Query feature instance.

Query the feature provider for a specific feature initialized with random values.

##### Parameters:

*name* Name of the requested feature.

##### Returns:

An instance of the requested feature.

Implements [FeatureProvider](#).

Definition at line 260 of file Wlan.cpp.

References WlanFeatureProvider::activeEssidFeature\_code, WlanFeatureProvider::activeEssidFeature\_maxlen, WlanFeatureProvider::activeMacAddressFeature\_code, WlanFeatureProvider::activeMacAddressFeature\_maxlen, WlanFeatureProvider::activeSignalLevelFeature\_maxval, WlanFeatureProvider::activeSignalLevelFeature\_minval, WlanFeatureProvider::featuremap, WlanFeatureProvider::numPeersFeature\_maxval, WlanFeatureProvider::numPeersFeature\_minval, WlanFeatureProvider::peersFeature\_code, and WlanFeatureProvider::peersFeature\_maxlen.

#### 6.76.4.7 **stringvector**\* FeatureProvider::getFeatureList () const [inline, inherited]

Query list of provided features.

##### Returns:

A list of feature names provided by that feature provider.

Definition at line 497 of file Feature.h.

Referenced by FeatureContainer::loadFeature().

#### 6.76.4.8 **bool** WlanLinuxFeatureProvider::getMode (**WlanActiveModeFeature::FeatureKind** & *mode*) [private]

##### Todo

documentation

Definition at line 210 of file WlanLinux.cpp.

References decodeMode(), and ifname.

#### 6.76.4.9 **string** FeatureProvider::getName () const [inline, inherited]

Query provider name.

##### Returns:

A global unique name for the feature provider.

Definition at line 487 of file Feature.h.

**6.76.4.10** [Feature](#) \* **WlanLinuxFeatureProvider::getSample (string *name*) const** [virtual]

Query sample instance.

Query the feature provider for a sample value of the specified feature. Prior to this method you have to invoke [nextSample\(\)](#).

**Parameters:**

*name* Name of the requested feature.

**Returns:**

An instance of the sample.

**See also:**

[nextSample](#)

Reimplemented from [WlanFeatureProvider](#).

Definition at line 169 of file WlanLinux.cpp.

References ifname, and wlsocket.

**6.76.4.11** **bool WlanLinuxFeatureProvider::initSocket (string *interfaceName*)** [private]**Todo**

documentation

Definition at line 177 of file WlanLinux.cpp.

References wlsocket.

**6.76.4.12** **bool Thread::isCancelled ()** [protected, inherited]**Returns:**

*true* if the thread is cancelled, *false* if the thread is still running

Definition at line 43 of file Thread.cpp.

References Thread::cancelled.

Referenced by GSMFeatureProvider::nextSample(), run(), SystemCommandStringListFeatureProvider::run(), and BluetoothLinuxFeatureProvider::run().

**6.76.4.13** **void WlanLinuxFeatureProvider::nextSample (clock\_t *checkpoint*)** [virtual]**Todo**

clean up

Reimplemented from [WlanFeatureProvider](#).

Definition at line 143 of file WlanLinux.cpp.

**6.76.4.14 void Thread::sleep (unsigned *milliseconds*)** [static, inherited]

sleep function

**Parameters:**

*milliseconds* Timeout in milliseconds

Definition at line 80 of file ThreadPosix.cpp.

Referenced by Main(), GSMFeatureProvider::readLine(), run(), SystemCommandStringListFeatureProvider::run(), Scanner::run(), and BluetoothLinuxFeatureProvider::run().

**6.76.5 Member Data Documentation****6.76.5.1 WlanPeerInfo WlanFeatureProvider::activePeer** [protected, inherited]**Todo**

documentation

Definition at line 277 of file Wlan.h.

Referenced by WlanFeatureProvider::getSample().

**6.76.5.2 bool WlanFeatureProvider::activePeerValid** [protected, inherited]**Todo**

documentation

Definition at line 279 of file Wlan.h.

Referenced by WlanFeatureProvider::getSample(), and WlanFeatureProvider::WlanFeatureProvider().

**6.76.5.3 int WlanLinuxFeatureProvider::delay** [private]**Todo**

documentation

Definition at line 41 of file WlanLinux.h.

Referenced by run().

**6.76.5.4 string WlanLinuxFeatureProvider::ifname** [private]**Todo**

documentation

Definition at line 39 of file WlanLinux.h.

Referenced by getApAddress(), getApList(), getApSignalLevel(), getEssid(), getMode(), and getSample().

**6.76.5.5** Lock [WlanLinuxFeatureProvider::lockApList](#) [mutable, private]

**Todo**

documentation

Definition at line 42 of file WlanLinux.h.

**6.76.5.6** [vector<WlanPeerInfo>](#) [WlanFeatureProvider::peerList](#) [protected, inherited]

**Todo**

documentation

Definition at line 278 of file Wlan.h.

Referenced by [WlanFeatureProvider::getSample\(\)](#).

**6.76.5.7** bool [WlanFeatureProvider::peerListValid](#) [protected, inherited]

**Todo**

documentation

Definition at line 279 of file Wlan.h.

Referenced by [WlanFeatureProvider::getSample\(\)](#), and [WlanFeatureProvider::WlanFeatureProvider\(\)](#).

**6.76.5.8** int [WlanLinuxFeatureProvider::wsocket](#) [private]

**Todo**

documentation

Definition at line 40 of file WlanLinux.h.

Referenced by [getApSignalLevel\(\)](#), [getSample\(\)](#), and [initSocket\(\)](#).

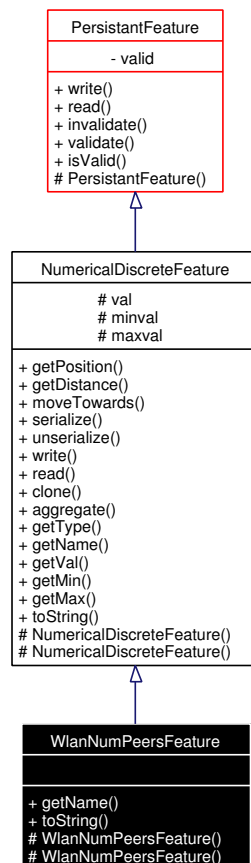
The documentation for this class was generated from the following files:

- [WlanLinux.h](#)
- [WlanLinux.cpp](#)

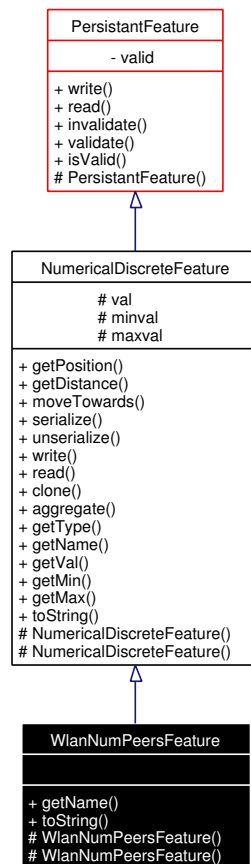
## 6.77 WlanNumPeersFeature Class Reference

```
#include <Wlan.h>
```

Inheritance diagram for WlanNumPeersFeature:



Collaboration diagram for WlanNumPeersFeature:



## Public Types

- enum `FeatureType` {  
**boolean, nominal, ordinal, numerical\_discrete,**  
**numerical\_continuous** }  
*Possible feature types.*

## Public Member Functions

- virtual const string `getName ()` const  
*Query a features name.*
- virtual string `toString ()` const  
*This is only for testing.*
- virtual double `getPosition ()` const  
*Query a features position.*
- virtual double `getDistance (Feature *f)` const  
*Calculates the distance between two features.*

- virtual void [moveTowards](#) ([Feature](#) \*f, double factor)

*Move feature.*

- virtual string [serialize](#) () const

*Serialize a samples data to a string.*

- virtual void [unserialize](#) (string value)

*Unserialize a samples data from a string.*

- virtual [featureparams](#) [write](#) () const

*Externalize feature.*

- virtual void [read](#) ([featureparams](#) \*param)

*Load feature from persistant data.*

- virtual [Feature](#) \* [clone](#) () const

*Clone a feature.*

- virtual void [aggregate](#) ([aggregatelist](#) samples)

*Aggregate a sample values from other sources.*

- virtual [FeatureType](#) [getType](#) () const

*Query a features type.*

- const long [getVal](#) () const

- const long [getMin](#) () const

- const long [getMax](#) () const

- void [invalidate](#) ()

*Invalidate feature.*

- void [validate](#) ()

*Set the validation flag to true.*

- bool [isValid](#) ()

*Query validation flag.*

- bool [isExternalizable](#) ()

*Query externalization flag.*

## Protected Member Functions

- [WlanNumPeersFeature](#) (long \*minval, long \*maxval)

*This constructor should not be used to construct feature objects.*

- [WlanNumPeersFeature](#) (long \*minval, long \*maxval, const long numPeers)

*This constructor should not be used to construct feature objects.*

## Protected Attributes

- double [val](#)  
*The feature value.*
- long \* [minval](#)  
*A reference to the static, persistent minimum value seen so far.*
- long \* [maxval](#)  
*A reference to the static, persistent maximum value seen so far.*
- const bool [externalize](#)  
*Externalization flag.*

### 6.77.1 Detailed Description

#### [Todo](#)

documentation

Definition at line 211 of file Wlan.h.

### 6.77.2 Constructor & Destructor Documentation

#### 6.77.2.1 **WlanNumPeersFeature::WlanNumPeersFeature** (long \* *minval*, long \* *maxval*) [inline, protected]

This constructor should not be used to construct feature objects.

They are only created by the respective feature provider. Definition at line 220 of file Wlan.h.

#### 6.77.2.2 **WlanNumPeersFeature::WlanNumPeersFeature** (long \* *minval*, long \* *maxval*, const long *numPeers*) [inline, protected]

This constructor should not be used to construct feature objects.

They are only created by the respective feature provider. Definition at line 226 of file Wlan.h.

### 6.77.3 Member Function Documentation

#### 6.77.3.1 **void NumericalDiscreteFeature::aggregate** ([aggregatelist](#) *samples*) [virtual, inherited]

Aggregate a sample values from other sources.

#### Parameters:

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 159 of file Numerical.cpp.

References [aggregatelist](#).



**6.77.3.2** `Feature * NumericalDiscreteFeature::clone () const` [virtual, inherited]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Implements [Feature](#).

Definition at line 154 of file Numerical.cpp.

References [NumericalDiscreteFeature::maxval](#), [NumericalDiscreteFeature::minval](#), [NumericalDiscreteFeature::NumericalDiscreteFeature\(\)](#), and [NumericalDiscreteFeature::val](#).

**6.77.3.3** `double NumericalDiscreteFeature::getDistance (Feature * f) const` [virtual, inherited]

Calculates the distance between two features.

**Parameters:**

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Implements [Feature](#).

Definition at line 73 of file Numerical.cpp.

References [NumericalDiscreteFeature::getPosition\(\)](#), [NumericalDiscreteFeature::maxval](#), and [NumericalDiscreteFeature::minval](#).

**6.77.3.4** `virtual const string WlanNumPeersFeature::getName () const` [inline, virtual]

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Reimplemented from [NumericalDiscreteFeature](#).

Definition at line 229 of file Wlan.h.

**6.77.3.5** `double NumericalDiscreteFeature::getPosition () const` [virtual, inherited]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Implements [Feature](#).

Definition at line 57 of file Numerical.cpp.

References `NumericalDiscreteFeature::maxval`, `NumericalDiscreteFeature::minval`, and `NumericalDiscreteFeature::val`.

Referenced by `NumericalDiscreteFeature::getDistance()`.

**6.77.3.6** `virtual FeatureType NumericalDiscreteFeature::getType () const` [`inline`, `virtual`, `inherited`]

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 103 of file Numerical.h.

**6.77.3.7** `const long NumericalDiscreteFeature::getVal () const` [`inline`, `inherited`]

**Returns:**

The value of the feature.

Definition at line 107 of file Numerical.h.

References `NumericalDiscreteFeature::val`.

Referenced by `Java_at_jku_intelligence_samples_NumericalDiscreteSample_nativeGetVal()`, `toString()`, `NumericalDiscreteFeature::toString()`, and `BluetoothNumPeersFeature::toString()`.

**6.77.3.8** `void PersistentFeature::invalidate ()` [`inline`, `inherited`]

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by `AbstractStringFeature::getCodeForName()`, `NumericalContinuousFeature::NumericalContinuousFeature()`, `NumericalDiscreteFeature::NumericalDiscreteFeature()`, `TimeFeature::TimeFeature()`, `NumericalContinuousFeature::unserialize()`, and `NumericalDiscreteFeature::unserialize()`.

**6.77.3.9** `bool Feature::isExternalizable ()` [`inline`, `inherited`]

Query externalization flag.

**Returns:**

*true* if the feature is externalizable ( i.e. it has persistent data that should be stored across restarts), *false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by `FeatureContainer::FeatureContainer()`, and `FeatureContainer::nextSample()`.

**6.77.3.10 bool PersistentFeature::isValid ()** [inline, inherited]

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

**6.77.3.11 void NumericalDiscreteFeature::moveTowards (Feature \*f, double factor)** [virtual, inherited]

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

**Parameters:**

*f* [Feature](#) used for distance measurement.

*factor* Distance weight.

Implements [Feature](#).

Definition at line 80 of file Numerical.cpp.

References [NumericalDiscreteFeature::maxval](#), [NumericalDiscreteFeature::minval](#), and [NumericalDiscreteFeature::val](#).

**6.77.3.12 void NumericalDiscreteFeature::read (featureparams \*param)** [virtual, inherited]

Load feature from persistent data.

Initializes the persistent feature data from the given representation.

**Parameters:**

*param* Persistent feature data.

**See also:**

[write](#)

Implements [PersistentFeature](#).

Definition at line 141 of file Numerical.cpp.

References [featureparams](#), [NumericalDiscreteFeature::maxval](#), and [NumericalDiscreteFeature::minval](#).

**6.77.3.13 string NumericalDiscreteFeature::serialize () const** [virtual, inherited]

Serialize a samples data to a string.

**Returns:**

String representation of the samples data.

Implements [Feature](#).

Definition at line 96 of file Numerical.cpp.

References NumericalDiscreteFeature::val.

**6.77.3.14 void NumericalDiscreteFeature::unserialize (string *value*)** [virtual, inherited]

Unserialize a samples data from a string.

**Parameters:**

*value* String representation of the samples data.

Implements [Feature](#).

Definition at line 104 of file Numerical.cpp.

References PersistentFeature::invalidate(), NumericalDiscreteFeature::maxval, NumericalDiscreteFeature::minval, and NumericalDiscreteFeature::val.

**6.77.3.15 featureparams NumericalDiscreteFeature::write () const** [virtual, inherited]

Externalize feature.

**Returns:**

Persistent feature data.

**See also:**

[read](#)

Implements [PersistentFeature](#).

Definition at line 127 of file Numerical.cpp.

References featureparams, NumericalDiscreteFeature::maxval, and NumericalDiscreteFeature::minval.

**6.77.4 Member Data Documentation****6.77.4.1 const bool [Feature::externalize](#)** [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistent features, the object should be cast to PersistentFeature, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**

[PersistentFeature](#)

Definition at line 187 of file Feature.h.

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

- [Wlan.h](#)
- [Wlan.cpp](#)

## 6.78 WlanPeerInfo Struct Reference

```
#include <Wlan.h>
```

### Public Attributes

- string [essid](#)
- string [mac](#)
- [WlanActiveModeFeature::FeatureKind](#) mode
- double [signalLevel](#)

### 6.78.1 Detailed Description

**Todo**

documentation

Definition at line 169 of file Wlan.h.

### 6.78.2 Member Data Documentation

#### 6.78.2.1 string [WlanPeerInfo::essid](#)

**Todo**

documentation

Definition at line 172 of file Wlan.h.

Referenced by [WlanFeatureProvider::getSample\(\)](#).

#### 6.78.2.2 string [WlanPeerInfo::mac](#)

**Todo**

documentation

Definition at line 173 of file Wlan.h.

Referenced by [WlanFeatureProvider::getSample\(\)](#).

#### 6.78.2.3 [WlanActiveModeFeature::FeatureKind](#) [WlanPeerInfo::mode](#)

**Todo**

documentation

Definition at line 174 of file Wlan.h.

Referenced by [WlanFeatureProvider::getSample\(\)](#).

#### 6.78.2.4 double [WlanPeerInfo::signalLevel](#)

##### [Todo](#)

documentation

Definition at line 175 of file Wlan.h.

Referenced by [WlanFeatureProvider::getSample\(\)](#).

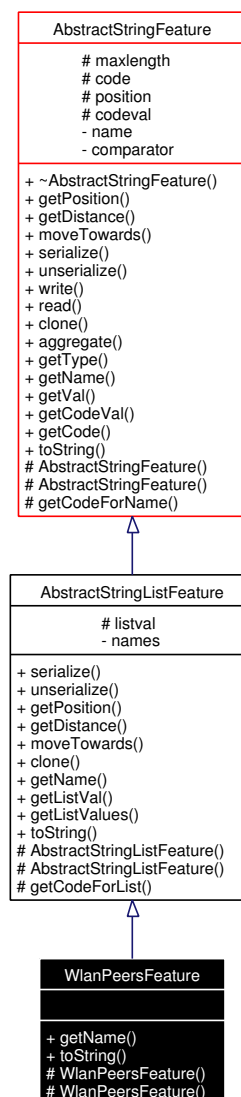
The documentation for this struct was generated from the following file:

- [Wlan.h](#)

## 6.79 WlanPeersFeature Class Reference

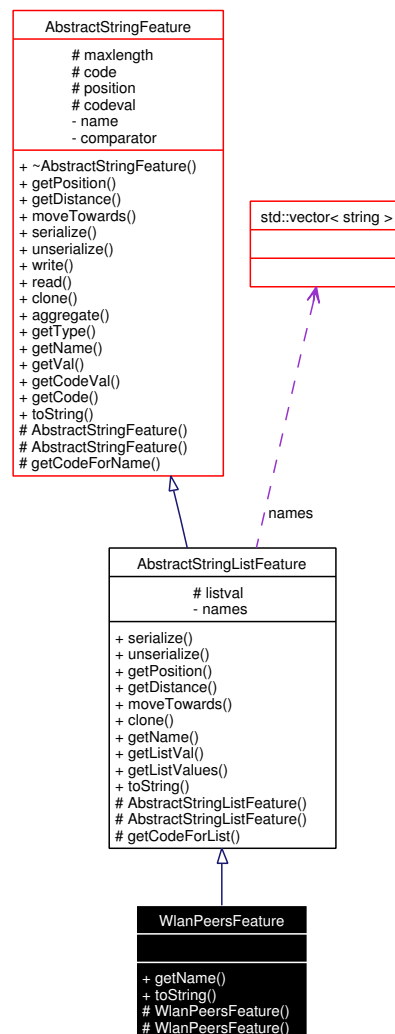
```
#include <Wlan.h>
```

Inheritance diagram for WlanPeersFeature:



Collaboration diagram for WlanPeersFeature:





## Public Types

- enum **ComparatorType** { **None**, **Levenshtein** }

*The distance metric to use.*

- enum **FeatureType** {  
**boolean**, **nominal**, **ordinal**, **numerical\_discrete**,  
**numerical\_continuous** }

*Possible feature types.*

## Public Member Functions

- virtual const string **getName** () const

*Query a features name.*

- virtual string [toString](#) () const  
*This is only for testing.*
- virtual string [serialize](#) () const  
*Serialize a samples data to a string.*
- virtual void [unserialize](#) (string value)  
*Unserialize a samples data from a string.*
- virtual double [getPosition](#) () const  
*Query a features position.*
- virtual double [getDistance](#) (Feature \*f) const  
*Calculates the distance between two features.*
- virtual void [moveTowards](#) (Feature \*f, double factor)  
*Move feature.*
- virtual Feature \* [clone](#) () const  
*Clone a feature.*
- const bit\_vector & [getListVal](#) () const  
*Query the list of values.*
- const stringvector & [getListValues](#) () const  
*Query the list of values.*
- virtual [featureparams write](#) () const  
*Externalize feature.*
- virtual void [read](#) (featureparams \*param)  
*Load feature from persistant data.*
- virtual void [aggregate](#) (aggregatelist samples)  
*Aggregate a sample values from other sources.*
- virtual FeatureType [getType](#) () const  
*Query a features type.*
- const string & [getVal](#) () const  
*Query the string values.*
- const unsigned long [getCodeVal](#) () const
- const [stringcode](#) \* [getCode](#) ()
- void [invalidate](#) ()  
*Invalidate feature.*

- void [validate](#) ()  
*Set the validation flag to true.*
- bool [isValid](#) ()  
*Query validation flag.*
- bool [isExternalizable](#) ()  
*Query externalization flag.*

## Protected Member Functions

- [WlanPeersFeature](#) ([stringcode](#) \*code, long \*maxlen)  
*This constructor should not be used to construct feature objects.*
- [WlanPeersFeature](#) ([stringcode](#) \*code, long \*maxlen, const [vector](#)< [WlanPeerInfo](#) > \*peers)  
*This constructor should not be used to construct feature objects.*
- bit\_vector [getCodeForList](#) (const [stringvector](#) \*names)  
*Get a code value for a given stringvector.*
- unsigned long [getCodeForName](#) (const string &name)  
*Get a code value for a given string.*

## Protected Attributes

- bit\_vector [listval](#)  
*The coded value of the feature.*
- long \* [maxlength](#)  
*A reference to the static, persistent maximum length of strings seen so far.*
- [stringcode](#) \* [code](#)  
*A reference to the static, persistent code table of strings seen so far.*
- char \* [position](#)  
*Only for internal usage in moveTowards and getDistance.*
- unsigned long [codeval](#)  
*The coded value of the feature.*
- const bool [externalize](#)  
*Externalization flag.*

## Related Functions

(Note that these are not member functions.)

- long [levenshtein](#) (char \*\*x, const char \*t, double \*factor)

*Modified Levenshtein algorithm.*

### 6.79.1 Detailed Description

#### [Todo](#)

documentation

Definition at line 182 of file Wlan.h.

### 6.79.2 Constructor & Destructor Documentation

#### 6.79.2.1 WlanPeersFeature::WlanPeersFeature ([stringcode](#) \* code, long \* maxlen) [protected]

This constructor should not be used to construct feature objects.

They are only created by the respective feature provider. Definition at line 139 of file Wlan.cpp.

References [stringcode](#).

#### 6.79.2.2 WlanPeersFeature::WlanPeersFeature ([stringcode](#) \* code, long \* maxlen, const [vector](#)<[WlanPeerInfo](#) > \* peers) [protected]

This constructor should not be used to construct feature objects.

They are only created by the respective feature provider. Definition at line 143 of file Wlan.cpp.

References [AbstractStringListFeature::getCodeForList\(\)](#), [stringcode](#), and [stringvector](#).

### 6.79.3 Member Function Documentation

#### 6.79.3.1 void AbstractStringFeature::aggregate ([aggregatelist](#) samples) [virtual, inherited]

Aggregate a sample values from other sources.

##### Parameters:

*samples* A list of <timestamp, sample> tuples.

Implements [Feature](#).

Definition at line 421 of file [AbstractString.cpp](#).

References [aggregatelist](#).

**6.79.3.2** `Feature * AbstractStringListFeature::clone () const` [virtual, inherited]

Clone a feature.

A feature must be able to clone itself. This is typically implemented with a copy constructor, but should return a new object.

Reimplemented from [AbstractStringFeature](#).

Definition at line 581 of file `AbstractString.cpp`.

References `AbstractStringListFeature::AbstractStringListFeature()`, `AbstractStringListFeature::listval`, and `AbstractStringListFeature::names`.

**6.79.3.3** `bit_vector AbstractStringListFeature::getCodeForList (const stringvector * names)`  
[protected, inherited]

Get a code value for a given stringvector.

The function composes a `bit_vector` wherein for each string in the stringvector the bit on the position of the strings code value is set to *true*. The remaining bits are initialized to *false*.

**Parameters:**

*names* The stringvector to code

**Returns:**

A code value as `bit_vector`

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

References `AbstractStringFeature::getCodeForName()`, and `stringvector`.

Referenced by `AbstractStringListFeature::AbstractStringListFeature()`, and `WlanPeersFeature()`.

**6.79.3.4** `unsigned long AbstractStringFeature::getCodeForName (const string & name)`  
[protected, inherited]

Get a code value for a given string.

A helper function to return the respective code for a given string. If the string is encountered the first time a new code is allocated.

**Parameters:**

*name* The string to code

**Returns:**

A code value

Definition at line 303 of file `AbstractString.cpp`.

References `AbstractStringFeature::code`, `PersistentFeature::invalidate()`, and `AbstractStringFeature::maxlength`.

Referenced by `AbstractStringFeature::AbstractStringFeature()`, and `AbstractStringListFeature::getCodeForList()`.

**6.79.3.5** `double AbstractStringListFeature::getDistance (Feature *f) const` [virtual, inherited]

Calculates the distance between two features.

**Parameters:**

*f* [Feature](#) used for distance measurement.

**Returns:**

Distance between the two samples. The return value has to be in the interval  $[0; 1]$  to guarantee comparability among samples of different features.

Reimplemented from [AbstractStringFeature](#).

Definition at line 518 of file `AbstractString.cpp`.

References `AbstractStringListFeature::listval`.

**6.79.3.6** `const bit_vector& AbstractStringListFeature::getListVal () const` [inline, inherited]

Query the list of values.

**Returns:**

Values list

Definition at line 244 of file `AbstractString.h`.

**6.79.3.7** `const stringvector& AbstractStringListFeature::getListValues () const` [inline, inherited]

Query the list of values.

**Returns:**

Values list

Definition at line 253 of file `AbstractString.h`.

References `stringvector`.

Referenced by `Java_at_jku_intelligence_samples_StringListSample_nativeGetValues()`, `toString()`, `SystemCommandStringListFeature::toString()`, `BluetoothPeersFeature::toString()`, and `AbstractStringListFeature::toString()`.

**6.79.3.8** `virtual const string WlanPeersFeature::getName () const` [inline, virtual]

Query a features name.

**Returns:**

Name of the [Feature](#) in the format "Featureprovider.Feature"

Reimplemented from [AbstractStringListFeature](#).

Definition at line 200 of file `Wlan.h`.

**6.79.3.9 double AbstractStringListFeature::getPosition () const** [virtual, inherited]

Query a features position.

**Returns:**

The distance to the origin. Every implementation must take care that only values in the interval  $[0; 1]$  are returned.

**Remarks:**

This function is only for the purpose of internal search operations and visualisation and should not be used in any other context.

Reimplemented from [AbstractStringFeature](#).

Definition at line 498 of file AbstractString.cpp.

References AbstractStringListFeature::listval.

**6.79.3.10 virtual FeatureType AbstractStringFeature::getType () const** [inline, virtual, inherited]

Query a features type.

**Returns:**

The type of the [Feature](#).

Implements [Feature](#).

Definition at line 100 of file AbstractString.h.

**6.79.3.11 const string& AbstractStringFeature::getVal () const** [inline, inherited]

Query the string values.

**Returns:**

Values list

Definition at line 108 of file AbstractString.h.

References AbstractStringFeature::name.

Referenced by Java\_at\_jku\_intelligence\_samples\_StringSample\_nativeGetVal().

**6.79.3.12 void PersistentFeature::invalidate ()** [inline, inherited]

Invalidate feature.

Sets the validation flag to *false* to signal the framework that the corresponding persistent feature data is not up to date and has to be written again. Definition at line 370 of file Feature.h.

Referenced by AbstractStringFeature::getCodeForName(), NumericalContinuousFeature::NumericalContinuousFeature(), NumericalDiscreteFeature::NumericalDiscreteFeature(), TimeFeature::TimeFeature(), NumericalContinuousFeature::unserialize(), and NumericalDiscreteFeature::unserialize().

**6.79.3.13 bool Feature::isExternalizable ()** [inline, inherited]

Query externalization flag.

**Returns:**

*true* if the feature is externalizeable ( i.e. it has persistend data that should be stored across restarts),  
*false* otherwise.

Definition at line 250 of file Feature.h.

Referenced by FeatureContainer::FeatureContainer(), and FeatureContainer::nextSample().

**6.79.3.14 bool PersistentFeature::isValid ()** [inline, inherited]

Query validation flag.

**Returns:**

*true* if the features persistent data is up to date, *false* otherwise.

Definition at line 381 of file Feature.h.

Referenced by FeatureContainer::nextSample().

**6.79.3.15 void AbstractStringListFeature::moveTowards (Feature \**f*, double *factor*)**  
[virtual, inherited]

Move feature.

Moves the feature towards a given sample value by the specified *factor* (*factor* times the distance). If the *factor* is 0, the old feature value must not change. If the *factor* is 1, the new feature value should be equivalent to the sample *s*.

**Parameters:**

*f* [Feature](#) used for distance measurement.

*factor* Distance weight.

Reimplemented from [AbstractStringFeature](#).

Definition at line 546 of file AbstractString.cpp.

References AbstractStringListFeature::listval, and rand\_double.

**6.79.3.16 void AbstractStringFeature::read (featureparams \**param*)** [virtual, inherited]

Load feature from persistent data.

Initializes the persistent feature data from the given representation.

**Parameters:**

*param* Persistent feature data.

**See also:**

[write](#)



Implements [PersistantFeature](#).

Definition at line 296 of file AbstractString.cpp.

References AbstractStringFeature::code, and featureparams.

#### 6.79.3.17 `string AbstractStringListFeature::serialize () const` [virtual, inherited]

Serialize a samples data to a string.

##### Returns:

String representation of the samples data.

Reimplemented from [AbstractStringFeature](#).

Definition at line 470 of file AbstractString.cpp.

References AbstractStringListFeature::listval.

Referenced by toString(), BluetoothPeersFeature::toString(), and AbstractStringListFeature::toString().

#### 6.79.3.18 `void AbstractStringListFeature::unserialize (string value)` [virtual, inherited]

Unserialize a samples data from a string.

##### Parameters:

*value* String representation of the samples data.

Reimplemented from [AbstractStringFeature](#).

Definition at line 480 of file AbstractString.cpp.

References AbstractStringListFeature::listval.

#### 6.79.3.19 `featureparams AbstractStringFeature::write () const` [virtual, inherited]

Externalize feature.

##### Returns:

Persistent feature data.

##### See also:

[read](#)

Implements [PersistantFeature](#).

Definition at line 283 of file AbstractString.cpp.

References AbstractStringFeature::code, and featureparams.

### 6.79.4 Friends And Related Function Documentation

#### 6.79.4.1 `long levenshtein (char ** x, const char * t, double * factor)` [related, inherited]

Modified Levenshtein algorithm.

This function implements a slightly modified version of the Levenshtein algorithm to not only calculate the distance of two strings but also modify the string  $x$  towards the string  $t$  with the propability *factor*. In other words, every transformation operation found by the algorithm is performed on the string  $x$  with the propability *factor*.

**Parameters:**

$x$  Input string.

$t$  String to compare the input string to.

*factor* Propability with witch transformations are performed. Has to be a value out of the intervall  $[0; 1]$ .

**Returns:**

The levenshtein distance between  $x$  and  $t$ .

**Todo**

alloc once

Definition at line 46 of file AbstractString.cpp.

References rand\_double.

Referenced by AbstractStringFeature::getDistance(), and AbstractStringFeature::moveTowards().

## 6.79.5 Member Data Documentation

### 6.79.5.1 unsigned long [AbstractStringFeature::codeval](#) [protected, inherited]

The coded value of the feature.

**See also:**

[name](#)

[code](#)

Definition at line 144 of file AbstractString.h.

Referenced by AbstractStringFeature::AbstractStringFeature(), AbstractStringFeature::clone(), AbstractStringFeature::getCodeVal(), AbstractStringFeature::getDistance(), AbstractStringFeature::moveTowards(), AbstractStringFeature::serialize(), and AbstractStringFeature::unserialize().

### 6.79.5.2 const bool [Feature::externalize](#) [protected, inherited]

Externalization flag.

If true, then this feature has persistent data which should be preserved across restarts of the application (e.g. a list of already seen feature values for nominal and ordinal types or maximum and minimum values for numerical types).

For all persistant features, the object should be cast to PersistentFeature, because only subclasses of this type are (by policy) allowed to set this variable to true.

**See also:**

[PersistantFeature](#)

Definition at line 187 of file Feature.h.

### 6.79.5.3 `bit_vector` [AbstractStringListFeature::listval](#) [protected, inherited]

The coded value of the feature.

**See also:**

[names](#)  
[code](#)

Definition at line 180 of file `AbstractString.h`.

Referenced by `AbstractStringListFeature::AbstractStringListFeature()`, `AbstractStringListFeature::clone()`, `AbstractStringListFeature::getDistance()`, `AbstractStringListFeature::getPosition()`, `AbstractStringListFeature::moveTowards()`, `AbstractStringListFeature::serialize()`, and `AbstractStringListFeature::unserialize()`.

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

- [Wlan.h](#)
- [Wlan.cpp](#)



## Chapter 7

# Intelligence File Documentation

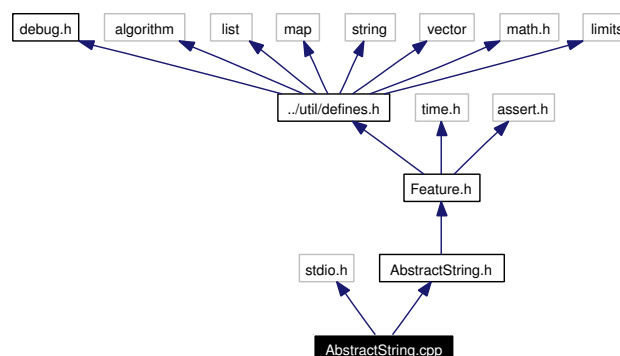
### 7.1 AbstractString.cpp File Reference

Abstract string feature class implementation.

```
#include <stdio.h>
```

```
#include "AbstractString.h"
```

Include dependency graph for AbstractString.cpp:



#### 7.1.1 Detailed Description

Abstract string feature class implementation.

©2003-2004 by Rene Mayrhofer, Harald Radi

##### Date

2004/06/04 11:37:10

##### Revision

1.76

##### Since:

Wed Mar 19 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

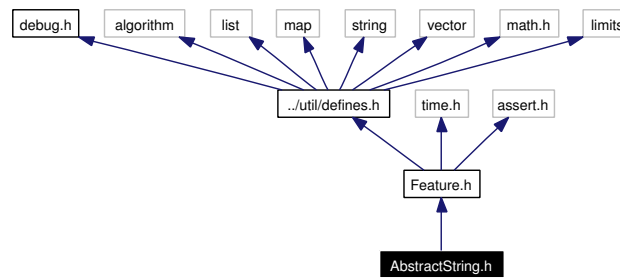
**Id**[AbstractString.cpp](#),v 1.76 2004/06/04 11:37:10 rene ExpDefinition in file [AbstractString.cpp](#).

## 7.2 AbstractString.h File Reference

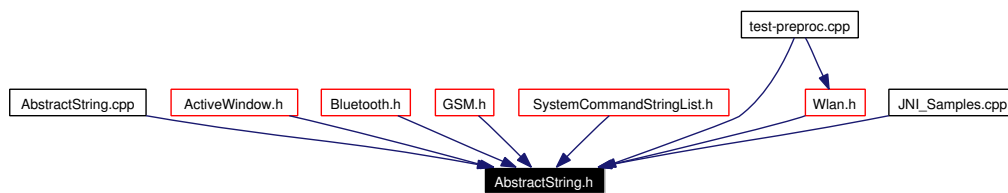
Abstract string feature class declaration.

```
#include "Feature.h"
```

Include dependency graph for AbstractString.h:



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



### Classes

- class [AbstractStringFeature](#)  
*Baseclass for features providing string values.*
- class [AbstractStringListFeature](#)  
*Baseclass for features providing lists of string values.*

### Typedefs

- typedef [map](#)< string, unsigned long > [stringcode](#)  
*An association of seen strings and their assigned numerical code.*

#### 7.2.1 Detailed Description

Abstract string feature class declaration.

This file contains the abstract class definitions that can be used by sensors that return values that can be represented as strings.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/03/01 11:15:30

**Revision**

1.38

**Since:**

Wed Mar 19 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**[AbstractString.h](#),v 1.38 2004/03/01 11:15:30 rene ExpDefinition in file [AbstractString.h](#).

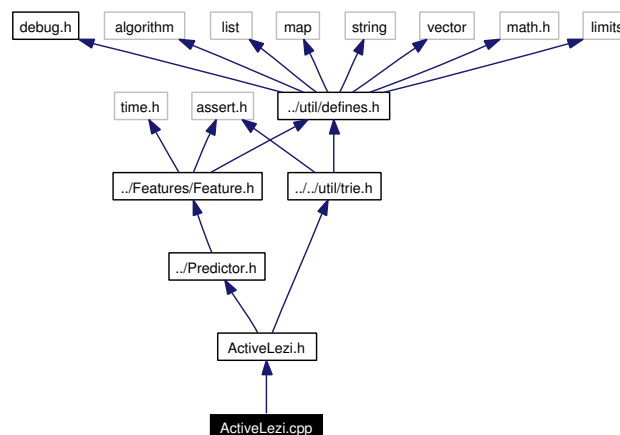


## 7.3 ActiveLezi.cpp File Reference

Active lezi predictor.

```
#include "ActiveLezi.h"
```

Include dependency graph for ActiveLezi.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [PredictorAlgorithm](#) \* [getPredictor](#) ([predictorparams](#) &params)  
*\ Returns an instance of a class implementing the [PredictorAlgorithm](#) interface \ \ param params A map of parameters to initialize the instance \ return A [PredictorAlgorithm](#) implementation \*

### Variables

- [ActiveLezi](#) \* [pred](#)  
*Singleton implementing the [PredictorAlgorithm](#) interface.*

#### 7.3.1 Detailed Description

Active lezi predictor.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/23 16:27:18

**Revision**

1.12

**Since:**

Tue Dec 16 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

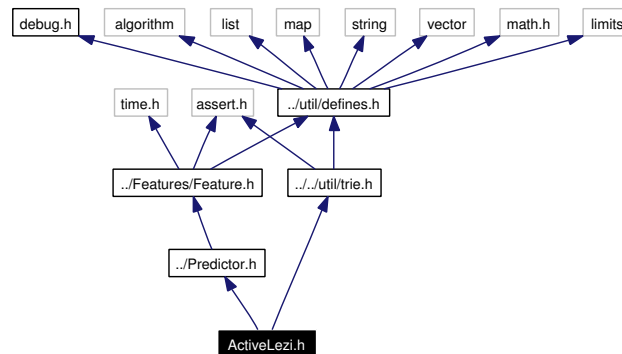
**Id**[ActiveLezi.cpp](#),v 1.12 2004/02/23 16:27:18 rene ExpDefinition in file [ActiveLezi.cpp](#).

## 7.4 ActiveLezi.h File Reference

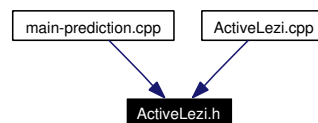
Active lezi predictor.

```
#include "../Predictor.h"
#include "../../util/trie.h"
```

Include dependency graph for ActiveLezi.h:



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



### Classes

- class [ActiveLezi](#)

### 7.4.1 Detailed Description

Active lezi predictor.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.11

#### Since:

Tue Dec 16 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>  
 Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[ActiveLezi.h](#),v 1.11 2004/02/12 20:08:37 harald Exp

Definition in file [ActiveLezi.h](#).

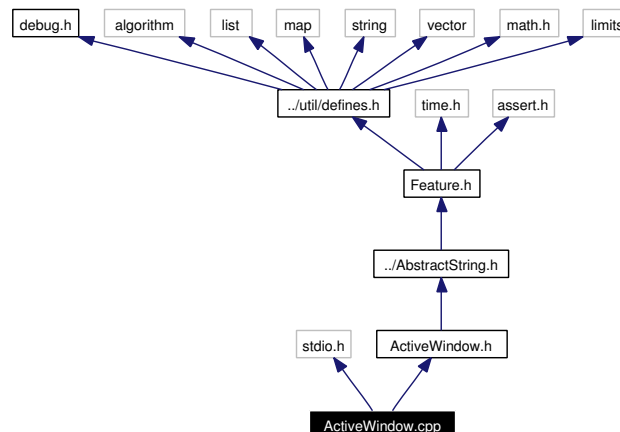
## 7.5 ActiveWindow.cpp File Reference

Active window feature.

```
#include <stdio.h>
```

```
#include "ActiveWindow.h"
```

Include dependency graph for ActiveWindow.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) (providerparams &params)  
*\ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param params A map of parameters to initialize the instance \ return A [FeatureProvider](#) implementation \*

### Variables

- [ActiveWindowFeatureProvider](#) \* fp  
*Singleton implementing the [FeatureProvider](#) interface.*

### 7.5.1 Detailed Description

Active window feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/02 12:36:18

**Revision**

1.21

**Since:**

Wed Apr 9 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

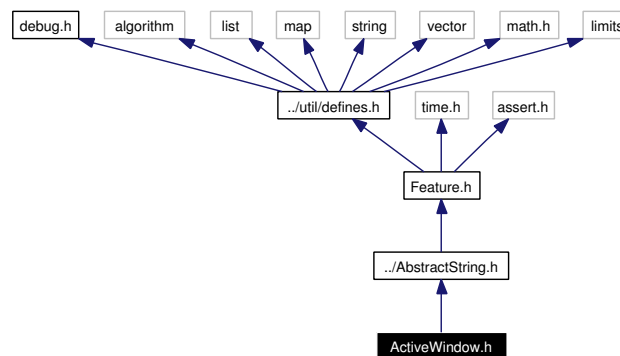
**Id**[ActiveWindow.cpp](#),v 1.21 2004/03/02 12:36:18 rene-cvs ExpDefinition in file [ActiveWindow.cpp](#).

## 7.6 ActiveWindow.h File Reference

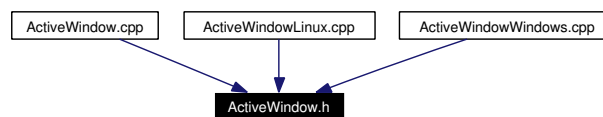
Active window feature.

```
#include "../AbstractString.h"
```

Include dependency graph for ActiveWindow.h:



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



### Classes

- class [ActiveWindowFeature](#)  
*Returns the application with focus.*
- class [ActiveWindowFeatureProvider](#)  
*Provider for the [ActiveWindowFeature](#).*

### 7.6.1 Detailed Description

Active window feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.21

#### Since:

Wed Apr 9 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**[ActiveWindow.h](#),v 1.21 2004/02/12 20:08:37 harald ExpDefinition in file [ActiveWindow.h](#).



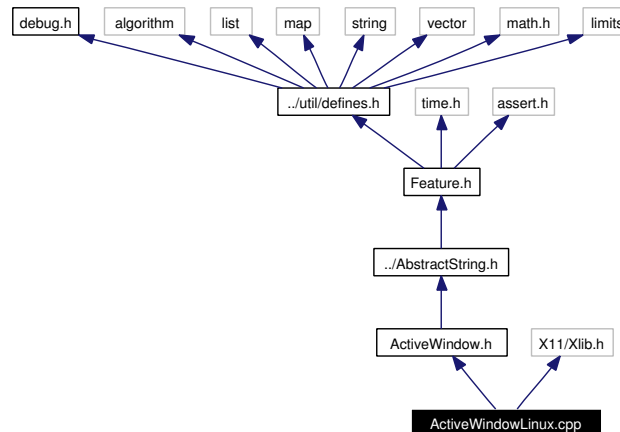
## 7.7 ActiveWindowLinux.cpp File Reference

Active window feature.

```
#include "ActiveWindow.h"
```

```
#include <X11/Xlib.h>
```

Include dependency graph for ActiveWindowLinux.cpp:



### 7.7.1 Detailed Description

Active window feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.11

#### Since:

Wed Apr 9 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[ActiveWindowLinux.cpp](#),v 1.11 2004/02/12 20:08:37 harald Exp

Definition in file [ActiveWindowLinux.cpp](#).

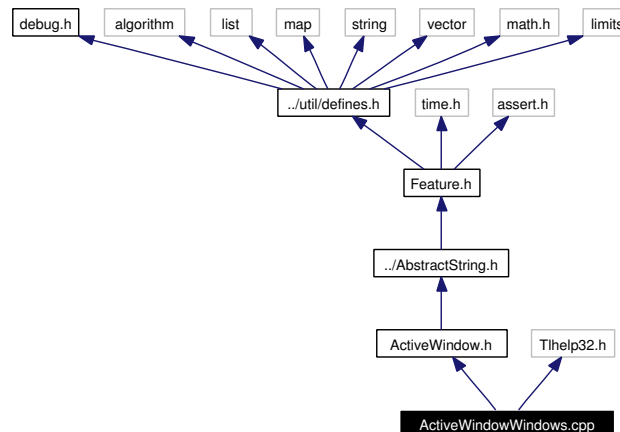
## 7.8 ActiveWindowWindows.cpp File Reference

Active window feature.

```
#include "ActiveWindow.h"
```

```
#include <Tlhelp32.h>
```

Include dependency graph for ActiveWindowWindows.cpp:



### Defines

- #define [CloseSnapshot](#)(ph) CloseHandle(ph);  
*CloseHandle helper macro.*

### 7.8.1 Detailed Description

Active window feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.12

#### Since:

Wed Apr 9 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[ActiveWindowWindows.cpp](#), v 1.12 2004/02/12 20:08:37 harald Exp

Definition in file [ActiveWindowWindows.cpp](#).

## 7.8.2 Define Documentation

### 7.8.2.1 `#define CloseSnapshot(ph) CloseHandle(ph);`

CloseHandle helper macro.

The Toolhelp32Snapshot handle has to be closed differently on WIN32 and CE, this macro provides a unified way to close such handles. Definition at line 37 of file ActiveWindowWindows.cpp.

## 7.9 Audio.cpp File Reference

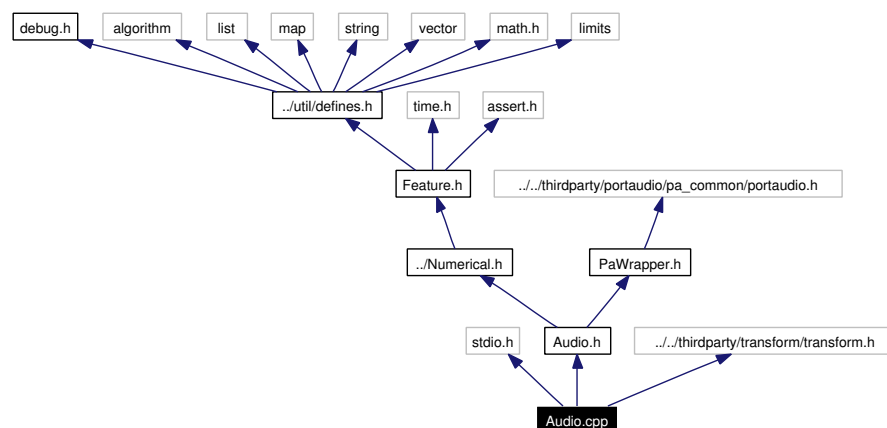
Audio feature.

```
#include <stdio.h>
```

```
#include "Audio.h"
```

```
#include "../thirdparty/transform/transform.h"
```

Include dependency graph for Audio.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) ([providerparams](#) &params)  
*\ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param params A map of parameters to initialize the instance \ return A [FeatureProvider](#) implementation \*

### Variables

- [AudioFeatureProvider](#) \* fp  
*Singleton implementing the [FeatureProvider](#) interface.*

#### 7.9.1 Detailed Description

Audio feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/06/08 09:38:39

**Revision**

1.41

**Since:**

Wed Apr 9 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

**Id**[Audio.cpp](#),v 1.41 2004/06/08 09:38:39 rene ExpDefinition in file [Audio.cpp](#).

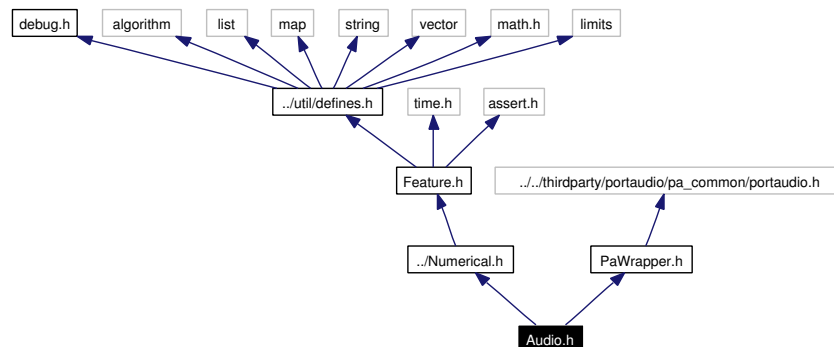
## 7.10 Audio.h File Reference

Audio feature.

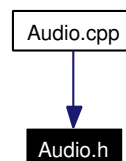
```
#include "../Numerical.h"
```

```
#include "PaWrapper.h"
```

Include dependency graph for Audio.h:



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



### Classes

- class [AudioBandFeature](#)  
*Represents a subband.*
- class [AudioMeanFeature](#)  
*Calculates the mean level of the input signal.*
- class [AudioPeakFeature](#)  
*Counts the peaks in the input signal.*
- class [AudioFeatureProvider](#)  
*Provider for the audio features.*
- struct [AudioFeatureProvider::SampleData](#)

### 7.10.1 Detailed Description

Audio feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/03/02 12:36:18

**Revision**

1.23

**Since:**

Wed Apr 9 2003

**Author:**

Rene Mayrhofer <[reene@mayrhofer.eu.org](mailto:reene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[Audio.h](#),v 1.23 2004/03/02 12:36:18 rene-cvs Exp

Definition in file [Audio.h](#).

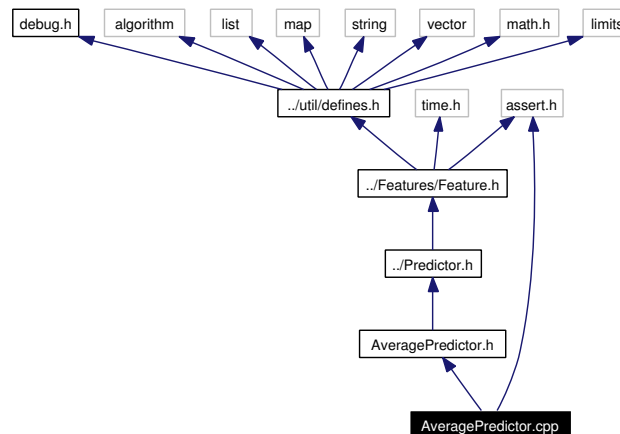
## 7.11 AveragePredictor.cpp File Reference

A simple Predictor using averages (median for ordinal time series, most frequent for categorical/nominal) as predicted next values.

```
#include "AveragePredictor.h"
```

```
#include <assert.h>
```

Include dependency graph for AveragePredictor.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [PredictorAlgorithm](#) \* [getPredictor](#) ([predictorparams](#) &params)  
*\ Returns an instance of a class implementing the [PredictorAlgorithm](#) interface \ \ param params A map of parameters to initialize the instance \ return A [PredictorAlgorithm](#) implementation \*

### Variables

- [AveragePredictor](#) \* [pred](#)  
*Singleton implementing the [PredictorAlgorithm](#) interface.*

#### 7.11.1 Detailed Description

A simple Predictor using averages (median for ordinal time series, most frequent for categorical/nominal) as predicted next values.

©2003-2004 by Rene Mayrhofer, Harald Radi



**Date**

2004/02/23 11:25:16

**Revision**

1.5

**Since:**

Wed Mar 19 2003

**Author:**Rene Mayrhofer <[reene@mayrhofer.eu.org](mailto:reene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

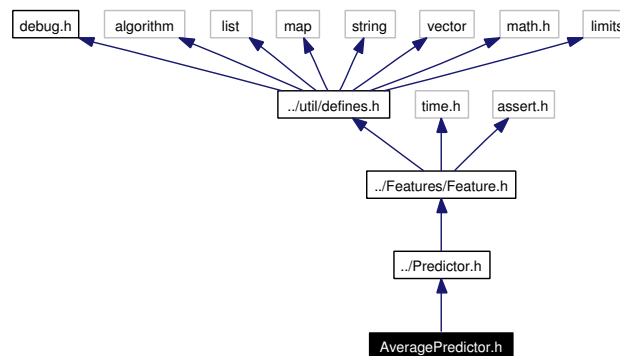
**Id**[AveragePredictor.cpp](#),v 1.5 2004/02/23 11:25:16 rene ExpDefinition in file [AveragePredictor.cpp](#).

## 7.12 AveragePredictor.h File Reference

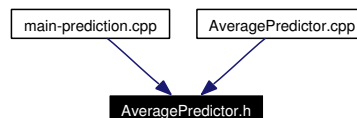
A simple Predictor using averages (median for ordinal time series, most frequent for categorical/nominal) as predicted next values.

```
#include "../Predictor.h"
```

Include dependency graph for AveragePredictor.h:



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



### Classes

- class [AveragePredictor](#)

### 7.12.1 Detailed Description

A simple Predictor using averages (median for ordinal time series, most frequent for categorical/nominal) as predicted next values.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/23 11:25:16

#### Revision

1.5

#### Since:

Wed Mar 19 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

**Id**

[AveragePredictor.h](#),v 1.5 2004/02/23 11:25:16 rene Exp

Definition in file [AveragePredictor.h](#).

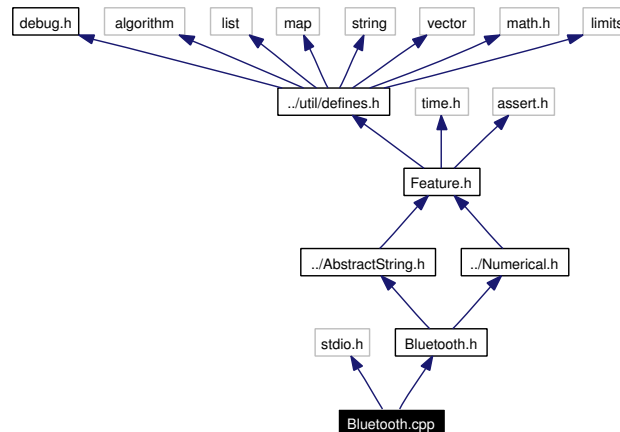
## 7.13 Bluetooth.cpp File Reference

Bluetooth feature.

```
#include <stdio.h>
```

```
#include "Bluetooth.h"
```

Include dependency graph for Bluetooth.cpp:



### 7.13.1 Detailed Description

Bluetooth feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/02 12:36:18

#### Revision

1.27

#### Since:

Tue Mar 18 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[Bluetooth.cpp](#),v 1.27 2004/03/02 12:36:18 rene-cvs Exp

Definition in file [Bluetooth.cpp](#).

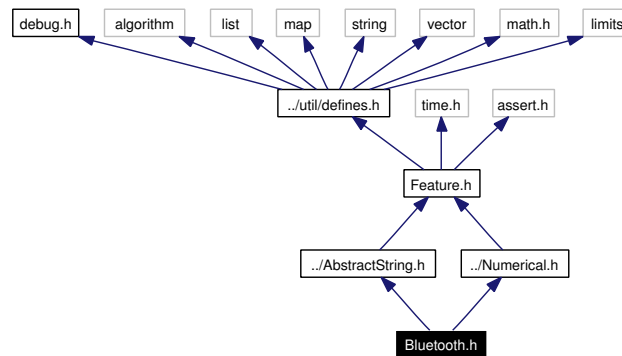
## 7.14 Bluetooth.h File Reference

Bluetooth feature.

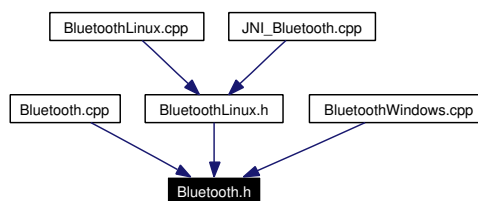
```
#include "../AbstractString.h"
```

```
#include "../Numerical.h"
```

Include dependency graph for Bluetooth.h:



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



### Classes

- class [BluetoothPeersFeature](#)  
*List of peers in range.*
- class [BluetoothNumPeersFeature](#)  
*Number of peers in range.*
- class [BluetoothFeatureProvider](#)  
*Provider for the Bluetooth features.*

### 7.14.1 Detailed Description

Bluetooth feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/12 20:08:37

**Revision**

1.24

**Since:**

Tue Mar 18 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

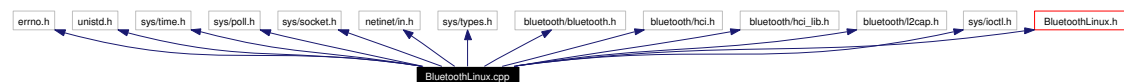
**Id**[Bluetooth.h](#),v 1.24 2004/02/12 20:08:37 harald ExpDefinition in file [Bluetooth.h](#).

## 7.15 BluetoothLinux.cpp File Reference

Bluetooth feature.

```
#include <errno.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/poll.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
#include <bluetooth/hci_lib.h>
#include <bluetooth/l2cap.h>
#include <sys/ioctl.h>
#include "BluetoothLinux.h"
```

Include dependency graph for BluetoothLinux.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) (providerparams &params)  
*\ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param params A map of parameters to initialize the instance \ return A [FeatureProvider](#) implementation \*
- float [tv2fl](#) (struct timeval tv)  
*a helper function for caclulating the time difference*
- int [find\\_conn](#) (int s, int dev\_id, long arg)  
*a helper function copied from hcitool.c*

### Variables

- [BluetoothLinuxFeatureProvider](#) \* fp

*Singleton implementing the [FeatureProvider](#) interface.*

### 7.15.1 Detailed Description

Bluetooth feature.

Some parts of this code have been taken from hcitool.c of the bluez-utils distribution and were modified accordingly. The original license is:

BlueZ - Bluetooth protocol stack for Linux Copyright (C) 2000-2001 Qualcomm Incorporated

Written 2000,2001 by Maxim Krasnyansky <[maxk@qualcomm.com](mailto:maxk@qualcomm.com)>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation;

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER(S) AND AUTHOR(S) BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS, RELATING TO USE OF THIS SOFTWARE IS DISCLAIMED.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/23 11:26:42

**Revision**

1.26

**Since:**

Tue Mar 18 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[BluetoothLinux.cpp](#),v 1.26 2004/02/23 11:26:42 rene Exp

Definition in file [BluetoothLinux.cpp](#).



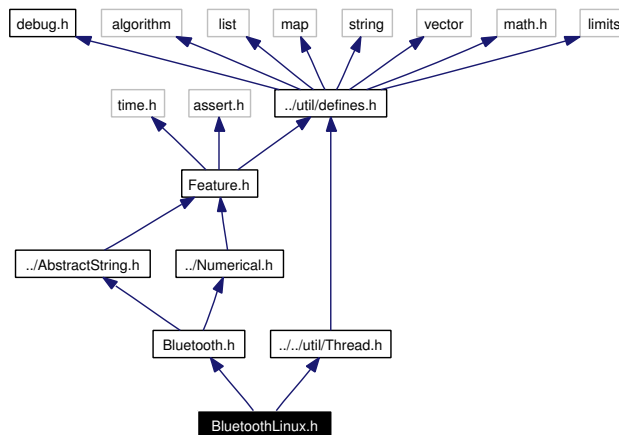
## 7.16 BluetoothLinux.h File Reference

Bluetooth feature.

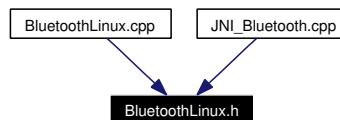
```
#include "Bluetooth.h"
```

```
#include "../../util/Thread.h"
```

Include dependency graph for BluetoothLinux.h:



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



### Classes

- class [BluetoothLinuxFeatureProvider](#)
- class [BluetoothLinuxFeatureProvider::L2Connection](#)

### 7.16.1 Detailed Description

Bluetooth feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/23 11:26:42

#### Revision

1.15

#### Since:

Tue Mar 18 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

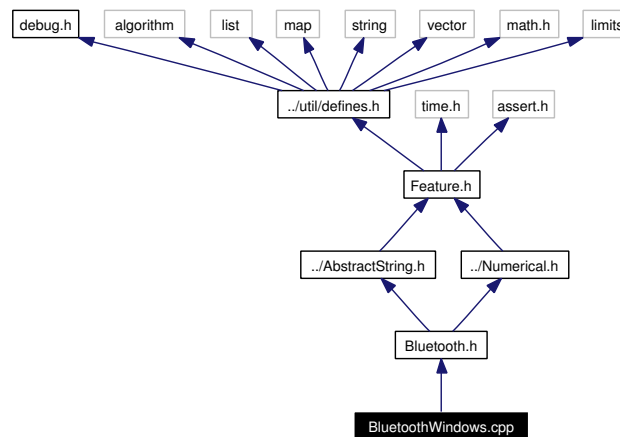
**Id**[BluetoothLinux.h](#),v 1.15 2004/02/23 11:26:42 rene ExpDefinition in file [BluetoothLinux.h](#).

## 7.17 BluetoothWindows.cpp File Reference

Bluetooth feature.

```
#include "Bluetooth.h"
```

Include dependency graph for BluetoothWindows.cpp:



### Driver Functions

API functions of the Digianswer Bluetooth driver.

- void(\* [BtInit](#) )()  
*Initialize the Bluetooth stack.*
- void(\* [BtStart](#) )()  
*Start inquiry.*
- void(\* [BtStop](#) )()  
*Stop inquiry.*
- char \*\*(\* [BtGetHardwareAddressesInRange](#) )()  
*Get hardware addresses in range.*
- void(\* [BtFreeHardwareAddressesList](#) )(char \*\*addrList)  
*Free the array of strings.*

### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*

- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) ([providerparams](#) &params)  
 \ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param params A map of parameters to initialize the instance \ return A [FeatureProvider](#) implementation \
- BOOL APIENTRY [DllMain](#) (HANDLE hModule, DWORD fdwReason, LPVOID lpReserved)  
*Library entry point.*

## Variables

- HMODULE [dll](#) = NULL  
*Handle of the Bluetooth driver library.*
- [BluetoothFeatureProvider](#) \* [fp](#)  
*Singleton implementing the [FeatureProvider](#) interface.*

### 7.17.1 Detailed Description

Bluetooth feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.13

#### Since:

Tue Mar 18 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>  
Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[BluetoothWindows.cpp](#),v 1.13 2004/02/12 20:08:37 harald Exp

Definition in file [BluetoothWindows.cpp](#).

### 7.17.2 Variable Documentation

#### 7.17.2.1 void(\* [BtFreeHardwareAddressesList](#))(char \*\*[addrList](#)) [static]

Free the array of strings.

**Parameters:**

*addrList* The array of strings

Definition at line 60 of file BluetoothWindows.cpp.

Referenced by DllMain().

**7.17.2.2 char\*\*(\* BtGetHardwareAddressesInRange)() [static]**

Get hardware addresses in range.

**Returns:**

An array of strings containing the hardware addresses

Definition at line 53 of file BluetoothWindows.cpp.

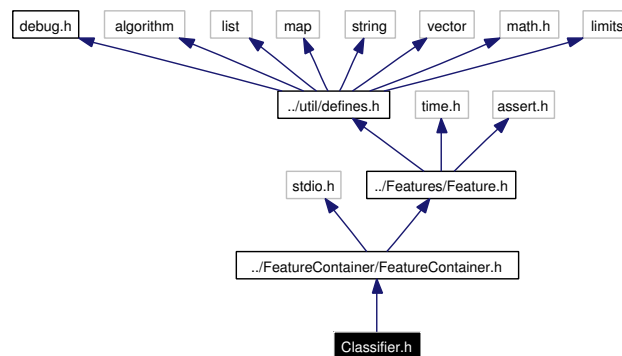
Referenced by DllMain().

## 7.18 Classifier.h File Reference

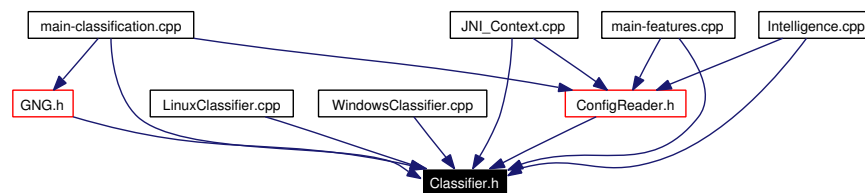
Classifier interface

```
#include "../FeatureContainer/FeatureContainer.h"
```

Include dependency graph for Classifier.h:



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



### Classes

- class [ClassifierAlgorithm](#)  
*ClassifierAlgorithm Interface.*
- class [Classifier](#)  
*Classifier Wrapper.*

### Defines

- #define [EXPORT\\_CLASSIFIER](#)(classifier)  
*Export a classifier.*

### Typedefs

- typedef [parametermap](#) classifierparams

*Parameters passed to classifiers.*

- typedef [ClassifierAlgorithm](#) \*(\* [getClassifier\\_t](#))(const [classifierparams](#) &params)

*Type of the [getClassifier](#) function.*

## 7.18.1 Detailed Description

Classifier interface

©2003-2004 by Rene Mayrhofer, Harald Radi

### Date

2004/03/02 13:02:00

### Revision

1.20

### Since:

Wed Mar 19 2003

### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

### Id

[Classifier.h](#),v 1.20 2004/03/02 13:02:00 rene-cvs Exp

Definition in file [Classifier.h](#).

## 7.18.2 Define Documentation

### 7.18.2.1 #define EXPORT\_CLASSIFIER(classifier)

#### Value:

```
\
static classifier *clf; \
void INIT_EXPORT library_initialize() { clf = NULL; } \
void FINI_EXPORT library_finalize() { if (clf != NULL) delete clf; } \ \
extern "C" CLASSIFIER_EXPORT ClassifierAlgorithm* getClassifier(classifierparams &params) { if (clf ==
```

Export a classifier.

This macro has to be used in global scope and gets substituted with the implementation for exporting a [ClassifierAlgorithm](#) singleton instance to the framework.

#### Parameters:

*classifier* The Class that implements the [ClassifierAlgorithm](#) interface and should be exported to the framework.

Definition at line 52 of file [Classifier.h](#).

### 7.18.3 Typedef Documentation

#### 7.18.3.1 typedef [parametermap](#) [classifierparams](#)

Parameters passed to classifiers.

When initializing a classifier, an arbitrary number of named (string) parameters can be passed from the application (e.g. read from a config file). These key-value tuples can then be used by the classifier as configuration or initialization values. They are completely classifier specific.

See also:

[Classifier](#)

[ClassifierAlgorithm](#)

Definition at line 72 of file Classifier.h.

Referenced by [Classifier::Classifier\(\)](#).

#### 7.18.3.2 typedef [ClassifierAlgorithm](#)\*([getClassifier\\_t](#))(const [classifierparams](#) &params)

Type of the [getClassifier](#) function.

This type definition is needed by the [Classifier](#) Wrapper to query a [ClassifierAlgorithm](#) implementation for an instance of its class. Definition at line 214 of file Classifier.h.

Referenced by [Classifier::Classifier\(\)](#).



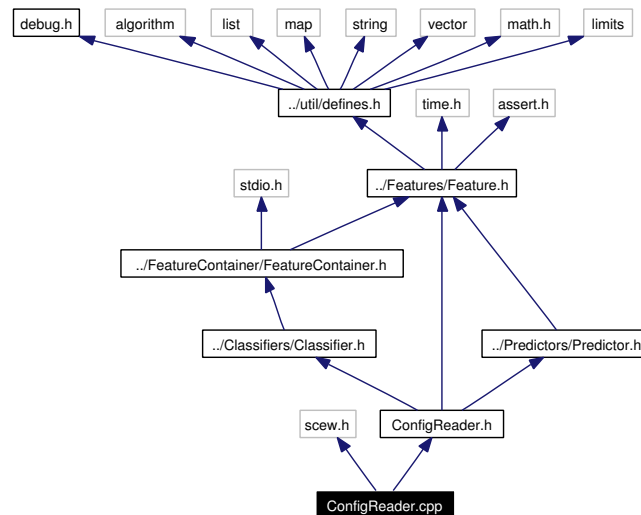
## 7.19 ConfigReader.cpp File Reference

Configuration parser.

```
#include <scew.h>
```

```
#include "ConfigReader.h"
```

Include dependency graph for ConfigReader.cpp:



### 7.19.1 Detailed Description

Configuration parser.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/02 12:42:25

#### Revision

1.20

#### Since:

Mon Aug 11 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[ConfigReader.cpp](#),v 1.20 2004/03/02 12:42:25 rene-cvs Exp

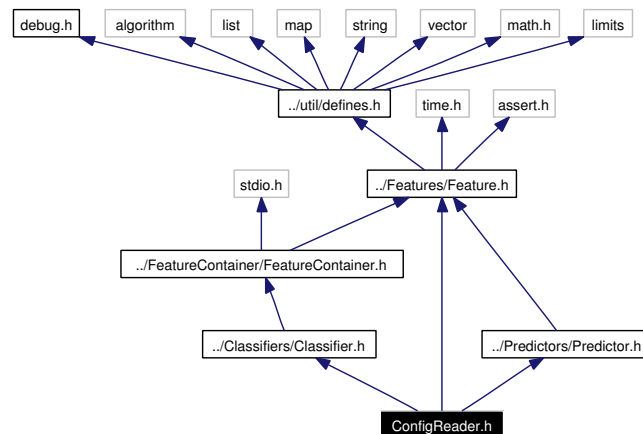
Definition in file [ConfigReader.cpp](#).

## 7.20 ConfigReader.h File Reference

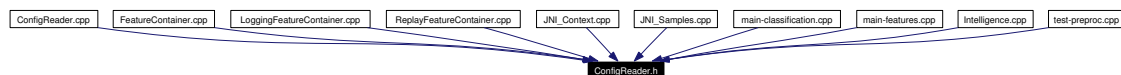
Configuration parser.

```
#include "../Features/Feature.h"
#include "../Classifiers/Classifier.h"
#include "../Predictors/Predictor.h"
```

Include dependency graph for ConfigReader.h:



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



### Classes

- class [ConfigReader](#)  
*Configuration parser.*

### 7.20.1 Detailed Description

Configuration parser.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.13

#### Since:

Mon Aug 11 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[ConfigReader.h](#),v 1.13 2004/02/12 20:08:37 harald Exp

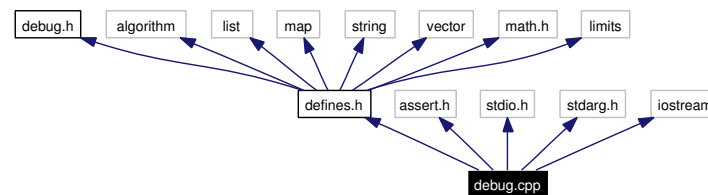
Definition in file [ConfigReader.h](#).

## 7.21 debug.cpp File Reference

Debug log implementation.

```
#include "defines.h"
#include "assert.h"
#include <stdio.h>
#include <stdarg.h>
#include <iostream>
```

Include dependency graph for debug.cpp:



### Typedefs

- typedef void(\* [log\\_t](#))(const char \*format, va\_list args)  
*Function type definitions for logging.*
- typedef void(\* [init\\_log\\_t](#))([log\\_t](#) log)  
*Function type definitions for initializing logging.*

### Functions

- void [\\_open\\_log](#) (const char \*path, bool console)  
*Initialize the logfile.*
- void [\\_close\\_log](#) ()  
*Close the logfile.*
- void [\\_dll\\_init\\_log](#) (void \*dlHandle)  
*Loads the main logging module into dynamically loaded libraries to ensure that logging is performed synchronously into a single logfile.*
- void [init\\_log](#) ([log\\_t](#) log)  
*Exported funtion to initialize logging.*
- void [\\_log](#) (const char \*format,...)  
*Wrapper function to convert printf style parameters into va\_list style parameters.*
- void [\\_logfoo](#) (const char \*,...)  
*dummy function if logging is disabled*

## Variables

- bool [con](#)  
*Flag, signaling whether output should be printed to the console or not.*
- FILE \* [logfile](#) = NULL  
*Logfile file handle.*
- [log\\_t log\\_cb](#) = \_\_logfoo  
*Globally used logging function set by the `init_log` function.*

### 7.21.1 Detailed Description

Debug log implementation.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:39

#### Revision

1.15

#### Since:

Sat Aug 30 2003

#### Author:

Rene Mayrhofer <[reene@mayrhofer.eu.org](mailto:reene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[debug.cpp](#),v 1.15 2004/02/12 20:08:39 harald Exp

Definition in file [debug.cpp](#).

### 7.21.2 Function Documentation

#### 7.21.2.1 void \_dll\_init\_log (void \* *dll*)

Loads the main logging module into dynamically loaded libraries to ensure that logging is performed synchronously into a single logfile.

#### Parameters:

*dll* Handle to the dynamically loaded library

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

References `__log()`, and `init_log_t`.

### 7.21.2.2 void \_log (const char \**format*, ...)

Wrapper function to convert printf style parameters into va\_list style parameters.

**Parameters:**

*format* printf style format string  
... Additional parameters

Definition at line 146 of file debug.cpp.

References log\_cb.

### 7.21.2.3 void \_logfoo (const char \**format*, ...)

dummy function if logging is disabled

**See also:**

[\\_log](#)

**Todo**

gcc supports 'args...' macros, afaik ms doesn't - check, would be a nicer hack

Definition at line 202 of file debug.cpp.

### 7.21.2.4 void \_open\_log (const char \**path*, bool *console*)

Initialize the logfile.

**Parameters:**

*path* Filename of the logfile or *NULL* if no logfile should be written.  
*console* *true* if the log should be printed to the console, *false* otherwise.

**Remarks:**

Not every platform provides a console, writing a logfile is the only possibility of logging information there.

Definition at line 69 of file debug.cpp.

References \_\_log(), con, log\_cb, and logfile.

### 7.21.2.5 void init\_log ([log\\_t](#) *log*)

Exported funtion to initialize logging.

**Parameters:**

*log* Logging function

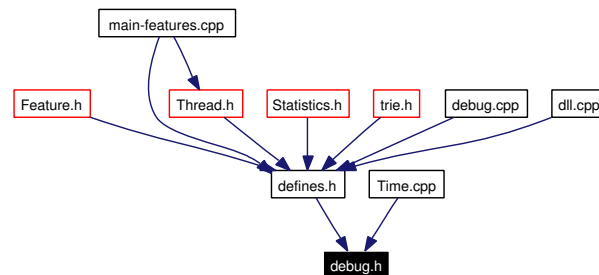
Definition at line 141 of file debug.cpp.

References log\_cb, and log\_t.

## 7.22 debug.h File Reference

Debug log declaration.

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



### Logging macros

Macros encapsulating the logging functions for easier use

- `#define open_log(path)`
- `#define close_log()`
- `#define dll_init_log(x)`
- `#define plog_logfoo`

### Functions

- `void _open_log (const char *path, bool console)`  
*Initialize the logfile.*
- `void _close_log ()`  
*Close the logfile.*
- `void _log (const char *format,...)`  
*Wrapper function to convert printf style parameters into va\_list style parameters.*
- `void _logfoo (const char *format,...)`  
*dummy function if logging is disabled*
- `void _dll_init_log (void *dll)`  
*Loads the main logging module into dynamically loaded libraries to ensure that logging is performed synchronously into a single logfile.*

#### 7.22.1 Detailed Description

Debug log declaration.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/12 20:08:39

**Revision**

1.4

**Since:**

Sat Aug 30 2003

**Author:**Rene Mayrhofer <[reene@mayrhofer.eu.org](mailto:reene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**[debug.h](#),v 1.4 2004/02/12 20:08:39 harald ExpDefinition in file [debug.h](#).

## 7.22.2 Function Documentation

### 7.22.2.1 void \_dll\_init\_log (void \* *dll*)

Loads the main logging module into dynamically loaded libraries to ensure that logging is performed synchronously into a single logfile.

**Parameters:***dll* Handle to the dynamically loaded library

Definition at line 127 of file debug.cpp.

References `__log()`, and `init_log_t`.

### 7.22.2.2 void \_log (const char \* *format*, ...)

Wrapper function to convert printf style parameters into va\_list style parameters.

**Parameters:***format* printf style format string

... Additional parameters

Definition at line 146 of file debug.cpp.

References `log_cb`.



### 7.22.2.3 void \_logfoo (const char \**format*, ...)

dummy function if logging is disabled

See also:

[\\_log](#)

#### Todo

gcc supports 'args...' macros, afaik ms doesn't - check, would be a nicer hack

Definition at line 202 of file debug.cpp.

### 7.22.2.4 void \_open\_log (const char \**path*, bool *console*)

Initialize the logfile.

#### Parameters:

*path* Filename of the logfile or *NULL* if no logfile should be written.

*console* *true* if the log should be printed to the console, *false* otherwise.

#### Remarks:

Not every platform provides a console, writing a logfile is the only possibility of logging information there.

Definition at line 69 of file debug.cpp.

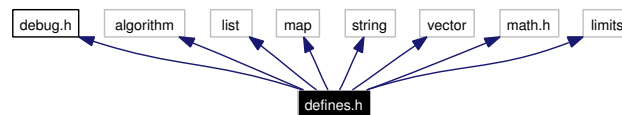
References `__log()`, `con`, `log_cb`, and logfile.

## 7.23 defines.h File Reference

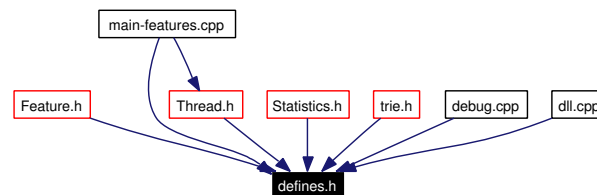
Global defines.

```
#include "debug.h"
#include <algorithm>
#include <list>
#include <map>
#include <string>
#include <vector>
#include <math.h>
#include <limits>
```

Include dependency graph for defines.h:



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



### Debugging Defines

Macros for enabling and disabling debug output

- `#define _DEBUG_CLASSIFIERS 1`
- `#define _DEBUG_FEATURES 1`
- `#define _DEBUG_PREDICTORS 1`

### Optimisation Defines

Macros for enabling and disabling various optimizations

- `#define NO_OPTIMIZATIONS 0`  
*turn off all optimizations (for debugging)*
- `#define OPTIMIZATIONS_SPLAYTREE 0`

*turn on splaytree optimization*

- #define [OPTIMIZATIONS\\_CACHE\\_SIMILARITY](#) 1  
*turn on cache similarity optimization*
- #define [OPTIMIZATIONS\\_SPLIT\\_MERGE](#) 0  
*turn on split/merge optimization*

## Global Typedefs

- typedef [vector](#)< string > [stringvector](#)  
*A simple ordered list of strings.*
- typedef [map](#)< string, string > [parametermap](#)  
*A map to represent configuration file entries.*

## Defines

- #define [NOMINMAX](#)  
*Prevent the MIN() and MAX() macros from being defined.*
- #define [DUMP\\_LEAKS](#)()  
*Hint the debug heap to dump all memory blocks not freed so far.*
- #define [rand\\_double](#)() ((double) rand() / (RAND\_MAX + 0.00001))  
*Get a random value.*
- #define [INIT\\_EXPORT](#)  
*Compiler attributes for library initialization.*
- #define [FINI\\_EXPORT](#)  
*Compiler attributes for library cleanup.*

### 7.23.1 Detailed Description

Global defines.

This file contains often used defines and includes and contains macros to have a platform and compiler independent way of accessing basic functions.

#### Note:

STL includes should only be done in this file to better cope with the different STL implementations.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/02 12:41:03

**Revision**

1.39

**Since:**

Sun Jul 13 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**[defines.h](#),v 1.39 2004/03/02 12:41:03 rene-cvs ExpDefinition in file [defines.h](#).

## 7.23.2 Define Documentation

### 7.23.2.1 `#define rand_double() ((double) rand() / (RAND_MAX + 0.00001))`

Get a random value.

This macro returns a random value within the interval [0; 1[

**Returns:**

Random value

Definition at line 56 of file [defines.h](#).

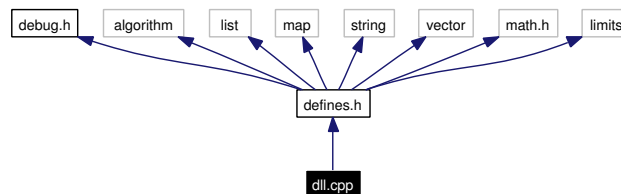
Referenced by `AbstractStringFeature::AbstractStringFeature()`, `AbstractStringListFeature::AbstractStringListFeature()`, `evaluate()`, `AbstractStringFeature::levenshtein()`, `WlanActiveModeFeature::moveTowards()`, `AbstractStringListFeature::moveTowards()`, `AbstractStringFeature::moveTowards()`, `NumericalContinuousFeature::NumericalContinuousFeature()`, `NumericalDiscreteFeature::NumericalDiscreteFeature()`, and `WlanActiveModeFeature::WlanActiveModeFeature()`.

## 7.24 dll.cpp File Reference

DLL wrapper.

```
#include "defines.h"
```

Include dependency graph for dll.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*

#### 7.24.1 Detailed Description

DLL wrapper.

This file wraps the [library\\_initialize\(\)](#) and [library\\_finalize\(\)](#) functions for every operating system.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/01/15 13:06:35

#### Revision

1.2

#### Since:

Thu Jan 15 2004

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[dll.cpp](#),v 1.2 2004/01/15 13:06:35 harald Exp

Definition in file [dll.cpp](#).

## 7.25 doxygen.h File Reference

Doxygen documentation file.

### Namespaces

- namespace **std**

### Classes

- class [std::list< element >](#)  
*STL list template.*
- class [std::map< key, value >](#)  
*STL map template.*
- class [std::pair< first, second >](#)  
*STL pair template.*
- class [std::vector< element >](#)  
*STL vector template.*

### 7.25.1 Detailed Description

Doxygen documentation file.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.6

#### Since:

Wed Jan 14 2004

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>  
Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[doxygen.h](#),v 1.6 2004/02/12 20:08:37 harald Exp

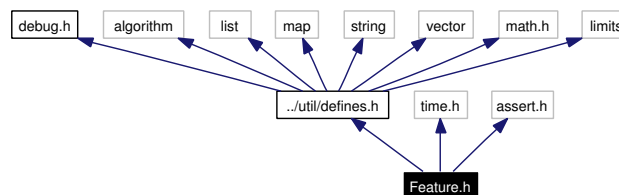
Definition in file [doxygen.h](#).

## 7.26 Feature.h File Reference

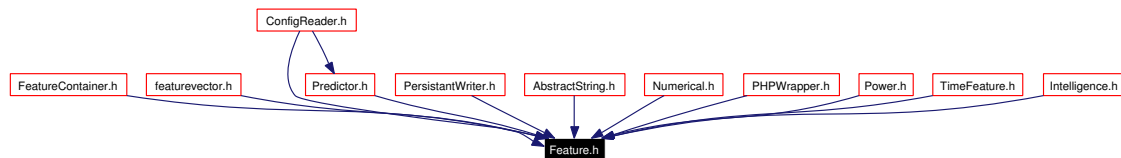
Feature interface declaration

```
#include "../util/defines.h"
#include <time.h>
#include <assert.h>
```

Include dependency graph for Feature.h:



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



### Classes

- class [Feature](#)  
*Feature interface.*
- class [PersistantFeature](#)  
*PersistantFeature interface.*
- class [FeatureProvider](#)  
*FeatureProvider interface.*

### DLL initialization functions

These functions will be called when a library is loaded or unloaded by the framework.

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*

## Defines

- #define `EXPORT_FEATURE(provider)`  
*Export a feature provider.*

## Typedefs

- typedef `parametermap providerparams`  
*Parameters passed to feature providers.*
- typedef `parametermap featureparams`
- typedef `pair< time_t, Feature * > aggregatefeature`  
*A <timestamp, Feature> tuple.*
- typedef `list< aggregatefeature > aggregatelist`  
*A list of <timestamp, Feature> tuples.*
- typedef `map< unsigned long, double > membershiplist`  
*A map type for returning membership values for each cluster.*
- typedef `vector< Feature * > featurevector`  
*Ordered list of features.*
- typedef `FeatureProvider *(* getProvider_t )(const providerparams &params)`  
*Type of the getProvider function.*

### 7.26.1 Detailed Description

Feature interface declaration

This file contains the interfaces that have to be implemented to enable a sensor to be used by the framework.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.56

#### Since:

Sun Mar 30 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**



**Id**

[Feature.h](#), v 1.56 2004/02/12 20:08:37 harald Exp

Definition in file [Feature.h](#).

**7.26.2 Define Documentation****7.26.2.1 #define EXPORT\_FEATURE(provider)****Value:**

```
\
static provider *fp; \
void INIT_EXPORT library_initialize() { fp = NULL; } \
void FINI_EXPORT library_finalize() { if (fp != NULL) delete fp; } \ \
extern "C" FEATURE_EXPORT FeatureProvider* getProvider(providerparams &params) { if (fp == NULL) fp =
```

Export a feature provider.

This macro has to be used in global scope and gets substituted with the implementation for exporting a [FeatureProvider](#) singleton instance to the framework.

**Parameters:**

*provider* The Class that implements the [FeatureProvider](#) interface and should be exported to the framework.

Definition at line 94 of file [Feature.h](#).

**7.26.3 Typedef Documentation****7.26.3.1 typedef [parametermap](#) [featureparams](#)****Todo**

documentation

Definition at line 122 of file [Feature.h](#).

Referenced by [FeatureContainer::FeatureContainer\(\)](#), [GSMFeatureProvider::init\(\)](#), [Java\\_at\\_jku\\_-bluetooth\\_LinuxBluetoothAdapter\\_nativeInit\(\)](#), [library\\_initialize\(\)](#), [Main\(\)](#), [BluetoothFeatureProvider::nextSample\(\)](#), [NumericalContinuousFeature::read\(\)](#), [NumericalDiscreteFeature::read\(\)](#), [AbstractStringFeature::read\(\)](#), [SystemCommandStringListFeatureProvider::SystemCommandStringListFeatureProvider\(\)](#), [NumericalContinuousFeature::write\(\)](#), [NumericalDiscreteFeature::write\(\)](#), and [AbstractStringFeature::write\(\)](#).

**7.26.3.2 typedef [vector](#)<[Feature\\*](#)> [featurevector](#)**

Ordered list of features.

This is one of the basic datatypes in this framework. It represents an ordered list of features, which describes a single point in the heterogeneous feature space.

**Note:**

The elements of this vector are pointers to feature objects. Therefore, the memory management of its members is not defined globally but depends on the method which returns or creates a [featurevector](#).

Definition at line 396 of file Feature.h.

Referenced by evaluate(), Classifier::getCa(), Unit::getDistance(), FeatureContainer::getFeatureVector(), Unit::getPosition(), ReplayFeatureContainer::getSampleVector(), FeatureContainer::getSampleVector(), GNG::getWinnerDistance(), GNG::init(), Java\_at\_jku\_intelligence\_samples\_SampleContainer\_nativeGetSampleVector(), Main(), Unit::moveTowards(), ReplayFeatureContainer::nextSample(), LoggingFeatureContainer::nextSample(), GNG::nextSample(), Scanner::run(), serializeFeatureVector(), Unit::splitUnit(), Unit::Unit(), Unit::unserialize(), unserializeFeatureVector(), and Unit::updateErrors().

### 7.26.3.3 typedef [FeatureProvider](#)\*([\\* getProvider\\_t](#))(const [providerparams](#) &params)

Type of the getProvider function.

This type definition is needed by the [FeatureContainer](#) to query a [Feature](#) implementation for an instance of its class. Definition at line 509 of file Feature.h.

Referenced by FeatureContainer::loadProvider().

### 7.26.3.4 typedef [parametermap](#) [providerparams](#)

Parameters passed to feature providers.

When initializing a feature provider, an arbitrary number of named (string) parameters can be passed from the application (e.g. read from a config file). These key-value tuples can then be used by the feature provider as configuration or initialization values. They are completely [FeatureProvider](#) specific.

The same structure is also used to store and read persistent feature data.

See also:

[FeatureProvider](#)

[PersistentFeature](#)

Definition at line 119 of file Feature.h.

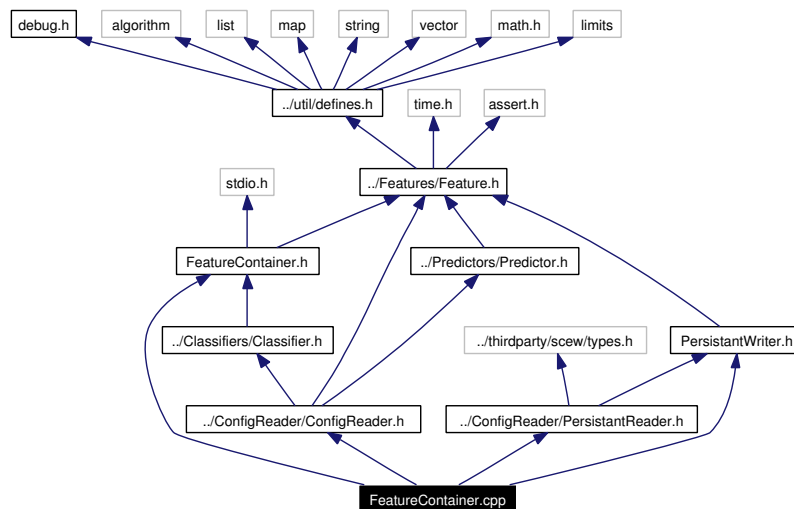
Referenced by PowerFeature::aggregate(), BluetoothFeatureProvider::BluetoothFeatureProvider(), library\_initialize(), FeatureContainer::loadProvider(), VideoFeatureProvider::nextSample(), and WlanFeatureProvider::WlanFeatureProvider().

## 7.27 FeatureContainer.cpp File Reference

Feature container

```
#include "FeatureContainer.h"
#include "../ConfigReader/ConfigReader.h"
#include "../ConfigReader/PersistentReader.h"
#include "../ConfigReader/PersistentWriter.h"
```

Include dependency graph for FeatureContainer.cpp:



### 7.27.1 Detailed Description

Feature container

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.43

#### Since:

Sun Apr 6 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

**Id**

[FeatureContainer.cpp](#), v 1.43 2004/02/12 20:08:37 harald Exp

Definition in file [FeatureContainer.cpp](#).

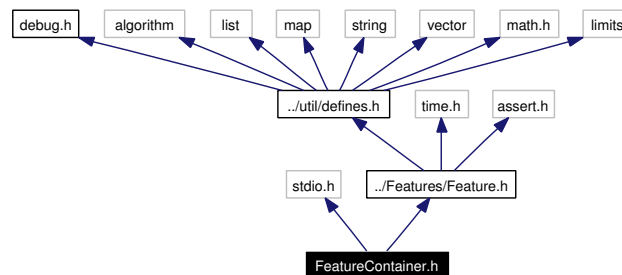
## 7.28 FeatureContainer.h File Reference

Feature container

```
#include <stdio.h>
```

```
#include "../Features/Feature.h"
```

Include dependency graph for FeatureContainer.h:



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



### Classes

- class [FeatureContainer](#)

*Container class.*

### 7.28.1 Detailed Description

Feature container

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/02 12:36:47

#### Revision

1.22

#### Since:

Sun Apr 6 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[FeatureContainer.h](#),v 1.22 2004/03/02 12:36:47 rene-cvs Exp

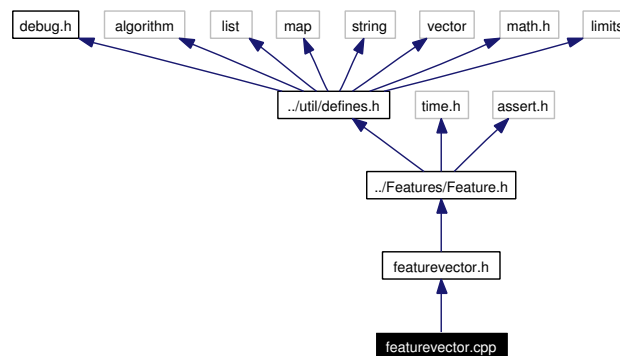
Definition in file [FeatureContainer.h](#).

## 7.29 featurevector.cpp File Reference

Feature helper functions

```
#include "featurevector.h"
```

Include dependency graph for featurevector.cpp:



### Functions

- char \* [strsplit](#) (char \*str, const char sep)

*A very short helper for replacing the brain-damaged strtok (which can't handle empty tokens).*

- string [serializeFeatureVector](#) (const [featurevector](#) &samples)

*These are just two helper functions for serializing and deserializing a whole feature vector just returns, for a given feature vector, the serialized, correctly escaped form with ';' as delimiter.*

- size\_t [unserializeFeatureVector](#) (const [featurevector](#) &features, [featurevector](#) &samples, char \*data)

*parameters: features must contain all \_features\_ that will be unserialized from data, while samples must be empty as this method will fill it with newly created [Feature](#) objects of the correct type (by cloning them from the objects in features) beware that data will be changed by this method ! returns the number of characters processed from dat*

### 7.29.1 Detailed Description

Feature helper functions

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/12 10:55:21

#### Revision

1.9

#### Since:

Thu Aug 28 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[featurevector.cpp](#),v 1.9 2004/03/12 10:55:21 rene Exp

Definition in file [featurevector.cpp](#).

## 7.29.2 Function Documentation

### 7.29.2.1 `char* strsplit (char * str, const char sep)`

A very short helper for replacing the brain-damaged strtok (which can't handle empty tokens).

**Parameters:**

*str* String to be tokenized, *NULL* on subsequent calls

*sep* Separator

**Returns:**

A valid token on success, *NULL* if the string can't be tokenized

**Remarks:**

This function alters the string passed as parameter *str*.

Definition at line 26 of file [featurevector.cpp](#).

Referenced by `Unit::unserialize()`, and `unserializeFeatureVector()`.

### 7.29.2.2 `size_t unserializeFeatureVector (const featurevector & features, featurevector & samples, char * data)`

parameters: *features* must contain all `_features_` that will be unserialized from *data*, while *samples* must be empty as this method will fill it with newly created [Feature](#) objects of the correct type (by cloning them from the objects in *features*) beware that *data* will be changed by this method ! returns the number of characters processed from *data*

**Todo**

catch "[invalid]" entries in the file and de-escape ";", "\", "[" and "]"

Definition at line 89 of file [featurevector.cpp](#).

References [featurevector](#), [strsplit\(\)](#), and `Feature::unserialize()`.

Referenced by `Unit::unserialize()`.

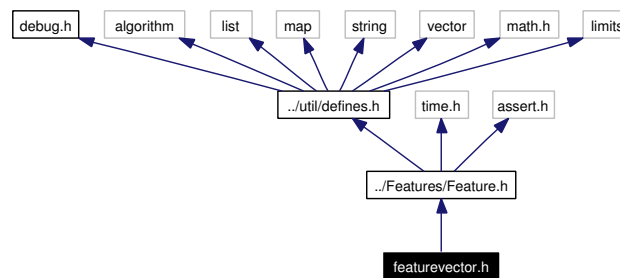


## 7.30 featurevector.h File Reference

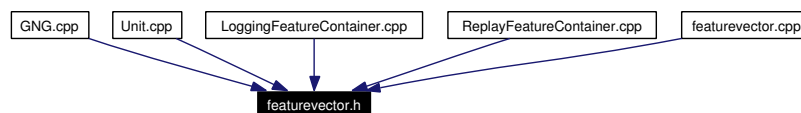
Feature helper function declarations

```
#include "../Features/Feature.h"
```

Include dependency graph for featurevector.h:



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



### Functions

- `char * strsplit (char *str, const char sep)`

*A very short helper for replacing the brain-damaged `strtok` (which can't handle empty tokens).*

- `string serializeFeatureVector (const featurevector &samples)`

*These are just two helper functions for serializing and deserializing a whole feature vector just returns, for a given feature vector, the serialized, correctly escaped form with ';' as delimiter.*

- `size_t unserializeFeatureVector (const featurevector &features, featurevector &samples, char *data)`

*parameters: features must contain all `_features_` that will be unserialized from data, while samples must be empty as this method will fill it with newly created [Feature](#) objects of the correct type (by cloning them from the objects in features) beware that data will be changed by this method ! returns the number of characters processed from dat*

### 7.30.1 Detailed Description

Feature helper function declarations

©2003-2004 by Rene Mayrhofer, Harald Radi

Date

2004/02/23 11:24:40

**Revision**

1.5

**Since:**

Thu Aug 28 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**[featurevector.h](#),v 1.5 2004/02/23 11:24:40 rene ExpDefinition in file [featurevector.h](#).

## 7.30.2 Function Documentation

### 7.30.2.1 char\* strsplit (char \* *str*, const char *sep*)

A very short helper for replacing the brain-damaged strtok (which can't handle empty tokens).

**Parameters:***str* String to be tokenized, *NULL* on subsequent calls*sep* Separator**Returns:**A valid token on success, *NULL* if the string can't be tokenized**Remarks:**This function alters the string passed as parameter *str*.Definition at line 26 of file [featurevector.cpp](#).Referenced by [Unit::unserialize\(\)](#), and [unserializeFeatureVector\(\)](#).

### 7.30.2.2 size\_t unserializeFeatureVector (const [featurevector](#) & *features*, [featurevector](#) & *samples*, char \* *data*)

parameters: *features* must contain all `_features_` that will be unserialized from *data*, while *samples* must be empty as this method will fill it with newly created [Feature](#) objects of the correct type (by cloning them from the objects in *features*) beware that *data* will be changed by this method ! returns the number of characters processed from *data*

**Todo**

catch "[invalid]" entries in the file and de-escape ";", "\", "[" and "]"

Definition at line 89 of file [featurevector.cpp](#).References [featurevector](#), [strsplit\(\)](#), and [Feature::unserialize\(\)](#).Referenced by [Unit::unserialize\(\)](#).

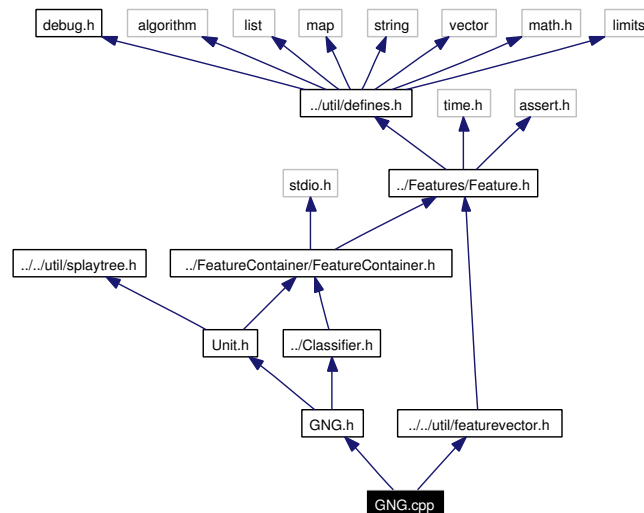
## 7.31 GNG.cpp File Reference

Growing neural gas.

```
#include "GNG.h"
```

```
#include "../util/featurevector.h"
```

Include dependency graph for GNG.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [ClassifierAlgorithm](#) \* [getClassifier](#) ([classifierparams](#) &params)  
*\ Returns an instance of a class implementing the [ClassifierAlgorithm](#) interface \ \ param params A map of parameters to initialize the instance \ return A [ClassifierAlgorithm](#) implementation \*

### Variables

- [GNG](#) \* [clf](#)  
*Singleton implementing the [ClassifierAlgorithm](#) interface.*

#### 7.31.1 Detailed Description

Growing neural gas.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/03/19 13:19:26

**Revision**

1.47

**Since:**

Mon Mar 3 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**[GNG.cpp](#),v 1.47 2004/03/19 13:19:26 rene ExpDefinition in file [GNG.cpp](#).

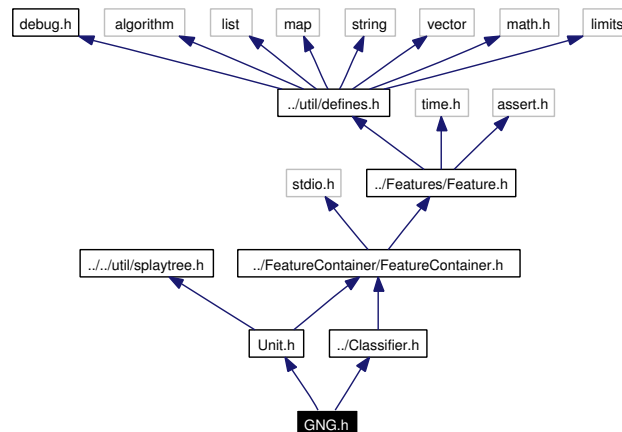
## 7.32 GNG.h File Reference

Growing neural gas.

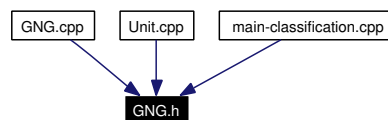
```
#include "../Classifier.h"
```

```
#include "Unit.h"
```

Include dependency graph for GNG.h:



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



### Classes

- struct [meta\\_cluster](#)
- class [GNG](#)

*This is lifelong GNG.*

### Typedefs

- typedef [list](#)< [meta\\_cluster](#) > [meta\\_cluster\\_list](#)

### 7.32.1 Detailed Description

Growing neural gas.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/03/02 12:37:23

**Revision**

1.31

**Since:**

Mon Mar 3 2003

**Author:**Rene Mayrhofer <[reene@mayrhofer.eu.org](mailto:reene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**[GNG.h](#),v 1.31 2004/03/02 12:37:23 rene-cvs ExpDefinition in file [GNG.h](#).

## 7.32.2 Typedef Documentation

### 7.32.2.1 typedef [list](#)<[meta\\_cluster](#)> [meta\\_cluster\\_list](#)

**Todo**

documentation

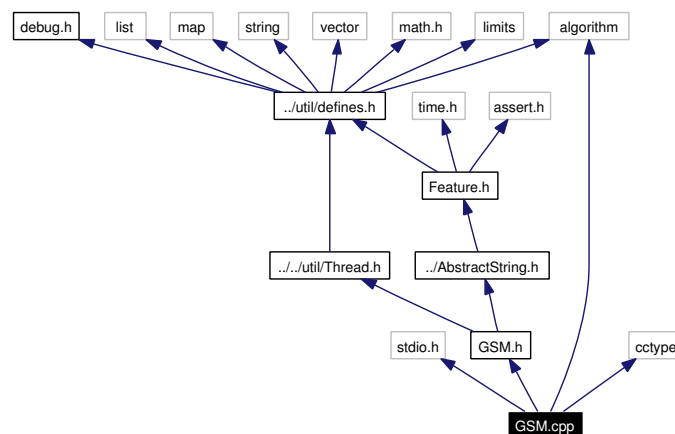
Definition at line 45 of file [GNG.h](#).Referenced by [GNG::getNumberNodes\(\)](#), and [Scanner::run\(\)](#).

## 7.33 GSM.cpp File Reference

GSM feature.

```
#include <stdio.h>
#include "GSM.h"
#include <algorithm>
#include <cctype>
```

Include dependency graph for GSM.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) ([providerparams](#) &params)  
*\ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param params A map of parameters to initialize the instance \ return A [FeatureProvider](#) implementation \*
- string [toupperstr](#) (string s)  
*toupper implementations for string class*

### Variables

- [GSMFeatureProvider](#) \* fp  
*Singleton implementing the [FeatureProvider](#) interface.*

### 7.33.1 Detailed Description

GSM feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/03/02 12:36:18

**Revision**

1.37

**Since:**

Mon May 12 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[GSM.cpp](#),v 1.37 2004/03/02 12:36:18 rene-cvs Exp

Definition in file [GSM.cpp](#).

### 7.33.2 Function Documentation

#### 7.33.2.1 `string toupperstr (string s) [inline, static]`

toupper implementations for string class

**Parameters:**

*s* Lowercase string

**Returns:**

Uppercase string

Definition at line 53 of file GSM.cpp.

References GSMCellFeature::getName(), and AbstractStringFeature::serialize().

Referenced by GSMFeatureProvider::run().



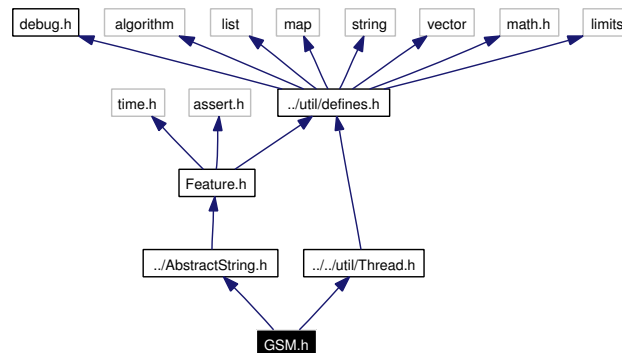
## 7.34 GSM.h File Reference

GSM feature.

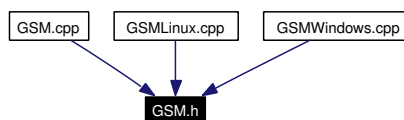
```
#include "../AbstractString.h"
```

```
#include "../../util/Thread.h"
```

Include dependency graph for GSM.h:



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



### Classes

- class [GSMCellFeature](#)
- class [GSMFeatureProvider](#)

### Defines

- #define [GSM\\_SCANDELAY](#) 30  
*Default delay (in seconds) between two GSM cell scans.*

#### 7.34.1 Detailed Description

GSM feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

**Revision**

1.26

**Since:**

Mon May 12 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

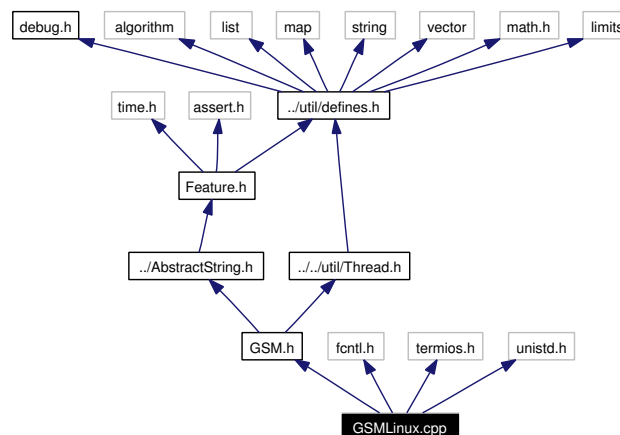
**Id**[GSM.h](#),v 1.26 2004/02/12 20:08:37 harald ExpDefinition in file [GSM.h](#).

## 7.35 GSMLinux.cpp File Reference

GSM feature.

```
#include "GSM.h"
#include <fcntl.h>
#include <termios.h>
#include <unistd.h>
```

Include dependency graph for GSMLinux.cpp:



### Defines

- `#define` [GSM\\_SERIALPORT](#) `"/dev/rfcomm0"`  
Default serial port to use.

### Variables

- `int` [portFd](#)  
Serial port handle.
- `termios` [term\\_old](#)  
Serial port attributes.

### 7.35.1 Detailed Description

GSM feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

**Revision**

1.15

**Since:**

Mon May 12 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

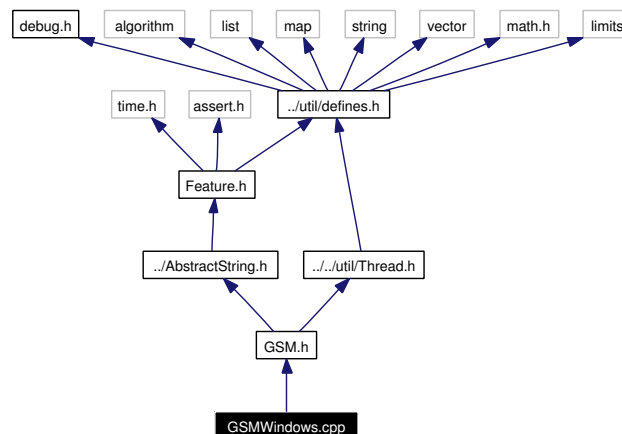
**Id**[GSMLinux.cpp](#),v 1.15 2004/02/12 20:08:37 harald ExpDefinition in file [GSMLinux.cpp](#).

## 7.36 GSMWindows.cpp File Reference

GSM feature.

```
#include "GSM.h"
```

Include dependency graph for GSMWindows.cpp:



### Defines

- #define [GSM\\_SERIALPORT](#) "COM8:"  
*Default serial port to use.*

### Variables

- HANDLE [hPort](#)  
*Serial port handle.*

### 7.36.1 Detailed Description

GSM feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.18

#### Since:

Mon May 12 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

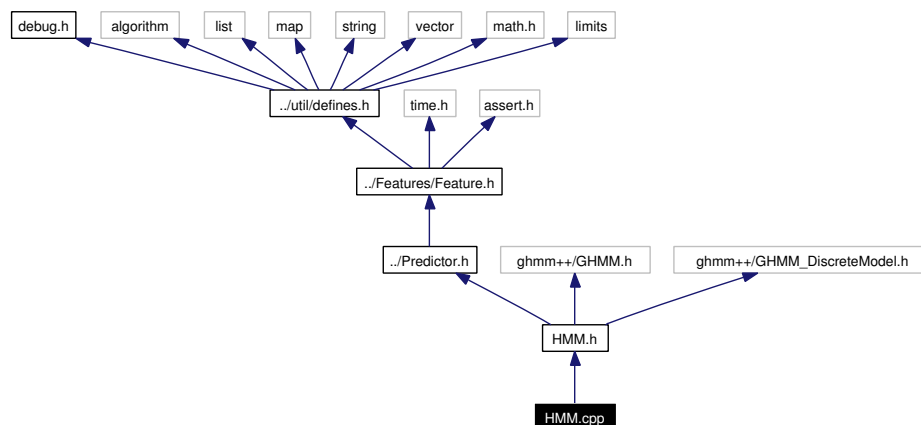
**Id**[GSMWindows.cpp](#),v 1.18 2004/02/12 20:08:37 harald ExpDefinition in file [GSMWindows.cpp](#).

## 7.37 HMM.cpp File Reference

Hidden Markov Model ([HMM](#)) predictor.

```
#include "HMM.h"
```

Include dependency graph for HMM.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [PredictorAlgorithm](#) \* [getPredictor](#) ([predictorparams](#) &params)  
*\ Returns an instance of a class implementing the [PredictorAlgorithm](#) interface \ \ param params A map of parameters to initialize the instance \ return A [PredictorAlgorithm](#) implementation \*
- double \* [createRandNormArray](#) (unsigned int length)

### Variables

- [HMM](#) \* [pred](#)  
*Singleton implementing the [PredictorAlgorithm](#) interface.*

#### 7.37.1 Detailed Description

Hidden Markov Model ([HMM](#)) predictor.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/05/19 11:54:53

**Revision**

1.2

**Since:**

Tue Dec 16 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**[HMM.cpp](#),v 1.2 2004/05/19 11:54:53 rene ExpDefinition in file [HMM.cpp](#).

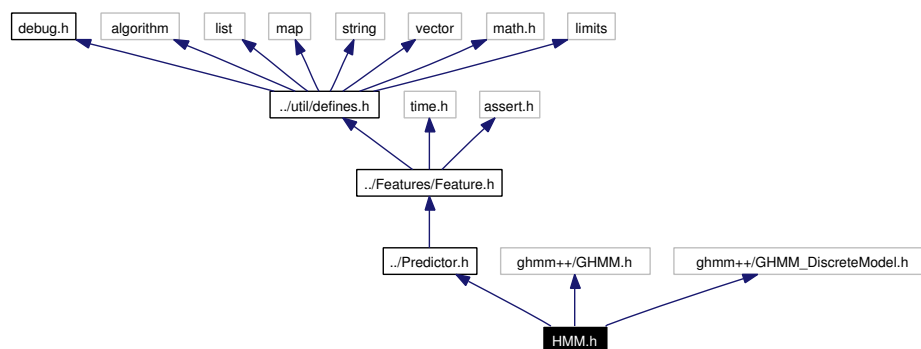


## 7.38 HMM.h File Reference

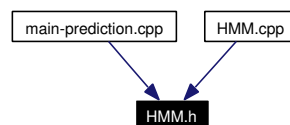
Hidden Markov Model ([HMM](#)) predictor.

```
#include "../Predictor.h"
#include <ghmm++/GHMM.h>
#include <ghmm++/GHMM_DiscreteModel.h>
```

Include dependency graph for HMM.h:



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



### Classes

- class [HMM](#)

### 7.38.1 Detailed Description

Hidden Markov Model ([HMM](#)) predictor.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/05/18 15:09:33

#### Revision

1.1

#### Since:

Tue Dec 16 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

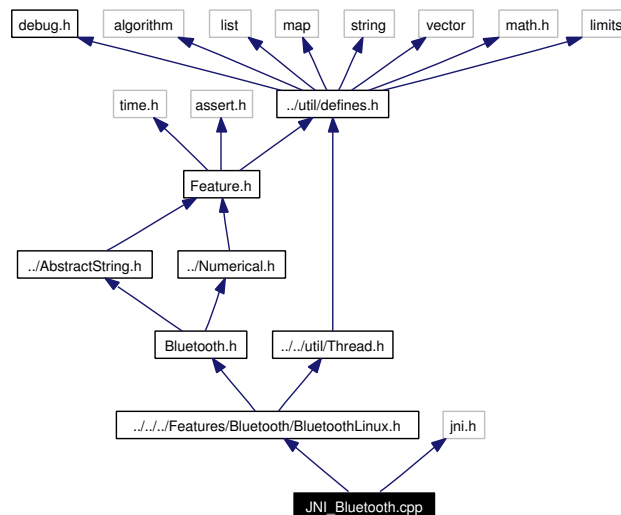
**Id**[HMM.h](#),v 1.1 2004/05/18 15:09:33 rene ExpDefinition in file [HMM.h](#).

## 7.39 JNI\_Bluetooth.cpp File Reference

Bluetooth feature JNI wrapper.

```
#include "../../Features/Bluetooth/BluetoothLinux.h"
#include <jni.h>
```

Include dependency graph for JNI\_Bluetooth.cpp:



### Functions

- [FeatureProvider \\* getProvider](#) (const [providerparams](#) &params)  
*Prototype.*
- JNIEXPORT void JNICALL [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeInit](#) (JNIEnv \*, jobject)
- JNIEXPORT void JNICALL [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeStart](#) (JNIEnv \*, jobject)
- JNIEXPORT void JNICALL [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeStop](#) (JNIEnv \*, jobject)
- JNIEXPORT jboolean JNICALL [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeIsAddressValid](#) (JNIEnv \*jenv, jobject, jstring pAdr)
- JNIEXPORT jobjectArray JNICALL [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeGetAddressesInRange](#) (JNIEnv \*jenv, jobject)
- JNIEXPORT jstring JNICALL [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeGetLastMessage](#) (JNIEnv \*, jobject)
- JNIEXPORT jstring JNICALL [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeGetHardwareAddress](#) (JNIEnv \*jenv, jobject)

### Variables

- [BluetoothLinuxFeatureProvider \\* JNI\\_provider](#) = NULL  
*Feature provider singleton.*

### 7.39.1 Detailed Description

Bluetooth feature JNI wrapper.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/12 20:08:38

**Revision**

1.1

**Since:**

Tue Mar 18 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[JNI\\_Bluetooth.cpp](#),v 1.1 2004/02/12 20:08:38 harald Exp

Definition in file [JNI\\_Bluetooth.cpp](#).

### 7.39.2 Function Documentation

#### 7.39.2.1 JNIEXPORT jobjectArray JNICALL Java\_at\_jku\_bluetooth\_LinuxBluetoothAdapter\_nativeGetAddressesInRange (JNIEnv \* *jenv*, jobject)

**Todo**

documentation

Definition at line 111 of file [JNI\\_Bluetooth.cpp](#).

References [BluetoothLinuxFeatureProvider::getSample\(\)](#), [JNI\\_provider](#), [BluetoothFeatureProvider::nextSample\(\)](#), and [stringvector](#).

#### 7.39.2.2 JNIEXPORT jstring JNICALL Java\_at\_jku\_bluetooth\_LinuxBluetoothAdapter\_nativeGetHardwareAddress (JNIEnv \* *jenv*, jobject)

**Todo**

documentation

Definition at line 145 of file [JNI\\_Bluetooth.cpp](#).

### 7.39.2.3 JNIEXPORT jstring JNICALL Java\_at\_jku\_bluetooth\_LinuxBluetoothAdapter\_nativeGetLastMessage (JNIEnv \*, jobject)

#### Todo

documentation

Definition at line 138 of file JNI\_Bluetooth.cpp.

### 7.39.2.4 JNIEXPORT void JNICALL Java\_at\_jku\_bluetooth\_LinuxBluetoothAdapter\_nativeInit (JNIEnv \*, jobject)

#### Todo

documentation

Definition at line 40 of file JNI\_Bluetooth.cpp.

References featureparams, getProvider(), and JNI\_provider.

### 7.39.2.5 JNIEXPORT jboolean JNICALL Java\_at\_jku\_bluetooth\_LinuxBluetoothAdapter\_nativeIsValidAddress (JNIEnv \* *jenv*, jobject, jstring *pAdr*)

#### Todo

documentation

Definition at line 71 of file JNI\_Bluetooth.cpp.

References BluetoothLinuxFeatureProvider::getSample(), JNI\_provider, BluetoothFeatureProvider::nextSample(), and stringvector.

### 7.39.2.6 JNIEXPORT void JNICALL Java\_at\_jku\_bluetooth\_LinuxBluetoothAdapter\_nativeStart (JNIEnv \*, jobject)

#### Todo

documentation

Definition at line 52 of file JNI\_Bluetooth.cpp.

### 7.39.2.7 JNIEXPORT void JNICALL Java\_at\_jku\_bluetooth\_LinuxBluetoothAdapter\_nativeStop (JNIEnv \*, jobject)

#### Todo

documentation

Definition at line 59 of file JNI\_Bluetooth.cpp.

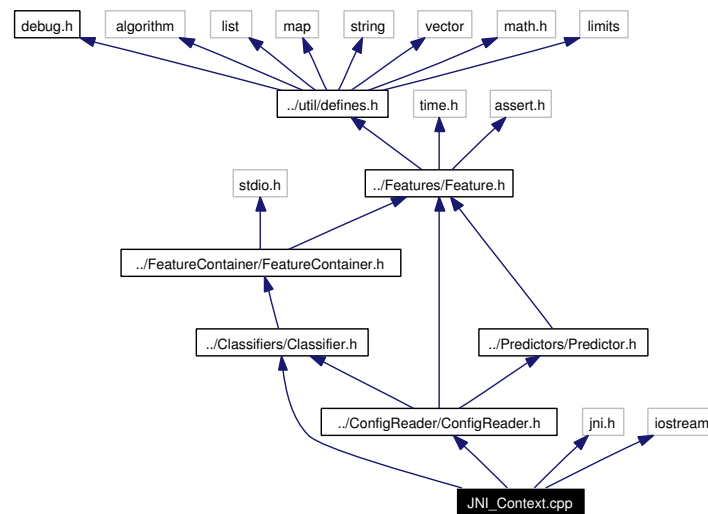
References Thread::cancel(), and JNI\_provider.

## 7.40 JNI\_Context.cpp File Reference

JNI wrapper.

```
#include "../Classifiers/Classifier.h"
#include "../ConfigReader/ConfigReader.h"
#include <jni.h>
#include <iostream>
```

Include dependency graph for JNI\_Context.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- JNIEXPORT jboolean JNICALL [Java\\_at\\_jku\\_intelligence\\_context\\_Context\\_nativeInit](#) (JNIEnv \*jenv, jclass, jstring configFilename, jstring persistFilename, jstring gngPersistFilename)
- JNIEXPORT jlong JNICALL [Java\\_at\\_jku\\_intelligence\\_context\\_Context\\_nativeNextSample](#) (JNIEnv \*, jclass)
- JNIEXPORT jint JNICALL [Java\\_at\\_jku\\_intelligence\\_context\\_Context\\_nativeGetNumClusters](#) (JNIEnv \*, jclass)
- JNIEXPORT void JNICALL [Java\\_at\\_jku\\_intelligence\\_context\\_Context\\_nativeGetClusterMembership](#) (JNIEnv \*jenv, jclass, jlongArray classes, jdoubleArray memberships)

### Variables

- [FeatureContainer \\* fc](#)  
*FeatureContainer instance.*

- [ClassifierAlgorithm](#) \* *ca*  
*ClassifierAlgorithm* instance.

### 7.40.1 Detailed Description

JNI wrapper.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/12 20:08:38

**Revision**

1.8

**Since:**

Thu May 8 2003

**Author:**

Rene Mayrhofer <[reene@mayrhofer.eu.org](mailto:reene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

**Id**

[JNI\\_Context.cpp](#),v 1.8 2004/02/12 20:08:38 harald Exp

Definition in file [JNI\\_Context.cpp](#).

### 7.40.2 Function Documentation

#### 7.40.2.1 JNIEXPORT void JNICALL Java\_at\_jku\_intelligence\_context\_Context\_nativeGetClusterMembership (JNIEnv \* *jenv*, jclass, jlongArray *classes*, jdoubleArray *memberships*)

**Todo**

documentation

Definition at line 137 of file [JNI\\_Context.cpp](#).

References [ca](#), [fc](#), [ClassifierAlgorithm::getClusterMembership\(\)](#), [FeatureContainer::getSampleVector\(\)](#), and [membershiplist](#).

#### 7.40.2.2 JNIEXPORT jint JNICALL Java\_at\_jku\_intelligence\_context\_Context\_nativeGetNumClusters (JNIEnv \*, jclass)

**Todo**

documentation

Definition at line 126 of file JNI\_Context.cpp.

References `ca`, `fc`, `ClassifierAlgorithm::getClusterMembership()`, and `FeatureContainer::getSampleVector()`.

**7.40.2.3 JNIEXPORT jboolean JNICALL Java\_at\_jku\_intelligence\_context\_Context\_nativeInit**  
(JNIEnv \* *jenv*, jclass, jstring *configFilename*, jstring *persistFilename*, jstring *gngPersistFilename*)

**Todo**

documentation

Definition at line 77 of file JNI\_Context.cpp.

References `ca`, `fc`, `ConfigReader::getClassifier()`, and `ClassifierAlgorithm::init()`.

**7.40.2.4 JNIEXPORT jlong JNICALL Java\_at\_jku\_intelligence\_context\_Context\_nativeNextSample** (JNIEnv \*, jclass)

**Todo**

documentation

Definition at line 115 of file JNI\_Context.cpp.

References `ca`, and `ClassifierAlgorithm::nextSample()`.

## 7.40.3 Variable Documentation

### 7.40.3.1 `FeatureContainer* fc`

`FeatureContainer` instance.

**Remarks:**

also used by `JNI_Samples.cpp`

Definition at line 53 of file JNI\_Context.cpp.

Referenced by `evaluate()`, `GNG::init()`, `Java_at_jku_intelligence_context_Context_nativeGetClusterMembership()`, `Java_at_jku_intelligence_context_Context_nativeGetNumClusters()`, `Java_at_jku_intelligence_context_Context_nativeInit()`, `Java_at_jku_intelligence_samples_SampleContainer_nativeGetSampleVector()`, `Java_at_jku_intelligence_samples_SampleContainer_nativeInit()`, `Java_at_jku_intelligence_samples_SampleContainer_nativeNextSamples()`, `library_finalize()`, `library_initialize()`, `main()`, and `Classifier::unserialize()`.

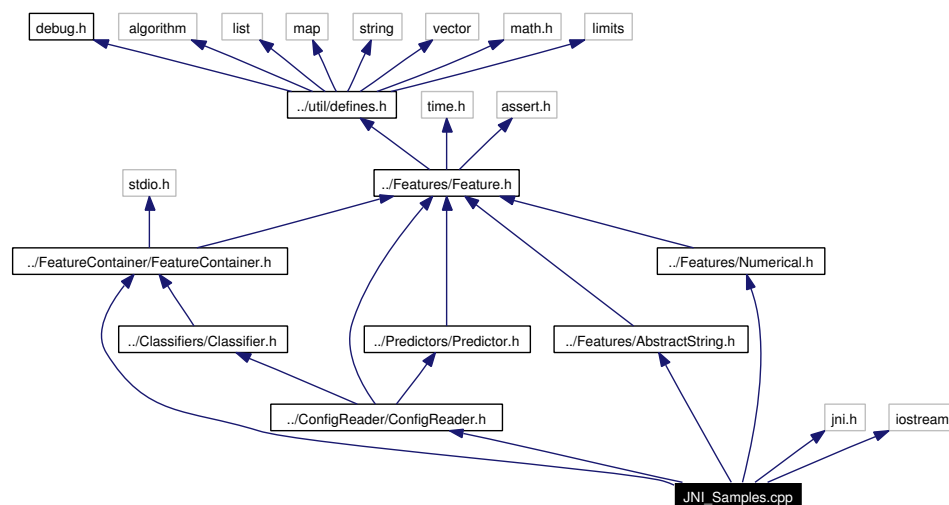


## 7.41 JNI\_Samples.cpp File Reference

JNI wrapper.

```
#include "../ConfigReader/ConfigReader.h"
#include "../Features/AbstractString.h"
#include "../Features/Numerical.h"
#include "../FeatureContainer/FeatureContainer.h"
#include <jni.h>
#include <iostream>
```

Include dependency graph for JNI\_Samples.cpp:



### Defines

- #define `cast_to(type, ref)` `*((type **) &ref)`  
Cast a Java object reference to an object.

### Functions

- JNIEXPORT jdouble JNICALL `Java_at_jku_intelligence_samples_Sample_nativeGetPosition` (JNIEnv \*, jobject, jint objRef)
- JNIEXPORT jint JNICALL `Java_at_jku_intelligence_samples_Sample_nativeGetType` (JNIEnv \*, jobject, jint objRef)
- JNIEXPORT jstring JNICALL `Java_at_jku_intelligence_samples_Sample_nativeGetName` (JNIEnv \*jenv, jobject, jint objRef)
- JNIEXPORT void JNICALL `Java_at_jku_intelligence_samples_Sample_nativeFreeObject` (JNIEnv \*, jobject, jint objRef)
- JNIEXPORT jstring JNICALL `Java_at_jku_intelligence_samples_Sample_nativeToString` (JNIEnv \*jenv, jobject, jint objRef)

- JNIEXPORT jint JNICALL [Java\\_at\\_jku\\_intelligence\\_samples\\_NumericalDiscreteSample\\_nativeGetVal](#) (JNIEnv \*, jobject, jint objRef)
- JNIEXPORT jfloat JNICALL [Java\\_at\\_jku\\_intelligence\\_samples\\_NumericalContinuousSample\\_nativeGetVal](#) (JNIEnv \*, jobject, jint objRef)
- JNIEXPORT jstring JNICALL [Java\\_at\\_jku\\_intelligence\\_samples\\_StringSample\\_nativeGetVal](#) (JNIEnv \*jenv, jobject, jint objRef)
- JNIEXPORT jobjectArray JNICALL [Java\\_at\\_jku\\_intelligence\\_samples\\_StringListSample\\_nativeGetValues](#) (JNIEnv \*jenv, jobject, jint objRef)
- JNIEXPORT jboolean JNICALL [Java\\_at\\_jku\\_intelligence\\_samples\\_SampleContainer\\_nativeInit](#) (JNIEnv \*jenv, jclass, jstring configFilename, jstring persistFilename)
- JNIEXPORT void JNICALL [Java\\_at\\_jku\\_intelligence\\_samples\\_SampleContainer\\_nativeNextSamples](#) (JNIEnv \*, jclass)
- JNIEXPORT jobjectArray JNICALL [Java\\_at\\_jku\\_intelligence\\_samples\\_SampleContainer\\_nativeGetSampleVector](#) (JNIEnv \*jenv, jclass)

## Variables

- [FeatureContainer](#) \* [fc](#)  
*FeatureContainer* instance.

### 7.41.1 Detailed Description

JNI wrapper.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/01 11:15:54

#### Revision

1.20

#### Since:

Thu May 8 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[JNI\\_Samples.cpp](#),v 1.20 2004/03/01 11:15:54 rene Exp

Definition in file [JNI\\_Samples.cpp](#).

## 7.41.2 Function Documentation

### 7.41.2.1 JNIEXPORT jfloat JNICALL Java\_at\_jku\_intelligence\_samples - NumericalContinuousSample\_nativeGetVal (JNIEnv \*, jobject, jint *objRef*)

#### Todo

documentation

Definition at line 120 of file JNI\_Samples.cpp.

References `cast_to`, and `NumericalContinuousFeature::getVal()`.

### 7.41.2.2 JNIEXPORT jint JNICALL Java\_at\_jku\_intelligence\_samples - NumericalDiscreteSample\_nativeGetVal (JNIEnv \*, jobject, jint *objRef*)

#### Todo

documentation

Definition at line 110 of file JNI\_Samples.cpp.

References `cast_to`, and `NumericalDiscreteFeature::getVal()`.

### 7.41.2.3 JNIEXPORT void JNICALL Java\_at\_jku\_intelligence\_samples\_Sample\_nativeFree-Object (JNIEnv \*, jobject, jint *objRef*)

#### Todo

documentation

Definition at line 88 of file JNI\_Samples.cpp.

### 7.41.2.4 JNIEXPORT jstring JNICALL Java\_at\_jku\_intelligence\_samples\_Sample\_nativeGetName (JNIEnv \* *jenv*, jobject, jint *objRef*)

#### Todo

documentation

Definition at line 78 of file JNI\_Samples.cpp.

References `cast_to`, and `Feature::getName()`.

### 7.41.2.5 JNIEXPORT jdouble JNICALL Java\_at\_jku\_intelligence\_samples\_Sample\_nativeGetPosition (JNIEnv \*, jobject, jint *objRef*)

#### Todo

documentation

Definition at line 58 of file JNI\_Samples.cpp.

References `cast_to`, and `Feature::getPosition()`.

**7.41.2.6 JNIEXPORT jint JNICALL Java\_at\_jku\_intelligence\_samples\_Sample\_nativeGetType (JNIEnv \*, jobject, jint *objRef*)**

**Todo**

documentation

Definition at line 68 of file JNI\_Samples.cpp.

References `cast_to`, and `Feature::getType()`.

**7.41.2.7 JNIEXPORT jobjectArray JNICALL Java\_at\_jku\_intelligence\_samples\_SampleContainer\_nativeGetSampleVector (JNIEnv \* *jenv*, jclass)**

**Todo**

documentation

Definition at line 191 of file JNI\_Samples.cpp.

References `fc`, `featurevector`, and `FeatureContainer::getSampleVector()`.

**7.41.2.8 JNIEXPORT jboolean JNICALL Java\_at\_jku\_intelligence\_samples\_SampleContainer\_nativeInit (JNIEnv \* *jenv*, jclass, jstring *configFilename*, jstring *persistFilename*)**

**Todo**

documentation

Definition at line 159 of file JNI\_Samples.cpp.

References `fc`.

**7.41.2.9 JNIEXPORT void JNICALL Java\_at\_jku\_intelligence\_samples\_SampleContainer\_nativeNextSamples (JNIEnv \*, jclass)**

**Todo**

documentation

Definition at line 182 of file JNI\_Samples.cpp.

References `fc`, and `FeatureContainer::nextSample()`.

**7.41.2.10 JNIEXPORT jobjectArray JNICALL Java\_at\_jku\_intelligence\_samples\_StringListSample\_nativeGetValues (JNIEnv \* *jenv*, jobject, jint *objRef*)**

**Todo**

documentation

Definition at line 140 of file JNI\_Samples.cpp.

References `cast_to`, `AbstractStringListFeature::getListValues()`, and `stringvector`.

#### 7.41.2.11 JNIEXPORT jstring JNICALL Java\_at\_jku\_intelligence\_samples\_StringSample\_nativeGetVal (JNIEnv \* *jenv*, jobject, jint *objRef*)

##### Todo

documentation

Definition at line 130 of file JNI\_Samples.cpp.

References `cast_to`, and `AbstractStringFeature::getVal()`.

### 7.41.3 Variable Documentation

#### 7.41.3.1 `FeatureContainer* fc`

`FeatureContainer` instance.

##### Remarks:

also used by [JNI\\_Samples.cpp](#)

Definition at line 53 of file JNI\_Context.cpp.

Referenced by `evaluate()`, `GNG::init()`, `Java_at_jku_intelligence_context_Context_nativeGetClusterMembership()`, `Java_at_jku_intelligence_context_Context_nativeGetNumClusters()`, `Java_at_jku_intelligence_context_Context_nativeInit()`, `Java_at_jku_intelligence_samples_SampleContainer_nativeGetSampleVector()`, `Java_at_jku_intelligence_samples_SampleContainer_nativeInit()`, `Java_at_jku_intelligence_samples_SampleContainer_nativeNextSamples()`, `library_finalize()`, `library_initialize()`, `main()`, and `Classifier::unserialize()`.

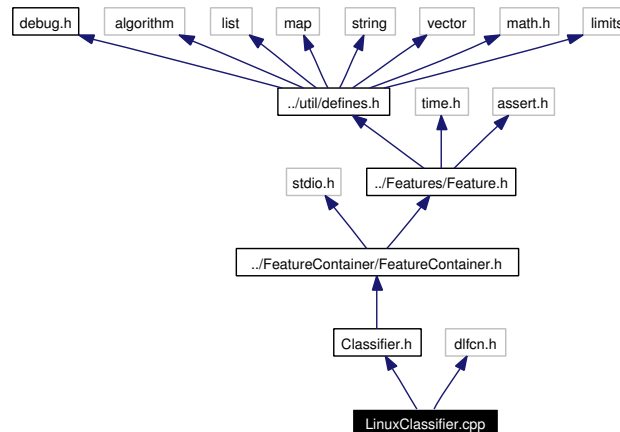
## 7.42 LinuxClassifier.cpp File Reference

Classifier linux implementation

```
#include "Classifier.h"
```

```
#include <dlfcn.h>
```

Include dependency graph for LinuxClassifier.cpp:



### 7.42.1 Detailed Description

Classifier linux implementation

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/24 13:18:47

#### Revision

1.12

#### Since:

Wed Mar 19 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[LinuxClassifier.cpp](#),v 1.12 2004/02/24 13:18:47 rene Exp

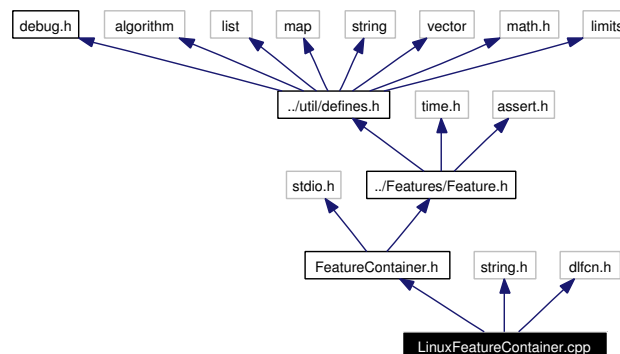
Definition in file [LinuxClassifier.cpp](#).

## 7.43 LinuxFeatureContainer.cpp File Reference

Feature container

```
#include "FeatureContainer.h"
#include <string.h>
#include <dlfcn.h>
```

Include dependency graph for LinuxFeatureContainer.cpp:



### 7.43.1 Detailed Description

Feature container

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.8

#### Since:

Thu Apr 10 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[LinuxFeatureContainer.cpp](#), v 1.8 2004/02/12 20:08:37 harald Exp

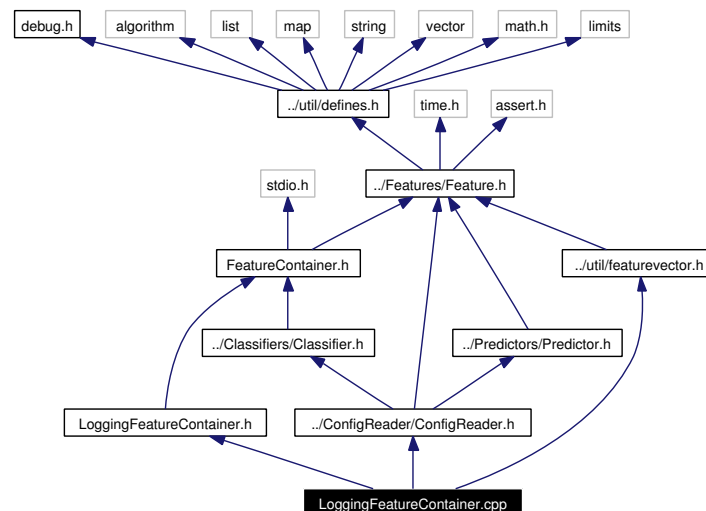
Definition in file [LinuxFeatureContainer.cpp](#).

## 7.44 LoggingFeatureContainer.cpp File Reference

Feature container

```
#include "LoggingFeatureContainer.h"
#include "../ConfigReader/ConfigReader.h"
#include "../util/featurevector.h"
```

Include dependency graph for LoggingFeatureContainer.cpp:



### 7.44.1 Detailed Description

Feature container

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.20

#### Since:

Thu Apr 10 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[LoggingFeatureContainer.cpp](#),v 1.20 2004/02/12 20:08:37 harald Exp



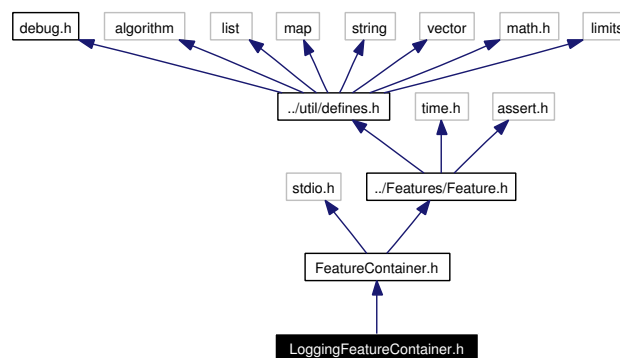
Definition in file [LoggingFeatureContainer.cpp](#).

## 7.45 LoggingFeatureContainer.h File Reference

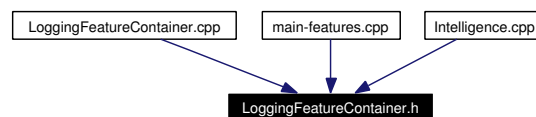
Feature container

```
#include "FeatureContainer.h"
```

Include dependency graph for LoggingFeatureContainer.h:



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



### Classes

- class [LoggingFeatureContainer](#)

### 7.45.1 Detailed Description

Feature container

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.12

#### Since:

Thu Apr 10 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

**Id**

[LoggingFeatureContainer.h](#),v 1.12 2004/02/12 20:08:37 harald Exp

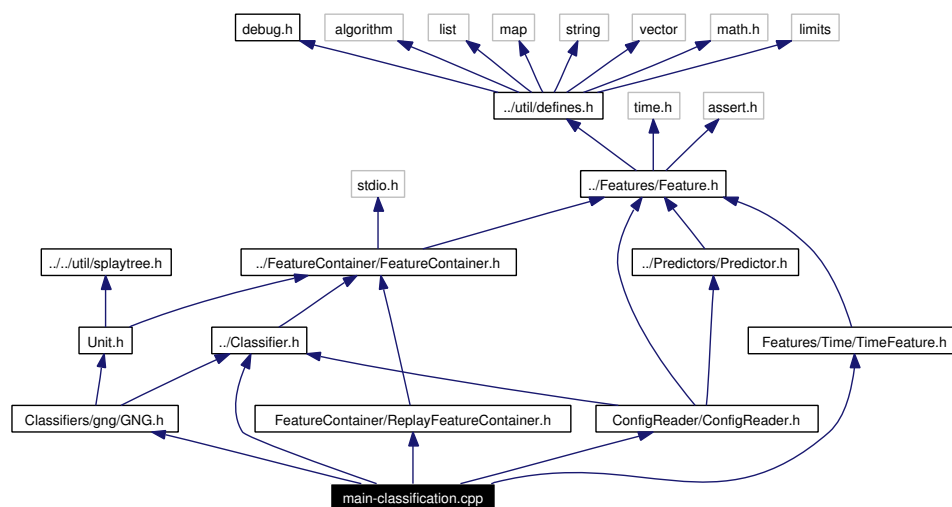
Definition in file [LoggingFeatureContainer.h](#).

## 7.46 main-classification.cpp File Reference

main

```
#include "Classifiers/gng/GNG.h"
#include "Features/Time/TimeFeature.h"
#include "ConfigReader/ConfigReader.h"
#include "FeatureContainer/ReplayFeatureContainer.h"
#include "Classifiers/Classifier.h"
```

Include dependency graph for main-classification.cpp:



### Defines

- #define **CLASSIFICATION\_RUNS** 1  
*amount of training iterations*
- #define **HOT** 0.995  
*start temperature for sa*
- #define **COLD** 0.0  
*end temperature for sa*
- #define **MINIMIZE\_ERROR** 0  
*try to minimize classification error by finding the best classifier parameters*
- #define **MINIMIZE\_VARIANCE** 1  
*try to minimize classification error by finding the best classifier parameters*
- #define **tune**(param) while (gng → param == param) { gng → param = param + param \* sign \* rand\_double(); }
- #define **bound**(param, min, max) if(gng → param < min) gng → param=min; if(gng → param > max) gng → param=max;

## Functions

- void [evaluate](#) ([ReplayFeatureContainer](#) \*fc, string [classes\\_log](#)="")  
*Evaluate quality of training.*
- int [main](#) (int argc, char \*argv[ ])  
*Main.*

### 7.46.1 Detailed Description

main

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/12 10:58:06

#### Revision

1.57

#### Since:

Mon Mar 10 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

main-classification.cpp,v 1.57 2004/03/12 10:58:06 rene Exp

Definition in file [main-classification.cpp](#).

### 7.46.2 Typedef Documentation

#### 7.46.2.1 typedef [map](#)<unsigned, [list](#)<double> > [cluster\\_list](#)

##### Todo

documentation

Definition at line 51 of file main-classification.cpp.

#### 7.46.2.2 typedef [map](#)<unsigned, [cluster\\_list](#)> [meta\\_list](#)

##### Todo

documentation

Definition at line 52 of file main-classification.cpp.

Referenced by [evaluate\(\)](#).

### 7.46.3 Function Documentation

#### 7.46.3.1 void evaluate ([ReplayFeatureContainer](#) \*fc, string classes\_log = " ") [static]

Evaluate quality of training.

##### Parameters:

*fc* [ReplayFeatureContainer](#) reference

*classes\_log* Path to logfile

Annealing operator Definition at line 65 of file main-classification.cpp.

References GNG::adaptation\_threshold, algo\_name, algo\_params, classes\_log, CLASSIFICATION\_RUNS, COLD, GNG::deletion\_threshold, GNG::deletion\_threshold\_lq, GNG::edge\_age\_max, fc, featurevector, Classifier::getClusterMembership(), FeatureContainer::getSampleVector(), TimeFeature::getVal(), GNG::getWinnerDistance(), HOT, Classifier::init(), GNG::insertion\_tolerance, GNG::lambda, GNG::lr\_insert\_threshold, GNG::lr\_neighbor, GNG::lr\_winner, membershiplist, meta\_list, GNG::minimal\_deletion\_age, MINIMIZE\_ERROR, GNG::minimum\_lr, Classifier::nextSample(), FeatureContainer::nextSample(), rand\_double, Classifier::serialize(), GNG::T\_insert\_threshold, GNG::T\_long, GNG::T\_short, and GNG::T\_youth.

Referenced by main().

### 7.46.4 Variable Documentation

#### 7.46.4.1 string algo\_name [static]

##### Todo

documentation

Definition at line 55 of file main-classification.cpp.

Referenced by evaluate(), and main().

#### 7.46.4.2 map<string, string> algo\_params [static]

##### Todo

documentation

Definition at line 56 of file main-classification.cpp.

Referenced by evaluate(), and main().

#### 7.46.4.3 string classes\_log [static]

##### Todo

documentation

Definition at line 54 of file main-classification.cpp.

Referenced by evaluate(), and main().

## 7.47 main-features.cpp File Reference

main

```
#include "util/defines.h"
#include "ConfigReader/ConfigReader.h"
#include "FeatureContainer/LoggingFeatureContainer.h"
#include "Classifiers/Classifier.h"
#include "util/Thread.h"
#include "trayicon/soapH.h"
#include "trayicon/ContextApplicationsBinding.nsmmap"
```

Include dependency graph for main-features.cpp:



### Classes

- class [Scanner](#)

### Defines

- #define **DO\_SOAP**
- #define **DEFAULT\_SOAP\_PORT** 8000
- #define **DO\_CLASSIFICATION**
- #define **DO\_CLASSIFICATION**

### Functions

- int [Main](#) ()  
*Main.*
- SOAP\_FMAC5 int SOAP\_FMAC6 [ns1\\_\\_getActiveId](#) (struct soap \*soapP, struct ns1\_\_getActiveId-Response &param)  
*Implementation of the SOAP method.*

#### 7.47.1 Detailed Description

main

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/06/08 10:45:41

**Revision**

1.32

**Since:**

Mon Mar 10 2003

**Author:**Rene Mayrhofer <[reene@mayrhofer.eu.org](mailto:reene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

main-features.cpp,v 1.32 2004/06/08 10:45:41 rene Exp

Definition in file [main-features.cpp](#).

## 7.47.2 Variable Documentation

**7.47.2.1** `long curContextId = -1` `[static]`**Todo**

documentation

Definition at line 66 of file main-features.cpp.

Referenced by `ns1__getActiveId()`, and `Scanner::run()`.**7.47.2.2** `bool keepRunning = true` `[static]`**Todo**

documentation

Definition at line 63 of file main-features.cpp.

Referenced by `Main()`, and `Scanner::run()`.**7.47.2.3** `bool terminated = false` `[static]`**Todo**

documentation

Definition at line 64 of file main-features.cpp.

Referenced by `Main()`, and `Scanner::run()`.

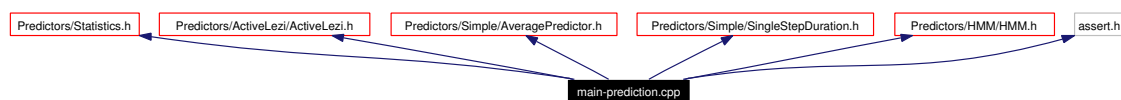


## 7.48 main-prediction.cpp File Reference

main

```
#include "Predictors/Statistics.h"
#include "Predictors/ActiveLezi/ActiveLezi.h"
#include "Predictors/Simple/AveragePredictor.h"
#include "Predictors/Simple/SingleStepDuration.h"
#include "Predictors/HMM/HMM.h"
#include <assert.h>
```

Include dependency graph for main-prediction.cpp:



### Classes

- class [PredWrapper](#)

### Functions

- int [main](#) (int argc, char \*argv[ ])  
*Main.*

### 7.48.1 Detailed Description

main

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/08/09 13:35:18

#### Revision

1.20

#### Since:

Wed Dec 17 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

**Id**

main-prediction.cpp,v 1.20 2004/08/09 13:35:18 rene Exp

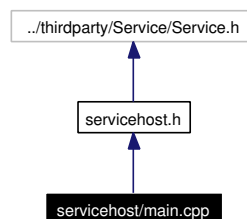
Definition in file [main-prediction.cpp](#).

## 7.49 main.cpp File Reference

Windows NT service.

```
#include "servicehost.h"
```

Include dependency graph for servicehost/main.cpp:



### Functions

- int [main](#) (int argc, char \*argv[ ])  
*Main.*

### 7.49.1 Detailed Description

Windows NT service.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/21 22:58:58

#### Revision

1.7

#### Since:

Tue Aug 8 2003

#### Author:

Rene Mayrhofer <[reene@mayrhofer.eu.org](mailto:reene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

main.cpp,v 1.7 2004/02/21 22:58:58 harald Exp

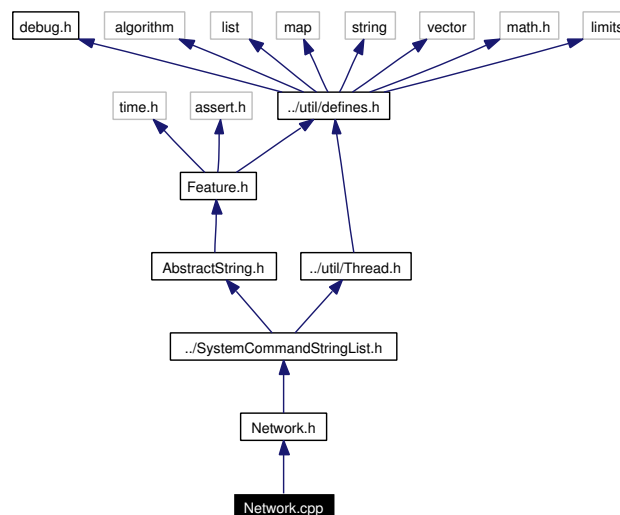
Definition in file [servicehost/main.cpp](#).

## 7.50 Network.cpp File Reference

Network feature.

```
#include "Network.h"
```

Include dependency graph for Network.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) ([providerparams](#) &params)  
*\ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param params A map of parameters to initialize the instance \ return A [FeatureProvider](#) implementation \*

### Variables

- [NetworkFeatureProvider](#) \* fp  
*Singleton implementing the [FeatureProvider](#) interface.*

#### 7.50.1 Detailed Description

Network feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/06/08 09:38:41

**Revision**

1.2

**Since:**

Mon Jul 7 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

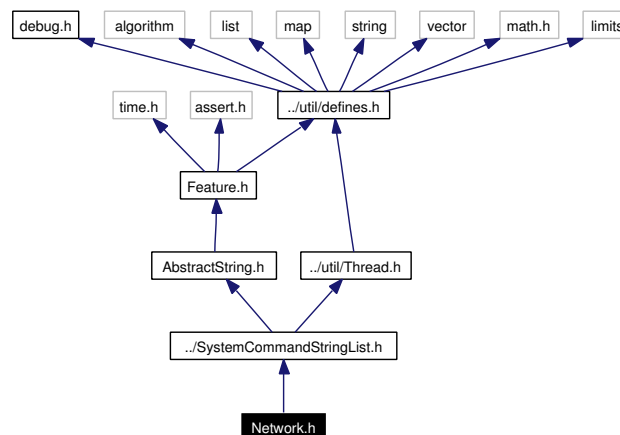
**Id**[Network.cpp](#),v 1.2 2004/06/08 09:38:41 rene ExpDefinition in file [Network.cpp](#).

## 7.51 Network.h File Reference

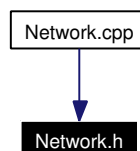
Wlan feature.

```
#include "../SystemCommandStringList.h"
```

Include dependency graph for Network.h:



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



### Classes

- class [NetworkFeatureProvider](#)

### 7.51.1 Detailed Description

Wlan feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/06/08 09:38:41

#### Revision

1.2

#### Since:

Mon Jul 7 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[Network.h](#),v 1.2 2004/06/08 09:38:41 rene Exp

Definition in file [Network.h](#).

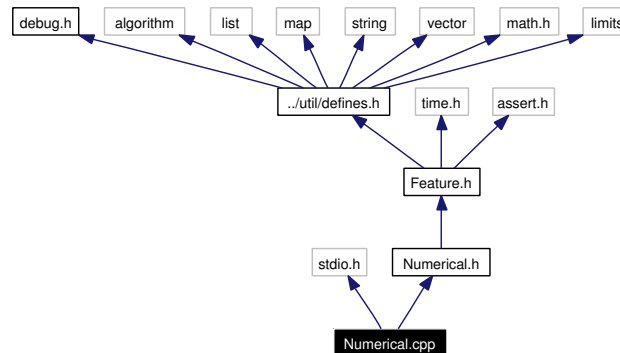
## 7.52 Numerical.cpp File Reference

Abstract string feature class implementation.

```
#include <stdio.h>
```

```
#include "Numerical.h"
```

Include dependency graph for Numerical.cpp:



### 7.52.1 Detailed Description

Abstract string feature class implementation.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/12 10:56:44

#### Revision

1.42

#### Since:

Wed May 7 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[Numerical.cpp](#),v 1.42 2004/03/12 10:56:44 rene Exp

Definition in file [Numerical.cpp](#).

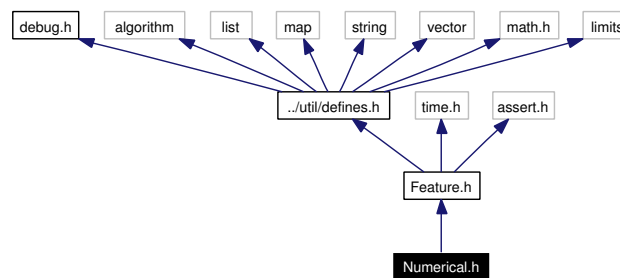


## 7.53 Numerical.h File Reference

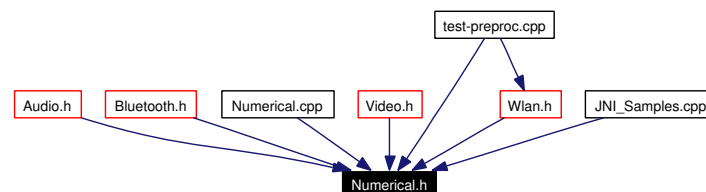
Numerical feature class declaration.

```
#include "Feature.h"
```

Include dependency graph for Numerical.h:



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



### Classes

- class [NumericalDiscreteFeature](#)  
*Abstract discrete numerical feature.*
- class [NumericalContinuousFeature](#)  
*Abstract continuous numerical feature.*

### 7.53.1 Detailed Description

Numerical feature class declaration.

This file contains the abstract class definitions that can be used by sensors that return numerical values.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/01 15:47:49

#### Revision

1.23

**Since:**

Wed May 7 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[Numerical.h](#),v 1.23 2004/03/01 15:47:49 rene Exp

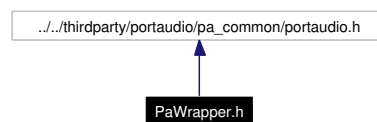
Definition in file [Numerical.h](#).

## 7.54 PaWrapper.h File Reference

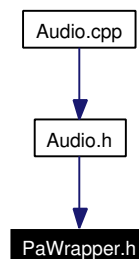
Microphone feature.

```
#include "../thirdparty/portaudio/pa_common/portaudio.h"
```

Include dependency graph for PaWrapper.h:



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



### Defines

- #define [Fixed\\_Pa\\_GetDefaultInputDeviceID\(\)](#) Pa\_GetDefaultInputDeviceID()  
*this returns "0" on Compaq Ipaq, but "1" is the correct input device ID*
- #define [PA\\_SAMPLE\\_TYPE](#) paInt16  
*Sample type identifier.*
- #define [SAMPLING\\_RATE](#) 22050  
*Sampling rate.*

### Typedefs

- typedef short [SAMPLE](#)  
*Sample type.*

#### 7.54.1 Detailed Description

Microphone feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/12 20:08:37

**Revision**

1.10

**Since:**

Wed Apr 9 2003

**Author:**

Rene Mayrhofer <[reene@mayrhofer.eu.org](mailto:reene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[PaWrapper.h](#),v 1.10 2004/02/12 20:08:37 harald Exp

Definition in file [PaWrapper.h](#).

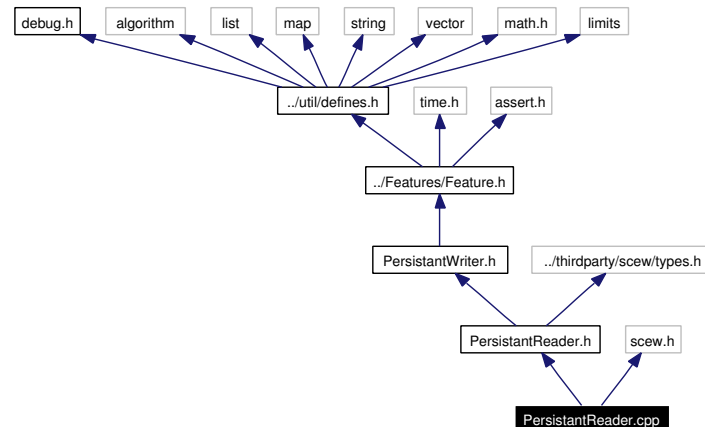
## 7.55 PersistentReader.cpp File Reference

Persistent file reader.

```
#include "PersistentReader.h"
```

```
#include <scew.h>
```

Include dependency graph for PersistentReader.cpp:



### 7.55.1 Detailed Description

Persistent file reader.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/02 15:55:18

#### Revision

1.6

#### Since:

Mon Aug 11 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[PersistentReader.cpp](#),v 1.6 2004/03/02 15:55:18 rene Exp

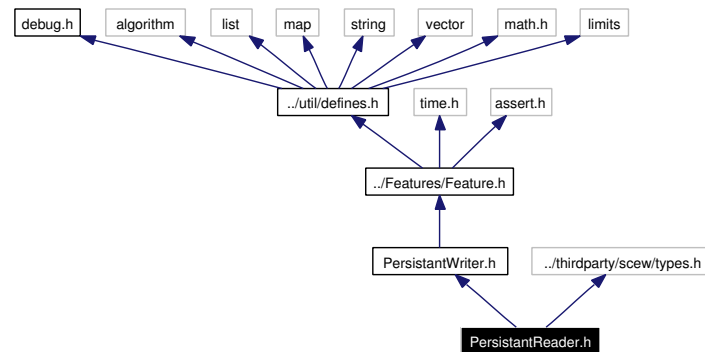
Definition in file [PersistentReader.cpp](#).

## 7.56 PersistantReader.h File Reference

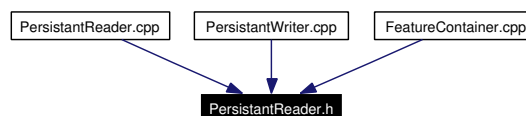
Persistent file reader.

```
#include "PersistantWriter.h"
#include "../thirdparty/scew/types.h"
```

Include dependency graph for PersistantReader.h:



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



### Classes

- class [PersistantReader](#)

### 7.56.1 Detailed Description

Persistent file reader.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/23 11:25:42

#### Revision

1.9

#### Since:

Mon Aug 11 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

**Id**

[PersistentReader.h](#),v 1.9 2004/02/23 11:25:42 rene Exp

Definition in file [PersistentReader.h](#).



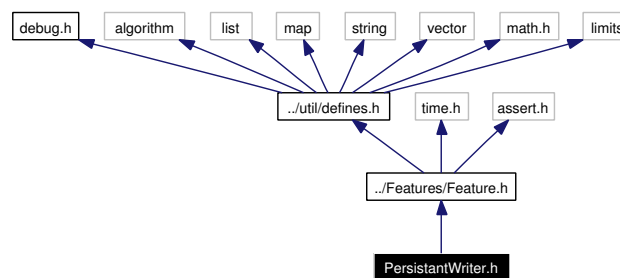


## 7.58 PersistentWriter.h File Reference

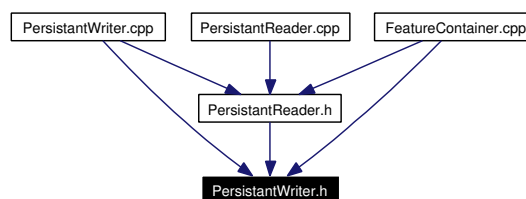
Persistent file writer.

```
#include "../Features/Feature.h"
```

Include dependency graph for PersistentWriter.h:



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



### Classes

- class [PersistentWriter](#)

### 7.58.1 Detailed Description

Persistent file writer.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.5

#### Since:

Mon Aug 11 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

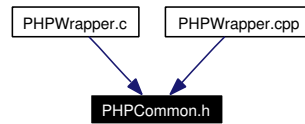
[PersistantWriter.h](#),v 1.5 2004/02/12 20:08:37 harald Exp

Definition in file [PersistantWriter.h](#).

## 7.59 PHPCommon.h File Reference

PHP feature.

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



### Enumerations

- enum [PHPFeatureType](#) {  
    **Error, NumericalDiscrete, NumericalContinuous, StringNone,**  
    **StringLevenshtein, StringListNone, StringListLevenshtein, Custom }**  
    *Feature Types.*

### 7.59.1 Detailed Description

PHP feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.2

#### Since:

Mon Sep 8 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[PHPCommon.h](#),v 1.2 2004/02/12 20:08:37 harald Exp

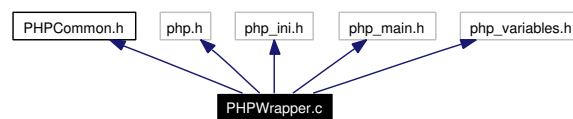
Definition in file [PHPCommon.h](#).

## 7.60 PHPWrapper.c File Reference

PHP feature.

```
#include "PHPCommon.h"
#include "php.h"
#include "php_ini.h"
#include "php_main.h"
#include "php_variables.h"
```

Include dependency graph for PHPWrapper.c:



### Defines

- `#define ZEND_DEBUG 0`  
*disable Zend debugging*

### Functions

- `PHP_MINIT_FUNCTION` (intelligence)  
*Module initializer.*
- `int php_intelligence_startup` (sapi\_module\_struct \*sapi\_module)  
*SAPI startup.*
- `int sapi_intelligence_ub_write` (const char \*str, uint str\_length TSRMLS\_DC)  
*SAPI write.*
- `void sapi_intelligence_flush` (void \*server\_context)  
*SAPI flush.*
- `void sapi_intelligence_register_variables` (zval \*track\_vars\_array TSRMLS\_DC)  
*SAPI register variables.*
- `int sapi_intelligence_post_read` (char \*buf, uint count\_bytes TSRMLS\_DC)  
*SAPI post data read.*
- `char * sapi_intelligence_read_cookies` (TSRMLS\_D)  
*SAPI cookies read.*
- `int sapi_intelligence_send_headers` (sapi\_headers\_struct \*sapi\_headers TSRMLS\_DC)  
*SAPI send headers.*

- void [init\\_php](#) ()  
*Load php runtime.*
- void [shutdown\\_php](#) ()  
*Unload PHP runtime.*
- zval [call\\_function](#) (const char \*file, const char \*func)
- [PHPFeatureType](#) [php\\_init\\_feature](#) (const char \*file)  
*Initialize feature.*
- void \* [php\\_next\\_sample](#) (const char \*file, [PHPFeatureType](#) type)  
*Next sample.*
- void \* [php\\_move\\_towards](#) (const char \*file, [PHPFeatureType](#) type, void \*a, void \*b)  
*Move towards.*
- double [php\\_get\\_distance](#) (const char \*file, [PHPFeatureType](#) type, void \*a, void \*b)  
*Get distance.*

## Variables

- void \*\*\* [tsrm\\_ls](#)  
*thread safe local storage*
- function\_entry [intelligence\\_functions](#) []  
*PHP exported functions.*
- zend\_module\_entry [php\\_intelligence\\_module](#)  
*PHP intelligence module.*
- sapi\_module\_struct [intelligence\\_sapi\\_module](#)  
*SAPI module.*

### 7.60.1 Detailed Description

PHP feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.2

#### Since:

Mon Sep 8 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**[PHPWrapper.c](#),v 1.2 2004/02/12 20:08:37 harald ExpDefinition in file [PHPWrapper.c](#).

## 7.60.2 Function Documentation

### 7.60.2.1 `zval call_function (const char *file, const char *func) [static]`

request shutdown would free the emalloc'ed string and spit out a warning, so we have to strndup the string first and free the emalloc'ed one to have the string available even after request shutdown.

**Todo**`strlen , Z_STRLEN(retval);`Definition at line 189 of file [PHPWrapper.c](#).Referenced by `php_init_feature()`, and `php_next_sample()`.

### 7.60.2.2 `void init_php ()`

Load php runtime.

**Todo**`fix that;`Definition at line 155 of file [PHPWrapper.c](#).References `intelligence_sapi_module`, and `tstrm_ls`.

### 7.60.2.3 `void* php_move_towards (const char *file, PHPFeatureType type, void *a, void *b)`

Move towards.

**Todo**`foo  
COPY CTOR`Definition at line 289 of file [PHPWrapper.c](#).

#### 7.60.2.4 void\* php\_next\_sample (const char \*file, PHPFeatureType type)

Next sample.

##### Todo

foo

Definition at line 251 of file PHPWrapper.c.

References `call_function()`.

### 7.60.3 Variable Documentation

#### 7.60.3.1 function\_entry intelligence\_functions[] [static]

**Initial value:**

```
{
    { NULL, NULL, NULL }
}
```

PHP exported functions.

Definition at line 58 of file PHPWrapper.c.

#### 7.60.3.2 zend\_module\_entry php\_intelligence\_module [static]

**Initial value:**

```
{
    STANDARD_MODULE_HEADER,
    "Intelligence",
    intelligence_functions,
    PHP_MINIT(intelligence),
    NULL,
    NULL,
    NULL,
    NULL,
    "0.1.0",
    STANDARD_MODULE_PROPERTIES
}
```

PHP intelligence module.

Definition at line 63 of file PHPWrapper.c.

Referenced by `php_intelligence_startup()`.

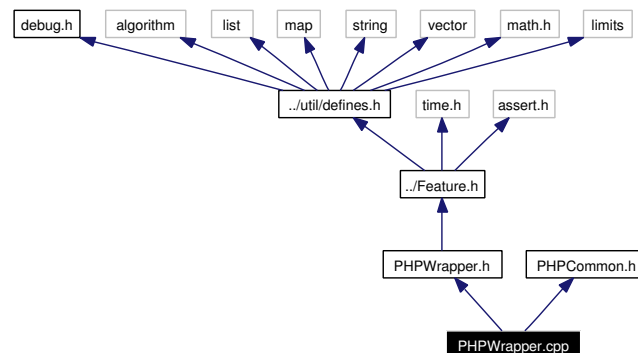
## 7.61 PHPWrapper.cpp File Reference

PHP feature.

```
#include "PHPWrapper.h"
```

```
#include "PHPCommon.h"
```

Include dependency graph for PHPWrapper.cpp:



### Functions

- void [init\\_php](#) ()  
*Load php runtime.*
- void [shutdown\\_php](#) ()  
*Unload PHP runtime.*
- [PHPFeatureType](#) [php\\_init\\_feature](#) (const char \*file)  
*Initialize feature.*
- void \* [php\\_next\\_sample](#) (const char \*file, [PHPFeatureType](#) type)  
*Next sample.*
- void \* [php\\_move\\_towards](#) (const char \*file, [PHPFeatureType](#) type, void \*a, void \*b)  
*Move towards.*
- double [php\\_get\\_distance](#) (const char \*file, [PHPFeatureType](#) type, void \*a, void \*b)  
*Get distance.*
- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) ([providerparams](#) &params)  
 \ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param params A map of  
 parameters to initialize the instance \ return A [FeatureProvider](#) implementation \



## Variables

- [PHPWrapperFeatureProvider](#) \* fp

*Singleton implementing the [FeatureProvider](#) interface.*

### 7.61.1 Detailed Description

PHP feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.2

#### Since:

Mon Sep 8 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[PHPWrapper.cpp](#),v 1.2 2004/02/12 20:08:37 harald Exp

Definition in file [PHPWrapper.cpp](#).

### 7.61.2 Function Documentation

#### 7.61.2.1 void init\_php ()

Load php runtime.

#### Todo

fix that;

Definition at line 155 of file [PHPWrapper.c](#).

References [intelligence\\_sapi\\_module](#), and [tsrm\\_ls](#).

#### 7.61.2.2 void\* php\_move\_towards (const char \*file, [PHPFeatureType](#) type, void \*a, void \*b)

Move towards.

**Todo**

foo  
COPY CTOR

Definition at line 289 of file PHPWrapper.c.

**7.61.2.3 void\* php\_next\_sample (const char \* *file*, **PHPFeatureType** *type*)**

Next sample.

**Todo**

foo

Definition at line 251 of file PHPWrapper.c.

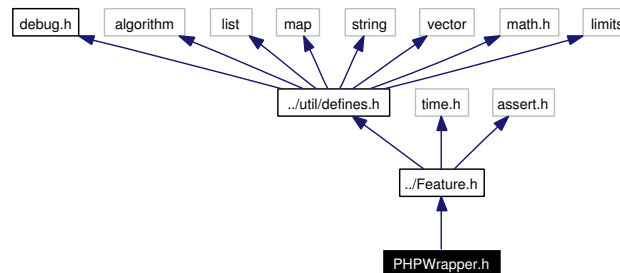
References call\_function().

## 7.62 PHPWrapper.h File Reference

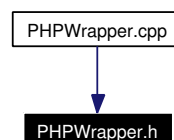
PHP feature.

```
#include "../Feature.h"
```

Include dependency graph for PHPWrapper.h:



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



### Classes

- class [PHPWrapperFeatureProvider](#)

### 7.62.1 Detailed Description

PHP feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.3

#### Since:

Mon Sep 8 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[PHPWrapper.h](#),v 1.3 2004/02/12 20:08:37 harald Exp

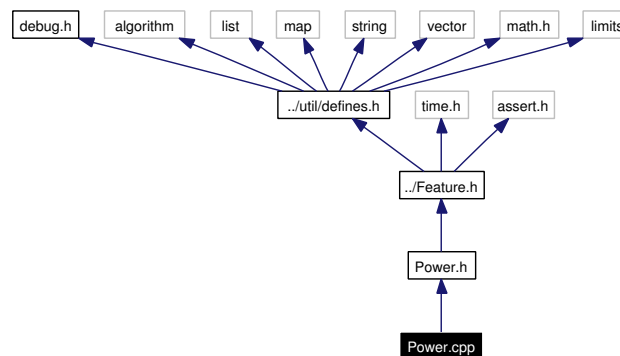
Definition in file [PHPWrapper.h](#).

## 7.63 Power.cpp File Reference

Power feature.

```
#include "Power.h"
```

Include dependency graph for Power.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) ([providerparams](#) &[params](#))  
 \ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param [params](#) A map of parameters to initialize the instance \ return A [FeatureProvider](#) implementation \

### Variables

- [PowerFeatureProvider](#) \* [fp](#)  
*Singleton implementing the [FeatureProvider](#) interface.*

### 7.63.1 Detailed Description

Power feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/02 12:36:18

#### Revision

1.27

**Since:**

Mon Mar 10 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[Power.cpp](#),v 1.27 2004/03/02 12:36:18 rene-cvs Exp

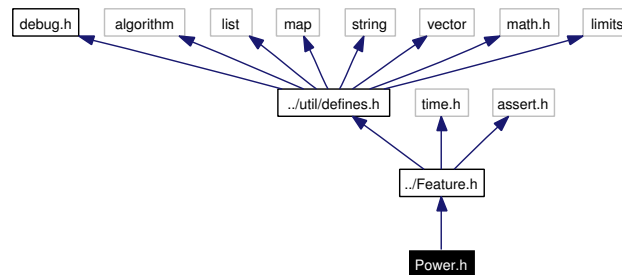
Definition in file [Power.cpp](#).

## 7.64 Power.h File Reference

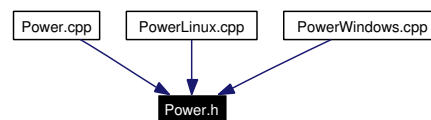
Power feature.

```
#include "../Feature.h"
```

Include dependency graph for Power.h:



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



### Classes

- class [PowerFeature](#)
- class [PowerFeatureProvider](#)

### 7.64.1 Detailed Description

Power feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.23

#### Since:

Mon Mar 10 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[Power.h](#),v 1.23 2004/02/12 20:08:37 harald Exp

Definition in file [Power.h](#).



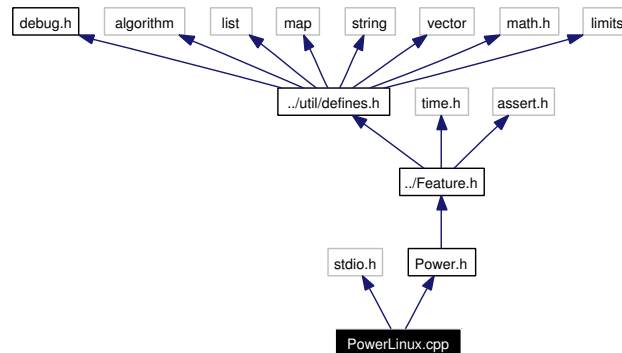
## 7.65 PowerLinux.cpp File Reference

Power feature.

```
#include <stdio.h>
```

```
#include "Power.h"
```

Include dependency graph for PowerLinux.cpp:



### 7.65.1 Detailed Description

Power feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/01/15 14:05:07

#### Revision

1.6

#### Since:

Mon Mar 10 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[PowerLinux.cpp](#),v 1.6 2004/01/15 14:05:07 harald Exp

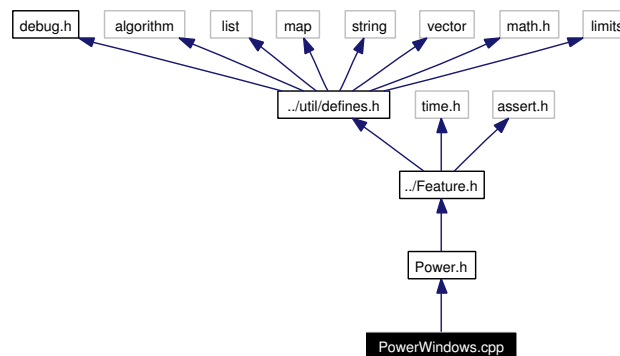
Definition in file [PowerLinux.cpp](#).

## 7.66 PowerWindows.cpp File Reference

Power feature.

```
#include "Power.h"
```

Include dependency graph for PowerWindows.cpp:



### 7.66.1 Detailed Description

Power feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/01/15 14:05:07

#### Revision

1.7

#### Since:

Mon Mar 10 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[PowerWindows.cpp](#),v 1.7 2004/01/15 14:05:07 harald Exp

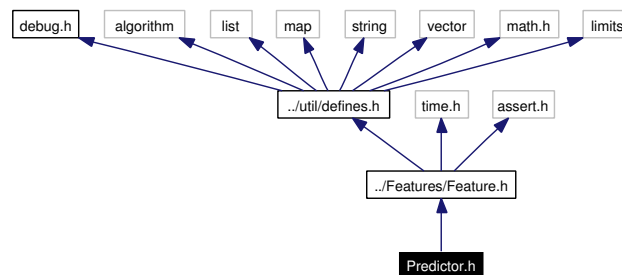
Definition in file [PowerWindows.cpp](#).

## 7.67 Predictor.h File Reference

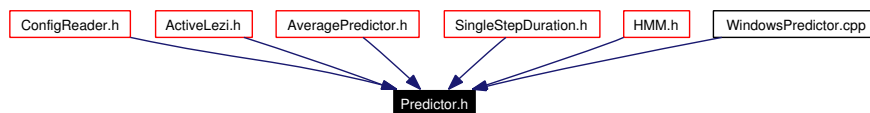
Predictor interface

```
#include "../Features/Feature.h"
```

Include dependency graph for Predictor.h:



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



### Classes

- class [PredictorAlgorithm](#)  
*Predictor Interface.*
- class [Predictor](#)  
*Predictor Wrapper.*

### Defines

- #define [EXPORT\\_PREDICTOR](#)(predictor)  
*Export a predictor.*

### Typedefs

- typedef [parametermap](#) [predictorparams](#)  
*Parameters passed to predictors.*
- typedef [vector](#)< unsigned long > [contexttrajectory](#)
- typedef [PredictorAlgorithm](#) \*(\* [getPredictor\\_t](#))(const [predictorparams](#) &params)  
*Type of the getPredictor function.*

## 7.67.1 Detailed Description

Predictor interface

©2003-2004 by Rene Mayrhofer, Harald Radi

### Date

2004/02/12 20:08:37

### Revision

1.11

### Since:

Wed Mar 19 2003

### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

### Id

[Predictor.h](#),v 1.11 2004/02/12 20:08:37 harald Exp

Definition in file [Predictor.h](#).

## 7.67.2 Define Documentation

### 7.67.2.1 #define EXPORT\_PREDICTOR(predictor)

#### Value:

```
\
static predictor *pred; \
void INIT_EXPORT library_initialize() { pred = NULL; } \
void FINI_EXPORT library_finalize() { if (pred != NULL) delete pred; } \ \
extern "C" PREDICTOR_EXPORT PredictorAlgorithm* getPredictor(predictorparams &params) { if (pred == NU
```

Export a predictor.

This macro has to be used in global scope and gets substituted with the implementation for exporting a [PredictorAlgorithm](#) singleton instance to the framework.

#### Parameters:

***predictor*** The Class that implements the [PredictorAlgorithm](#) interface and should be exported to the framework.

Definition at line 52 of file Predictor.h.

### 7.67.3 Typedef Documentation

#### 7.67.3.1 typedef [vector](#)<[unsigned long](#)> [contexttrajectory](#)

##### [Todo](#)

documentation

Definition at line 75 of file Predictor.h.

Referenced by `PredWrapper::predictAndUpdateMultiStep()`.

#### 7.67.3.2 typedef [PredictorAlgorithm](#)\*([getPredictor\\_t](#))(const [predictorparams](#) &params)

Type of the `getPredictor` function.

This type definition is needed by the [Predictor](#) Wrapper to query a [PredictorAlgorithm](#) implementation for an instance of its class. Definition at line 181 of file Predictor.h.

Referenced by `Predictor::Predictor()`.

#### 7.67.3.3 typedef [parametermap](#) [predictorparams](#)

Parameters passed to predictors.

When initializing a predictor, an arbitrary number of named (string) parameters can be passed from the application (e.g. read from a config file). These key-value tuples can then be used by the predictor as configuration or initialization values. They are completely predictor specific.

##### See also:

[Predictor](#)

[PredictorAlgorithm](#)

Definition at line 72 of file Predictor.h.

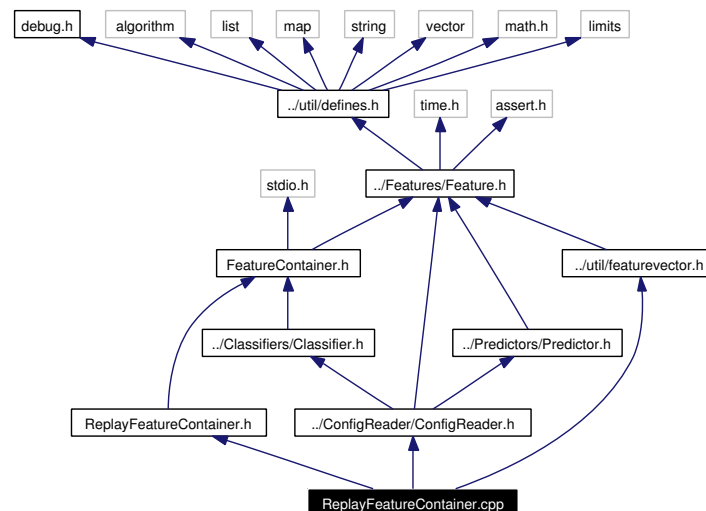
Referenced by `main()`, and `Predictor::Predictor()`.

## 7.68 ReplayFeatureContainer.cpp File Reference

Feature container

```
#include "ReplayFeatureContainer.h"
#include "../ConfigReader/ConfigReader.h"
#include "../util/featurevector.h"
```

Include dependency graph for ReplayFeatureContainer.cpp:



### 7.68.1 Detailed Description

Feature container

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.22

#### Since:

Thu Apr 10 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

#### Id

[ReplayFeatureContainer.cpp](#), v 1.22 2004/02/12 20:08:37 harald Exp

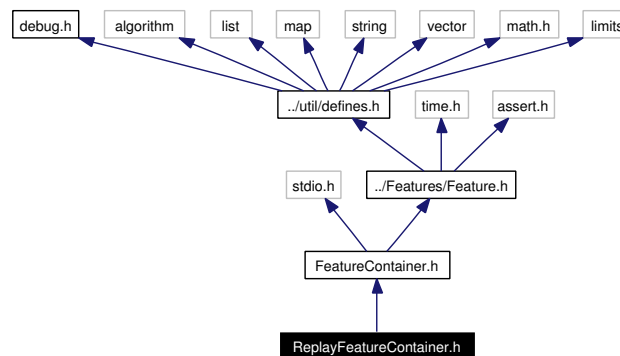
Definition in file [ReplayFeatureContainer.cpp](#).

## 7.69 ReplayFeatureContainer.h File Reference

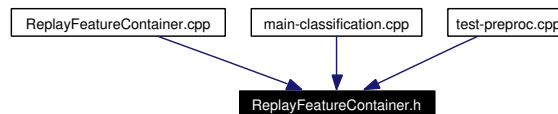
Feature container

```
#include "FeatureContainer.h"
```

Include dependency graph for ReplayFeatureContainer.h:



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



### Classes

- class [ReplayFeatureContainer](#)

### 7.69.1 Detailed Description

Feature container

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.13

#### Since:

Thu Apr 10 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>



This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

**Id**

[ReplayFeatureContainer.h](#),v 1.13 2004/02/12 20:08:37 harald Exp

Definition in file [ReplayFeatureContainer.h](#).

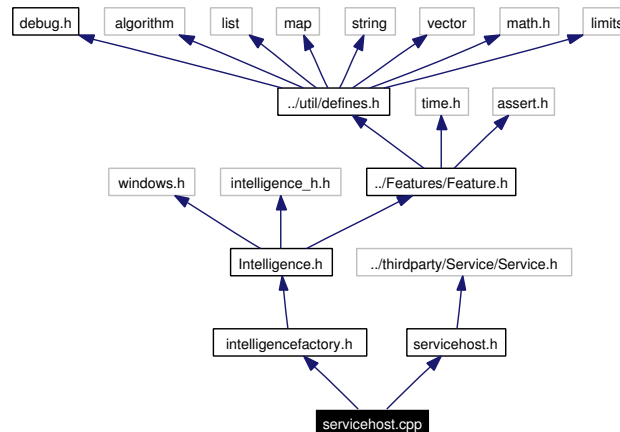
## 7.70 servicehost.cpp File Reference

Windows NT service.

```
#include "intelligencefactory.h"
```

```
#include "servicehost.h"
```

Include dependency graph for servicehost.cpp:



### 7.70.1 Detailed Description

Windows NT service.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/21 22:58:58

#### Revision

1.11

#### Since:

Tue Aug 8 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[servicehost.cpp](#),v 1.11 2004/02/21 22:58:58 harald Exp

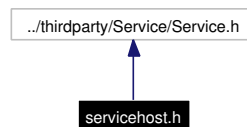
Definition in file [servicehost.cpp](#).

## 7.71 servicehost.h File Reference

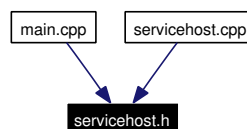
Windows NT service.

```
#include "../thirdparty/Service/Service.h"
```

Include dependency graph for servicehost.h:



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



### Classes

- class [ServiceHost](#)

### 7.71.1 Detailed Description

Windows NT service.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/21 22:58:58

#### Revision

1.5

#### Since:

Tue Aug 8 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[servicehost.h](#),v 1.5 2004/02/21 22:58:58 harald Exp

Definition in file [servicehost.h](#).

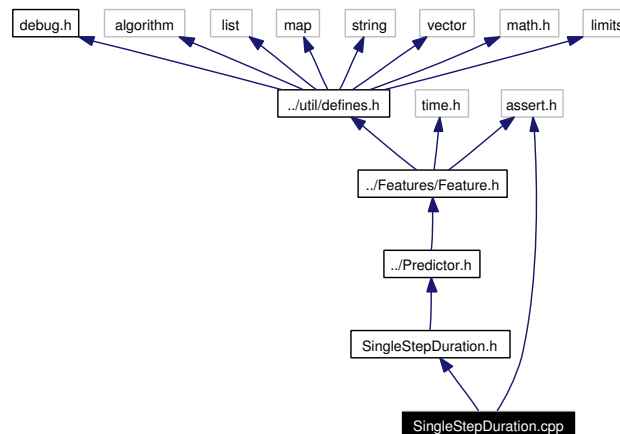
## 7.72 SingleStepDuration.cpp File Reference

©2003-2004 by Rene Mayrhofer, Harald Radi

```
#include "SingleStepDuration.h"
```

```
#include <assert.h>
```

Include dependency graph for SingleStepDuration.cpp:



### Defines

- #define [DEFAULT\\_SMOOTHING\\_WINDOW\\_WIDTH](#) 4
- #define [DEFAULT\\_SMOOTHING\\_FACTOR](#) 0.8
- #define [DEFAULT\\_PEAK\\_THRESHOLD](#) 0.01

### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [PredictorAlgorithm](#) \* [getPredictor](#) ([predictorparams](#) &params)  
*\ Returns an instance of a class implementing the [PredictorAlgorithm](#) interface \ \ param params A map of parameters to initialize the instance \ return A [PredictorAlgorithm](#) implementation \*

### Variables

- [SingleStepDurationPredictor](#) \* [pred](#)  
*Singleton implementing the [PredictorAlgorithm](#) interface.*

### 7.72.1 Detailed Description

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/23 16:27:24

**Revision**

1.11

**Since:**

Wed Mar 19 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[SingleStepDuration.cpp](#),v 1.11 2004/02/23 16:27:24 rene Exp

Definition in file [SingleStepDuration.cpp](#).

### 7.72.2 Define Documentation

#### 7.72.2.1 #define DEFAULT\_PEAK\_THRESHOLD 0.01

**Todo**

documentation

Definition at line 28 of file [SingleStepDuration.cpp](#).

Referenced by [SingleStepDurationPredictor::SingleStepDurationPredictor\(\)](#).

#### 7.72.2.2 #define DEFAULT\_SMOOTHING\_FACTOR 0.8

**Todo**

documentation

Definition at line 27 of file [SingleStepDuration.cpp](#).

Referenced by [SingleStepDurationPredictor::SingleStepDurationPredictor\(\)](#).

#### 7.72.2.3 #define DEFAULT\_SMOOTHING\_WINDOW\_WIDTH 4

**Todo**

documentation

Definition at line 26 of file [SingleStepDuration.cpp](#).

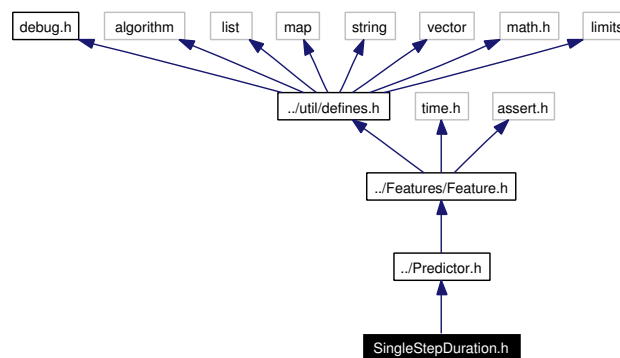
Referenced by [SingleStepDurationPredictor::SingleStepDurationPredictor\(\)](#).

## 7.73 SingleStepDuration.h File Reference

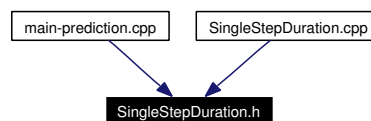
A simple Predictor which tries to predict the duration of staying in the current state.

```
#include "../Predictor.h"
```

Include dependency graph for SingleStepDuration.h:



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



### Classes

- struct [next\\_context\\_description](#)
- class [SingleStepDurationPredictor](#)
- struct [SingleStepDurationPredictor::contextinfo](#)

### Typedefs

- typedef [map](#)< unsigned long, [next\\_context\\_description](#) > [membershipelist\\_duration](#)

#### 7.73.1 Detailed Description

A simple Predictor which tries to predict the duration of staying in the current state.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/23 11:25:16

#### Revision

1.9

**Since:**

Wed Mar 19 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[SingleStepDuration.h](#),v 1.9 2004/02/23 11:25:16 rene Exp

Definition in file [SingleStepDuration.h](#).

## 7.73.2 Typedef Documentation

### 7.73.2.1 typedef [map](#)<unsigned long, [next\\_context\\_description](#)> [membershiplist\\_duration](#)

**Todo**

documentation

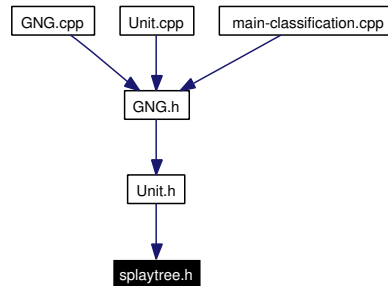
Definition at line 41 of file SingleStepDuration.h.

Referenced by SingleStepDurationPredictor::getContextTrajectory().

## 7.74 splaytree.h File Reference

Splaytree template.

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



### Classes

- struct [node< comparable >](#)
- class [splaytree< comparable >](#)
- class [splaytree\\_iterator< comparable >](#)

### 7.74.1 Detailed Description

Splaytree template.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:39

#### Revision

1.6

#### Since:

Tue Oct 7 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[splaytree.h](#),v 1.6 2004/02/12 20:08:39 harald Exp

Definition in file [splaytree.h](#).



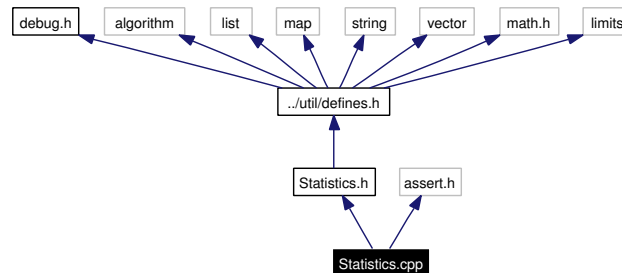
## 7.75 Statistics.cpp File Reference

Predictor interface

```
#include "Statistics.h"
```

```
#include <assert.h>
```

Include dependency graph for Statistics.cpp:



### 7.75.1 Detailed Description

Predictor interface

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/03/02 15:56:08

**Revision**

1.9

**Since:**

Wed Mar 19 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[Statistics.cpp](#),v 1.9 2004/03/02 15:56:08 rene Exp

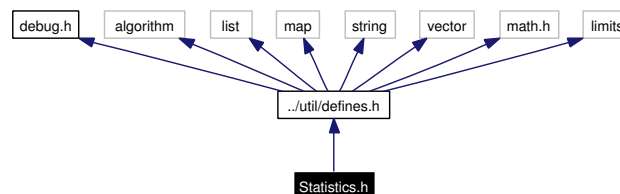
Definition in file [Statistics.cpp](#).

## 7.76 Statistics.h File Reference

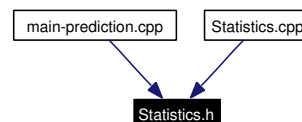
[Statistics](#) of time series.

```
#include "../util/defines.h"
```

Include dependency graph for Statistics.h:



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



### Classes

- class [Statistics](#)

*This class can be used to compute statistics on a `_single_` numerical continuous variable.*

### 7.76.1 Detailed Description

[Statistics](#) of time series.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/02 15:56:08

#### Revision

1.9

#### Since:

Wed Mar 19 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[Statistics.h](#), v 1.9 2004/03/02 15:56:08 rene Exp

Definition in file [Statistics.h](#).

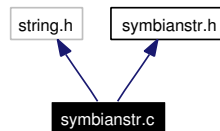
## 7.77 symbianstr.c File Reference

Missing Symbian functions implementation.

```
#include <string.h>
```

```
#include "symbianstr.h"
```

Include dependency graph for symbianstr.c:



### Functions

- `wchar_t * wcsncat (wchar_t *dst, const wchar_t *src, size_t count)`  
*wcsncat implementation*
- `wchar_t * wcsncpy (wchar_t *dst, const wchar_t *src, size_t count)`  
*wcsncpy implementation*

### 7.77.1 Detailed Description

Missing Symbian functions implementation.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Since:

Tue Nov 11 2003

#### Date

2004/01/15 13:06:35

#### Version

1.2

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

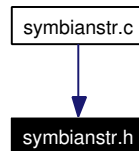
[symbianstr.c](#),v 1.2 2004/01/15 13:06:35 harald Exp

Definition in file [symbianstr.c](#).

## 7.78 symbianstr.h File Reference

Missing Symbian functions declaration.

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



### Symbian OS Compatibility Functions

- `wchar_t * wcsncat (wchar_t *, const wchar_t *, size_t)`  
*wcsncat implementation*
- `wchar_t * wcsncpy (wchar_t *, const wchar_t *, size_t)`  
*wcsncpy implementation*

#### 7.78.1 Detailed Description

Missing Symbian functions declaration.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/12 20:08:39

**Revision**

1.3

**Since:**

Tue Nov 11 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[symbianstr.h](#),v 1.3 2004/02/12 20:08:39 harald Exp

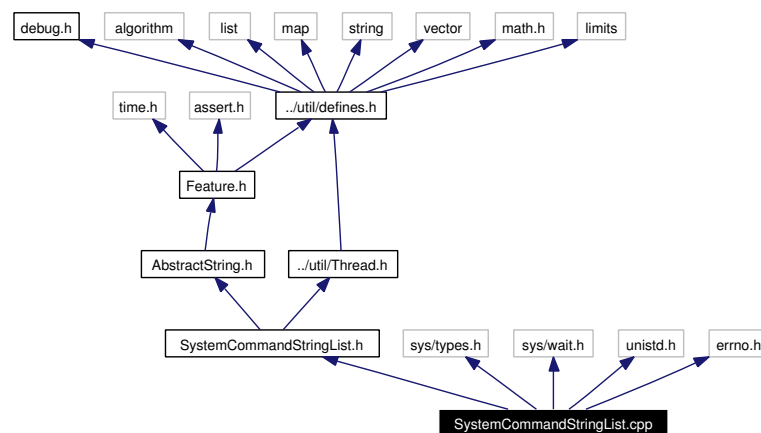
Definition in file [symbianstr.h](#).

## 7.79 SystemCommandStringList.cpp File Reference

System command string feature class implementation.

```
#include "SystemCommandStringList.h"
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <errno.h>
```

Include dependency graph for SystemCommandStringList.cpp:



### Defines

- #define `SYSTEMCOMMAND_DELAY` 15  
Default delay (in seconds) between two system calls.

### 7.79.1 Detailed Description

System command string feature class implementation.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/06/08 09:37:19

#### Revision

1.15

#### Since:

Mon Jul 7 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>  
Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

**Id**

[SystemCommandStringList.cpp](#),v 1.15 2004/06/08 09:37:19 rene Exp

Definition in file [SystemCommandStringList.cpp](#).

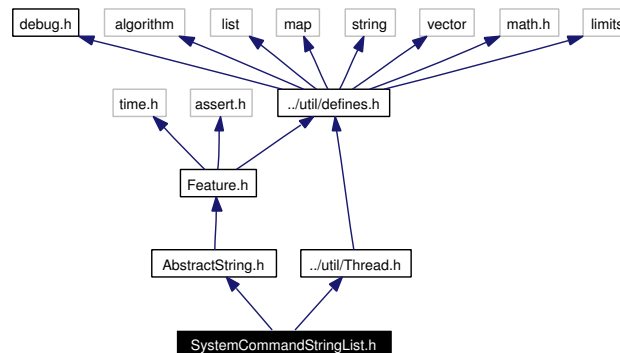
## 7.80 SystemCommandStringList.h File Reference

System command string feature class declaration.

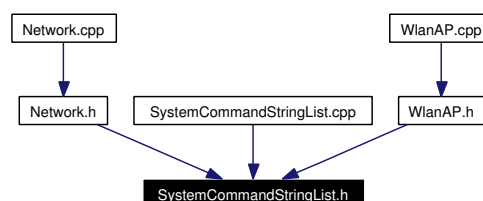
```
#include "AbstractString.h"
```

```
#include "../util/Thread.h"
```

Include dependency graph for SystemCommandStringList.h:



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



### Classes

- class [SystemCommandStringListFeature](#)
- class [SystemCommandStringListFeatureProvider](#)

*This class returns a feature out of any system command that can return a newline-separated list of strings.*

### 7.80.1 Detailed Description

System command string feature class declaration.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/23 11:26:34

#### Revision

1.11



**Since:**

Mon Jul 7 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[SystemCommandStringList.h](#),v 1.11 2004/02/23 11:26:34 rene Exp

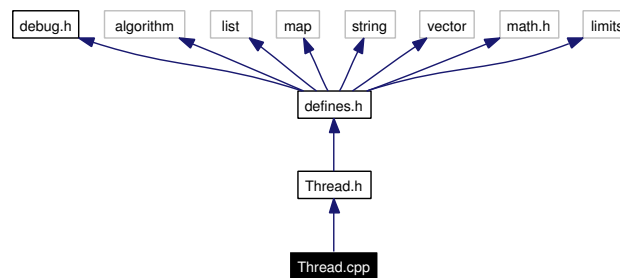
Definition in file [SystemCommandStringList.h](#).

## 7.81 Thread.cpp File Reference

Abstract thread function implementation.

```
#include "Thread.h"
```

Include dependency graph for Thread.cpp:



### 7.81.1 Detailed Description

Abstract thread function implementation.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/23 11:24:40

**Revision**

1.7

**Since:**

Tue May 13 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[Thread.cpp](#),v 1.7 2004/02/23 11:24:40 rene Exp

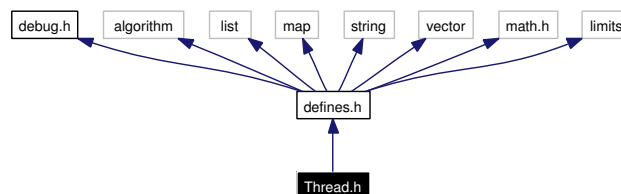
Definition in file [Thread.cpp](#).

## 7.82 Thread.h File Reference

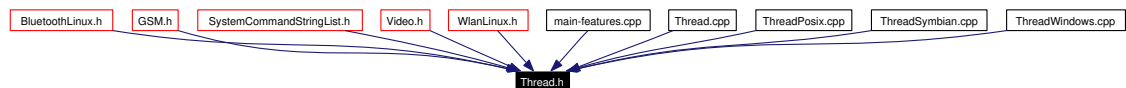
Abstract thread function declaration.

```
#include "defines.h"
```

Include dependency graph for Thread.h:



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



### Classes

- class [Thread](#)  
*Platform independent thread class.*
- class [Thread::ThreadHandle](#)  
*Thread handle*
- class [Thread::Lock](#)  
*Lock.*
- class [Thread::Guard](#)  
*Guard.*

### Defines

- `#define` [THREAD](#) void\*  
*Attributes for thread main callback function.*

#### 7.82.1 Detailed Description

Abstract thread function declaration.

This is a simple, really cross-platform thread class with support for

- Linux
- Win32
- WinCE

It doesn't do much, just offer methods to start a native thread (no cancellation !) and some synchronization. Why has it been written when there are so many other thread toolkits out there with a lot of functionality (e.g. the very nice ZThreads package) ? Because none of them was able to get a thread running in Linux, Win32 \_and\_ WinCE without major porting efforts (major >= 2 days). In short, this class grew out of frustration !

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/23 11:24:40

**Revision**

1.8

**Since:**

Tue May 13 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[Thread.h](#),v 1.8 2004/02/23 11:24:40 rene Exp

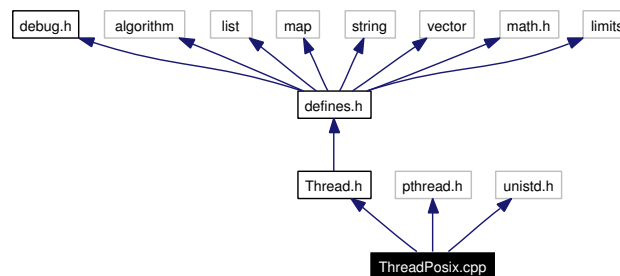
Definition in file [Thread.h](#).

## 7.83 ThreadPosix.cpp File Reference

Abstract thread function implementation.

```
#include "Thread.h"  
#include <pthread.h>  
#include <unistd.h>
```

Include dependency graph for ThreadPosix.cpp:



### 7.83.1 Detailed Description

Abstract thread function implementation.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/23 11:24:40

**Revision**

1.5

**Since:**

Wed May 14 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[ThreadPosix.cpp](#),v 1.5 2004/02/23 11:24:40 rene Exp

Definition in file [ThreadPosix.cpp](#).

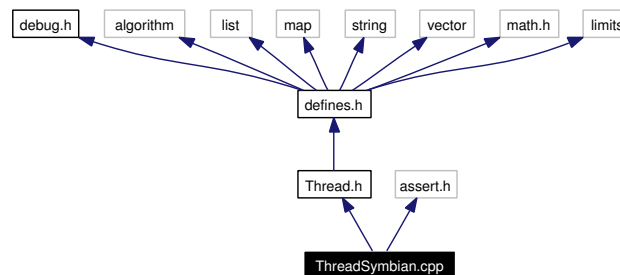
## 7.84 ThreadSymbian.cpp File Reference

Abstract thread function implementation.

```
#include "Thread.h"
```

```
#include <assert.h>
```

Include dependency graph for ThreadSymbian.cpp:



### Defines

- #define [new](#) new(ELeave)

### 7.84.1 Detailed Description

Abstract thread function implementation.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:39

#### Revision

1.4

#### Since:

Fri Nov 28 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[ThreadSymbian.cpp](#),v 1.4 2004/02/12 20:08:39 harald Exp

Definition in file [ThreadSymbian.cpp](#).

## 7.84.2 Define Documentation

### 7.84.2.1 #define new new(ELeave)

#### Todo

documentation

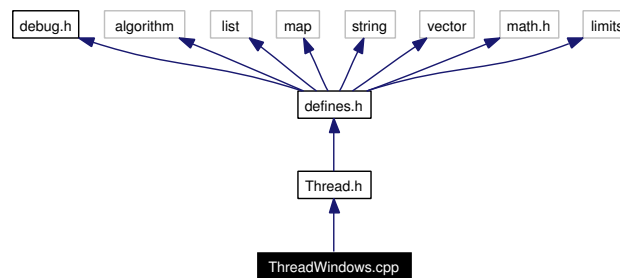
Definition at line 29 of file ThreadSymbian.cpp.

## 7.85 ThreadWindows.cpp File Reference

Abstract thread function implementation.

```
#include "Thread.h"
```

Include dependency graph for ThreadWindows.cpp:



### 7.85.1 Detailed Description

Abstract thread function implementation.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:39

#### Revision

1.8

#### Since:

Tue May 13 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[ThreadWindows.cpp](#),v 1.8 2004/02/12 20:08:39 harald Exp

Definition in file [ThreadWindows.cpp](#).

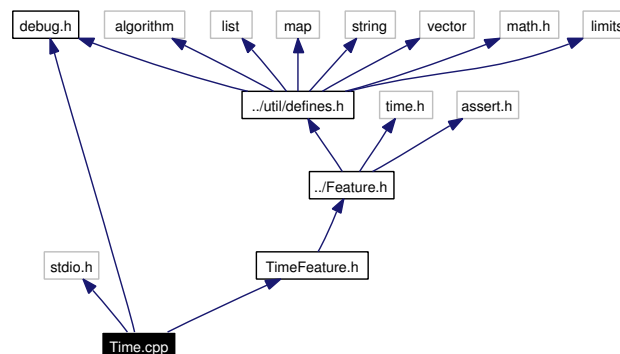


## 7.86 Time.cpp File Reference

Time feature.

```
#include <stdio.h>
#include "TimeFeature.h"
#include "../../util/debug.h"
```

Include dependency graph for Time.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) ([providerparams](#) &[params](#))  
 \ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param [params](#) A map of parameters to initialize the instance \ return A [FeatureProvider](#) implementation \

### Variables

- [TimeFeatureProvider](#) \* [fp](#)  
*Singleton implementing the [FeatureProvider](#) interface.*

### 7.86.1 Detailed Description

Time feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/02 12:36:18

**Revision**

1.45

**Since:**

Wed Apr 9 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

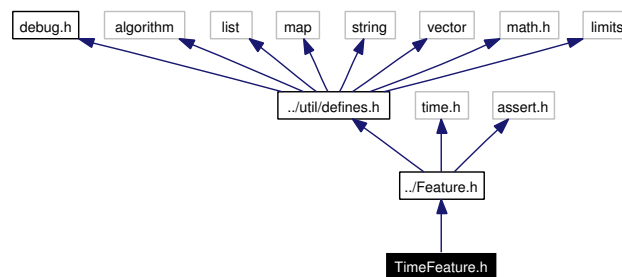
**Id**[Time.cpp](#),v 1.45 2004/03/02 12:36:18 rene-cvs ExpDefinition in file [Time.cpp](#).

## 7.87 TimeFeature.h File Reference

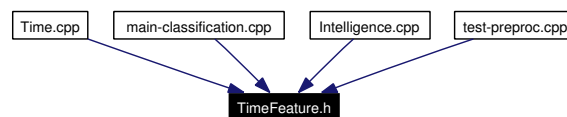
Time feature.

```
#include "../Feature.h"
```

Include dependency graph for TimeFeature.h:



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



### Classes

- class [TimeFeature](#)
- class [TimeFeatureProvider](#)

### 7.87.1 Detailed Description

Time feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/01 15:47:51

#### Revision

1.10

#### Since:

Wed Apr 9 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[TimeFeature.h](#),v 1.10 2004/03/01 15:47:51 rene Exp

Definition in file [TimeFeature.h](#).

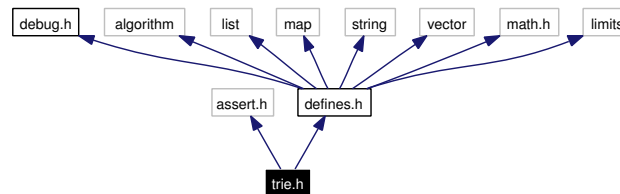
## 7.88 trie.h File Reference

[Trie](#) template.

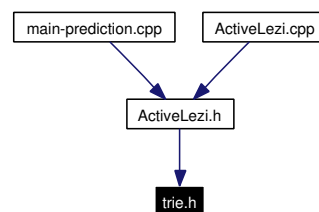
```
#include <assert.h>
```

```
#include "defines.h"
```

Include dependency graph for trie.h:



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



### Classes

- class [Trie< \\_type >](#)
- class [Trie< \\_type >::Node](#)

### 7.88.1 Detailed Description

[Trie](#) template.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:39

#### Revision

1.15

#### Since:

Thu Dec 11 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[trie.h](#),v 1.15 2004/02/12 20:08:39 harald Exp

Definition in file [trie.h](#).

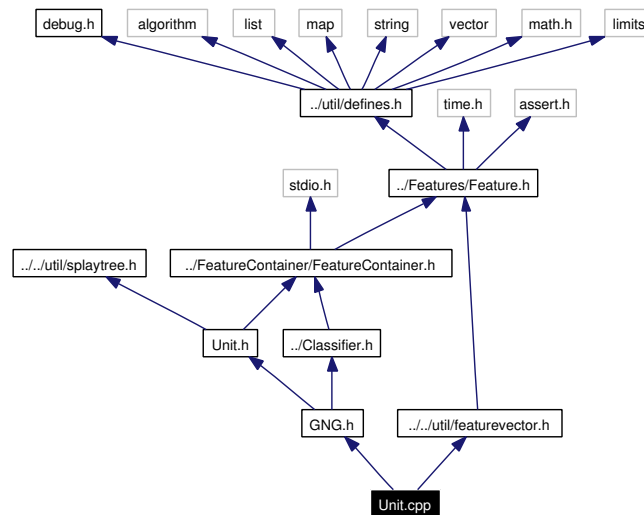
## 7.89 Unit.cpp File Reference

Growing neural gas.

```
#include "GNG.h"
```

```
#include "../util/featurevector.h"
```

Include dependency graph for Unit.cpp:



### 7.89.1 Detailed Description

Growing neural gas.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/12 10:57:21

#### Revision

1.40

#### Since:

Mon Mar 3 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

#### Id

[Unit.cpp](#),v 1.40 2004/03/12 10:57:21 rene Exp

Definition in file [Unit.cpp](#).

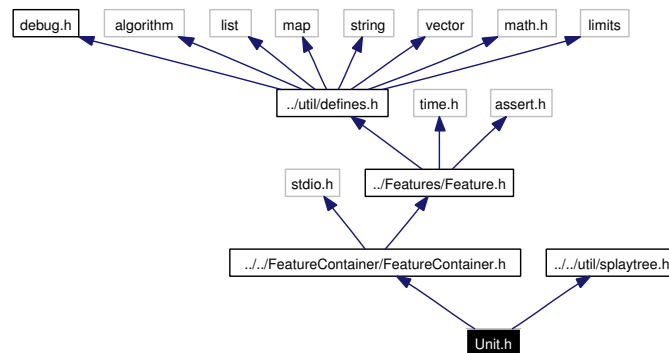
## 7.90 Unit.h File Reference

Growing neural gas.

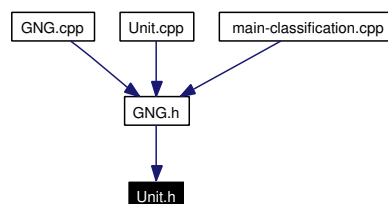
```
#include "../../FeatureContainer/FeatureContainer.h"
```

```
#include "../../util/splaytree.h"
```

Include dependency graph for Unit.h:



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



### Classes

- class [unittree< comparator >](#)
- class [Unit](#)

### Typedefs

- typedef [map< Unit \\*, unsigned long >](#) [edge\\_map](#)

### 7.90.1 Detailed Description

Growing neural gas.

©2003-2004 by Rene Mayrhofer, Harald Radi

### Date

2004/02/12 20:08:37



**Revision**

1.29

**Since:**

Mon Mar 3 2003

**Author:**Rene Mayrhofer <[reene@mayrhofer.eu.org](mailto:reene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**[Unit.h](#),v 1.29 2004/02/12 20:08:37 harald ExpDefinition in file [Unit.h](#).

## 7.90.2 Typedef Documentation

### 7.90.2.1 typedef [map](#)<[Unit](#)\*, unsigned long> [edge\\_map](#)

**Todo**

documentation

Definition at line 50 of file Unit.h.

Referenced by [Unit::getEdges\(\)](#).

## 7.91 version.h File Reference

Windows NT service.

### 7.91.1 Detailed Description

Windows NT service.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/12 20:08:38

**Revision**

1.2

**Since:**

Tue Jan 15 2004

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[version.h](#),v 1.2 2004/02/12 20:08:38 harald Exp

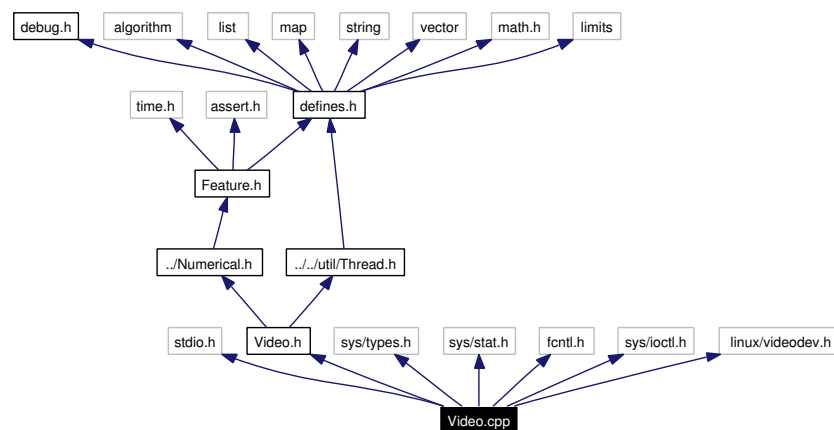
Definition in file [version.h](#).

## 7.92 Video.cpp File Reference

Video feature.

```
#include <stdio.h>
#include "Video.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/videodev.h>
```

Include dependency graph for Video.cpp:



### Defines

- #define **DEFAULT\_WIDTH** 320
- #define **DEFAULT\_HEIGHT** 240
- #define **DEFAULT\_FREQUENCY** 5
- #define **DEFAULT\_DEVICE** "/dev/video0"

### Typedefs

- typedef u\_int64\_t **\_\_u64**

### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*

- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) ([providerparams](#) &params)  
*\ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param params A map of parameters to initialize the instance \ return A [FeatureProvider](#) implementation \*

## Variables

- [VideoFeatureProvider](#) \* fp  
*Singleton implementing the [FeatureProvider](#) interface.*
- int dev\_fd
- int v4l\_fmt
- int v4l\_norm
- int v4l\_input
- int v4l\_bufsize
- int v4l\_curbuffer
- int v4l\_maxbuffer
- unsigned char \* img\_buf
- unsigned char \* v4l\_buffers [2]

### 7.92.1 Detailed Description

Video feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/06/16 15:22:00

#### Revision

1.3

#### Since:

Wed Apr 9 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>  
 Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

Quite some of the code in here has been ripped from various files in the "motion" source code, a tool for detecting motion using webcams.

#### Id

[Video.cpp](#),v 1.3 2004/06/16 15:22:00 rene Exp

Definition in file [Video.cpp](#).

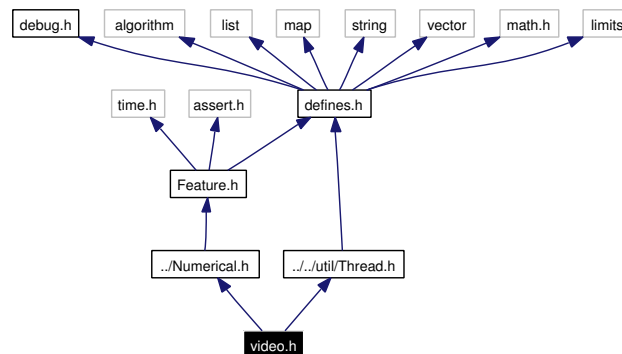
## 7.93 Video.h File Reference

Video feature.

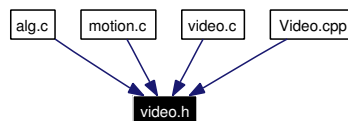
```
#include "../..//util/Thread.h"
```

```
#include "../Numerical.h"
```

Include dependency graph for video.h:



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



### Classes

- class **VideoMotionFeature**
- class [VideoFeatureProvider](#)  
*Provider for the video features.*
- struct [VideoFeatureProvider::SampleData](#)

### 7.93.1 Detailed Description

Video feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/06/16 15:22:00

#### Revision

1.3

#### Since:

Wed Apr 9 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

Video.h,v 1.3 2004/06/16 15:22:00 rene Exp

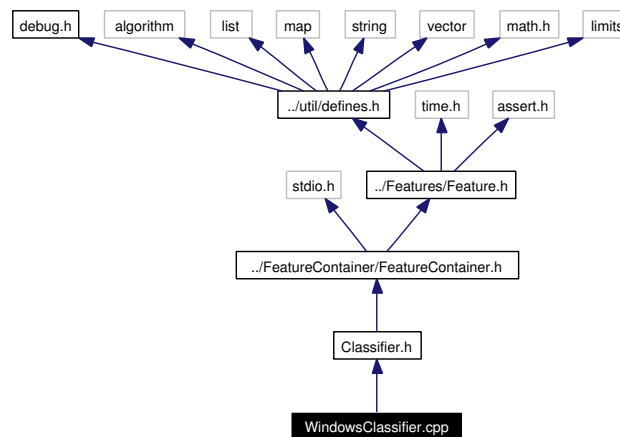
Definition in file [video.h](#).

## 7.94 WindowsClassifier.cpp File Reference

Classifier windows implementation

```
#include "Classifier.h"
```

Include dependency graph for WindowsClassifier.cpp:



### 7.94.1 Detailed Description

Classifier windows implementation

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/02 13:02:00

#### Revision

1.7

#### Since:

Wed Mar 19 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[WindowsClassifier.cpp](#),v 1.7 2004/03/02 13:02:00 rene-cvs Exp

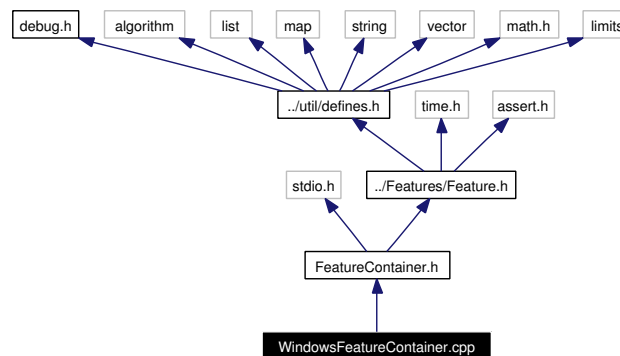
Definition in file [WindowsClassifier.cpp](#).

## 7.95 WindowsFeatureContainer.cpp File Reference

Feature container

```
#include "FeatureContainer.h"
```

Include dependency graph for WindowsFeatureContainer.cpp:



### 7.95.1 Detailed Description

Feature container

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.11

#### Since:

Thu Apr 10 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[WindowsFeatureContainer.cpp](#),v 1.11 2004/02/12 20:08:37 harald Exp

Definition in file [WindowsFeatureContainer.cpp](#).

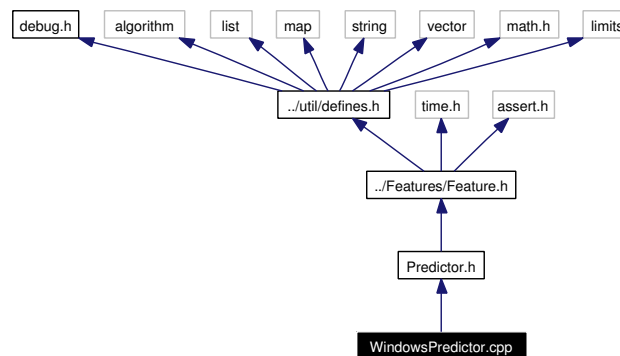


## 7.96 WindowsPredictor.cpp File Reference

Predictor windows implementation

```
#include "Predictor.h"
```

Include dependency graph for WindowsPredictor.cpp:



### 7.96.1 Detailed Description

Predictor windows implementation

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/01/29 10:03:05

**Revision**

1.1

**Since:**

Wed Mar 19 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

**Id**

[WindowsPredictor.cpp](#), v 1.1 2004/01/29 10:03:05 harald Exp

Definition in file [WindowsPredictor.cpp](#).

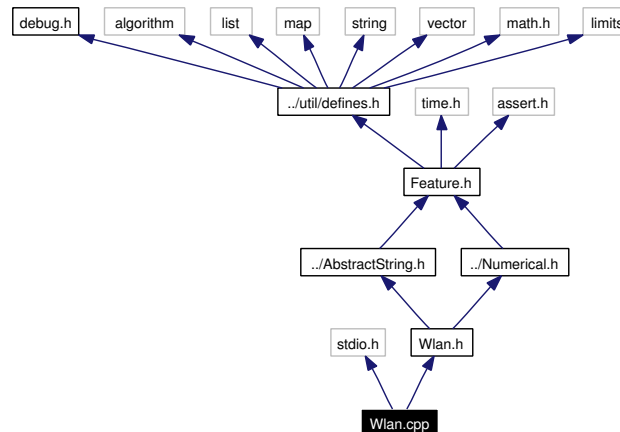
## 7.97 Wlan.cpp File Reference

Wlan feature.

```
#include <stdio.h>
```

```
#include "Wlan.h"
```

Include dependency graph for Wlan.cpp:



### 7.97.1 Detailed Description

Wlan feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/03/02 12:36:18

#### Revision

1.43

#### Since:

Wed Mar 12 2003

#### Author:

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

#### Id

[Wlan.cpp](#),v 1.43 2004/03/02 12:36:18 rene-cvs Exp

Definition in file [Wlan.cpp](#).

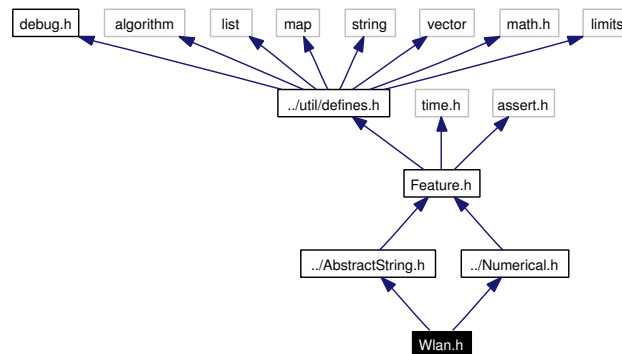
## 7.98 Wlan.h File Reference

Wlan feature.

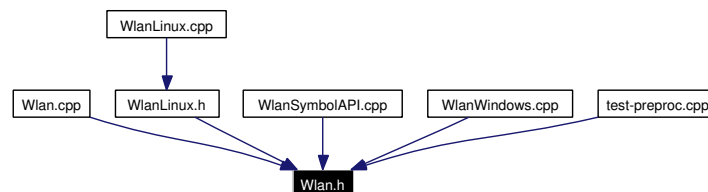
```
#include "../AbstractString.h"
```

```
#include "../Numerical.h"
```

Include dependency graph for Wlan.h:



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



## Classes

- class [WlanActiveEssidFeature](#)

*This feature encapsulates the ESSID which is currently set for a WLAN network interface or the ESSID which it is currently bound to (if not set explicitly).*

- class [WlanActiveMacAddressFeature](#)

*This feature encapsulates the ESSID which is currently set for a WLAN network interface or the ESSID which it is currently bound to (if not set explicitly).*

- class [WlanActiveModeFeature](#)
- class [WlanActiveSignalLevelFeature](#)
- struct [WlanPeerInfo](#)
- class [WlanPeersFeature](#)
- class [WlanNumPeersFeature](#)
- class [WlanFeatureProvider](#)

### 7.98.1 Detailed Description

Wlan feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/03/01 11:15:32

**Revision**

1.33

**Since:**

Wed Mar 12 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[Wlan.h](#),v 1.33 2004/03/01 11:15:32 rene Exp

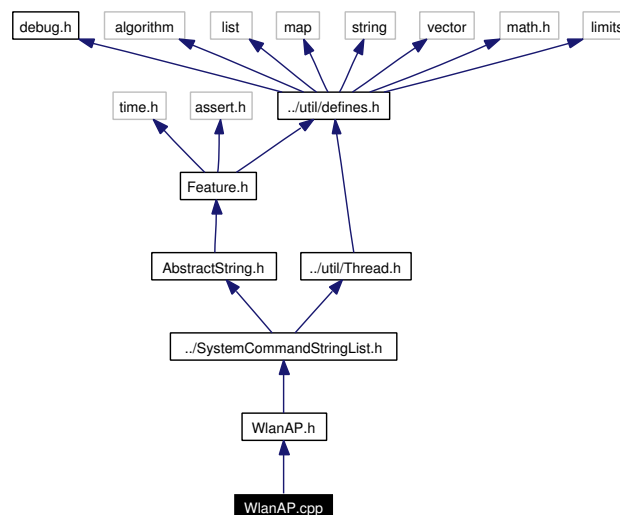
Definition in file [Wlan.h](#).

## 7.99 WlanAP.cpp File Reference

Wlan feature.

```
#include "WlanAP.h"
```

Include dependency graph for WlanAP.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) ([providerparams](#) &params)  
*\ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param params A map of parameters to initialize the instance \ return A [FeatureProvider](#) implementation \*

### Variables

- [WlanAccessPointFeatureProvider](#) \* fp  
*Singleton implementing the [FeatureProvider](#) interface.*

### 7.99.1 Detailed Description

Wlan feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/06/15 19:37:06

**Revision**

1.9

**Since:**

Mon Jul 7 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[WlanAP.cpp](#),v 1.9 2004/06/15 19:37:06 rene Exp

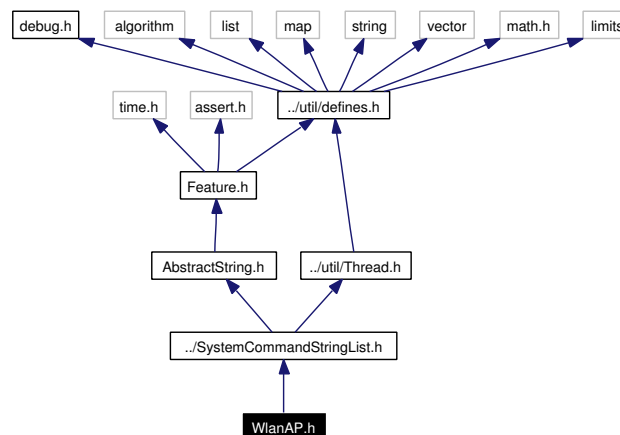
Definition in file [WlanAP.cpp](#).

## 7.100 WlanAP.h File Reference

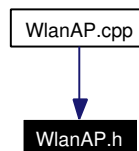
Wlan feature.

```
#include "../SystemCommandStringList.h"
```

Include dependency graph for WlanAP.h:



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



### Classes

- class [WlanAccessPointFeatureProvider](#)

### 7.100.1 Detailed Description

Wlan feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/12 20:08:37

#### Revision

1.8

#### Since:

Mon Jul 7 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**[WlanAP.h](#),v 1.8 2004/02/12 20:08:37 harald ExpDefinition in file [WlanAP.h](#).

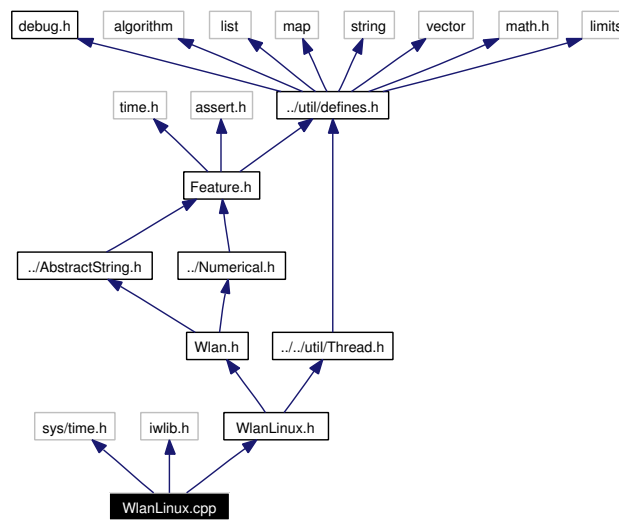


## 7.101 WlanLinux.cpp File Reference

Wlan feature.

```
#include <sys/time.h>
#include <iwlib.h>
#include "WlanLinux.h"
```

Include dependency graph for WlanLinux.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) ([providerparams](#) &params)  
*\ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param params A map of parameters to initialize the instance \ return A [FeatureProvider](#) implementation \*

### Variables

- [WlanLinuxFeatureProvider](#) \* fp  
*Singleton implementing the [FeatureProvider](#) interface.*

#### 7.101.1 Detailed Description

Wlan feature.

Some parts of this code have been taken from iwconfig.c and iwlist.c of the wireless-tools distribution and were modified accordingly. The original license is:

This file is released under the GPL license. Copyright (c) 1997-2002 Jean Tourrilhes  
<[jt@hpl.hp.com](mailto:jt@hpl.hp.com)>

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/06/04 11:37:10

**Revision**

1.24

**Since:**

Wed Mar 19 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[WlanLinux.cpp](#),v 1.24 2004/06/04 11:37:10 rene Exp

Definition in file [WlanLinux.cpp](#).

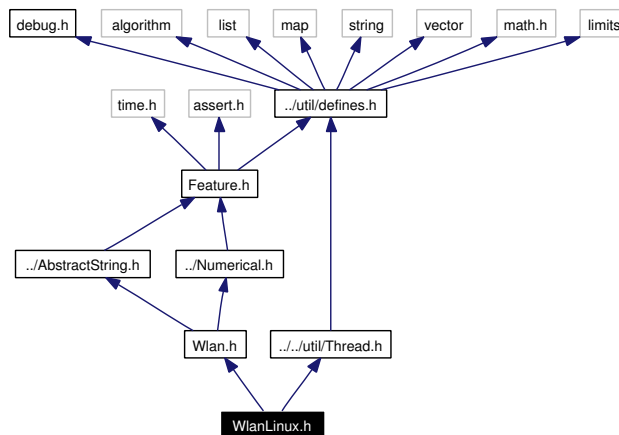
## 7.102 WlanLinux.h File Reference

Wlan feature.

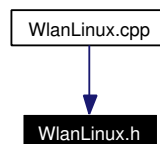
```
#include "Wlan.h"
```

```
#include "../../util/Thread.h"
```

Include dependency graph for WlanLinux.h:



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



### Classes

- class [WlanLinuxFeatureProvider](#)

### 7.102.1 Detailed Description

Wlan feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

#### Date

2004/02/23 11:26:48

#### Revision

1.16

#### Since:

Wed Mar 19 2003

**Author:**Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

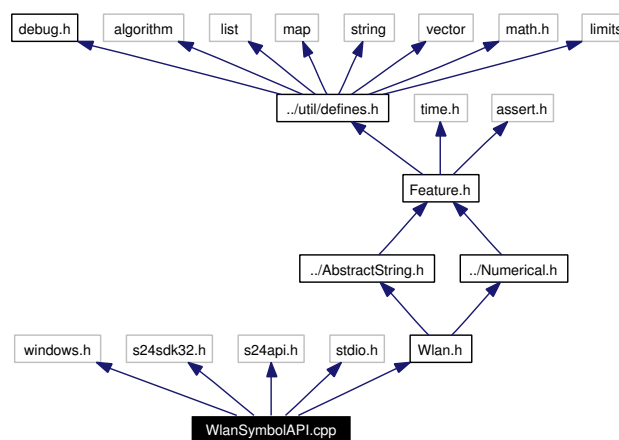
**Id**[WlanLinux.h](#),v 1.16 2004/02/23 11:26:48 rene ExpDefinition in file [WlanLinux.h](#).

## 7.103 WlanSymbolAPI.cpp File Reference

Wlan feature.

```
#include <windows.h>
#include <s24sdk32.h>
#include <s24api.h>
#include <stdio.h>
#include "Wlan.h"
```

Include dependency graph for WlanSymbolAPI.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) ([providerparams](#) &params)  
*\ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param params A map of parameters to initialize the instance \ return A [FeatureProvider](#) implementation \*
- BOOL APIENTRY [DllMain](#) (HANDLE hModule, DWORD fdwReason, LPVOID lpReserved)  
*Library entry point.*

### Variables

- [WlanFeatureProvider](#) \* [fp](#)  
*Singleton implementing the [FeatureProvider](#) interface.*
- HINSTANCE [hLib](#) = NULL

*Driver handle.*

### 7.103.1 Detailed Description

Wlan feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/12 20:08:37

**Revision**

1.12

**Since:**

Wed May 14 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[WlanSymbolAPI.cpp](#),v 1.12 2004/02/12 20:08:37 harald Exp

Definition in file [WlanSymbolAPI.cpp](#).

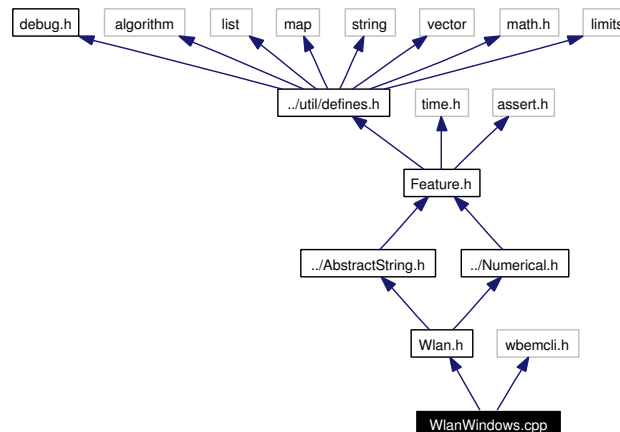
## 7.104 WlanWindows.cpp File Reference

Wlan feature.

```
#include "Wlan.h"
```

```
#include <wbemcli.h>
```

Include dependency graph for WlanWindows.cpp:



### Functions

- void [library\\_initialize](#) ()  
*Load library.*
- void [library\\_finalize](#) ()  
*Unload library.*
- C FEATURE\_EXPORT [FeatureProvider](#) \* [getProvider](#) ([providerparams](#) &params)  
*\ Returns an instance of a class implementing the [FeatureProvider](#) interface \ \ param params A map of parameters to initialize the instance \ return A [FeatureProvider](#) implementation \*
- BOOL WINAPI [DllMain](#) (HANDLE hModule, DWORD fdwReason, LPVOID lpReserved)  
*Library entry point.*

### Variables

- [WlanFeatureProvider](#) \* fp  
*Singleton implementing the [FeatureProvider](#) interface.*

#### 7.104.1 Detailed Description

Wlan feature.

©2003-2004 by Rene Mayrhofer, Harald Radi

**Date**

2004/02/12 20:08:37

**Revision**

1.12

**Since:**

Wed Mar 12 2003

**Author:**

Rene Mayrhofer <[rene@mayrhofer.eu.org](mailto:rene@mayrhofer.eu.org)>

Harald Radi <[harald.radi@nme.at](mailto:harald.radi@nme.at)>

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

**Id**

[WlanWindows.cpp](#),v 1.12 2004/02/12 20:08:37 harald Exp

Definition in file [WlanWindows.cpp](#).



## **Chapter 8**

# **Intelligence Page Documentation**

### **8.1 Introduction**

This is the introduction.

## 8.2 Installation

blah blah

## 8.3 Framework

foobar

## 8.4 Todo List

Member [AbstractStringFeature::getDistance](#)([Feature \\*f](#)) **const** implement a better 'get string for code'

Member [AbstractStringFeature::moveTowards](#)([Feature \\*f](#), **double factor**) implement a better 'get string for code'

Member [AbstractStringFeature::levenshtein](#)(**char \*\*x**, **const char \*t**, **double \*factor**) alloc once

Member [AbstractStringListFeature::getDistance](#)([Feature \\*f](#)) **const** this should not happen, find the reason for that

Class [ActiveLezi](#) documentation

Member [ActiveLezi::dictionary](#) documentation

Member [ActiveLezi::ActiveLezi](#)(**predictorparams &params**) documentation

Member [ActiveLezi::getMaxDepth](#)() **const** documentation

Member [ActiveLezi::getNextContext](#)(**const list< unsigned long > \*context**) **const** documentation

Member [AudioFeatureProvider::AudioFeatureProvider](#)(**providerparams &params**) symbian stl fix

Class [AudioFeatureProvider::SampleData](#) documentation

Class [AveragePredictor](#) documentation

Member [AveragePredictor::AveragePredictor](#)(**predictorparams &params**) documentation

Member [AveragePredictor::windowSize](#) documentation

Member [BluetoothFeatureProvider::peerList](#) move to features

Member [BluetoothFeatureProvider::peersFeature\\_code](#) move to features

Class [BluetoothLinuxFeatureProvider](#) documentation

Member [BluetoothLinuxFeatureProvider::BluetoothLinuxFeatureProvider\(featureparams &params\)](#)

check parameter !!

check parameter !!

check parameter !!

check parameter !!

check parameter !!

Class [BluetoothLinuxFeatureProvider::L2Connection](#) documentation

Member [ConfigReader::getClassifier\(\)](#) documentation

source cleanup (don't copy maps)

Member [FeatureContainer::featurepair](#) documentation

Member [FeatureContainer::persist](#) documentation

Class [GNG](#) documentation

Member [GNG::lambda](#) documentation

Member [GNG::getMetaClusters\(\)](#) THIS MUST WORK !!!!

Member [GNG::getNumberNodes\(\)](#) const documentation

Member [GNG::getWinnerDistance\(const featurevector \\*sample, unsigned int &cluster\\_id, unsigned int &meta\\_cluster\)](#) documentation

Member [GNG::nextSample\(const featurevector \\*sample\)](#) cope with increasing dimension of the feature vector ?

Member [GNG::nextSample\(const featurevector \\*sample\)](#) this should be done in a better way

Class [GSMCellFeature](#) documentation

Class [GSMFeatureProvider](#) documentation

Member [GSMFeatureProvider::cellId](#) documentation

Member [GSMFeatureProvider::init\(featureparams &params\)](#) check parameter !!

Member [GSMFeatureProvider::features](#) documentation

Member [GSMFeatureProvider::GSMFeatureProvider\(providerparams &params\)](#) check parameter !!

Class [HMM](#) documentation

Member [HMM::numHiddenStates](#) documentation

Class [LoggingFeatureContainer](#) documentation

Class [meta\\_cluster](#) documentation

Member [meta\\_cluster::cluster\\_id](#) documentation

Class [NetworkFeatureProvider](#) documentation

Member [SystemCommandStringListFeatureProvider::stringList](#) documentation

Member [NetworkFeatureProvider::features](#) documentation

Member [NetworkFeatureProvider::NetworkFeatureProvider\(featureparams &params\)](#) documentation

Member [NetworkFeatureProvider::NetworkFeatureProvider\(featureparams &params\)](#) check parameter !!

Member [SystemCommandStringListFeatureProvider::initialize\(string name\)](#) documentation

Class [next\\_context\\_description](#) documentation

Member [next\\_context\\_description::duration](#) documentation

Class [node< comparable >](#) documentation

Member [node::left](#) documentation

Class [PersistantReader](#) documentation

Member [PersistantReader::getPersistantData\(string feature\)](#) documentation

Member [PersistantReader::data](#) name type

Class [PersistantWriter](#) documentation

Member [PersistantWriter::write\(const char \\*file, map< string, featureparams > persistdata\)](#)  
documentation

Class [PHPWrapperFeatureProvider](#) documentation

Member [PHPWrapperFeatureProvider::PHPWrapperFeatureProvider\(providerparams &params\)](#)  
documentation

Class [PowerFeature](#) documentation

Member [PowerFeature::moveTowards\(Feature \\*f, double factor\)](#) can we do this more deterministic ?

Class [PowerFeatureProvider](#) documentation

Member [PowerFeatureProvider::plugged](#) documentation

Member [PredictorAlgorithm::getContextTrajectory\(unsigned int start, unsigned int end\) const =0](#)  
documentation

Class [PredictorAlgorithm](#) documentation

Member [PredictorAlgorithm::addContext\(unsigned long contextId, time\\_t time\)=0](#) documentation

Member [PredictorAlgorithm::addContexts\(const membershiplist \\*contexts, time\\_t time\)=0](#)  
documentation

Member [PredictorAlgorithm::getContextAt\(time\\_t time\) const =0](#) documentation

Member [PredictorAlgorithm::getContextsAt\(time\\_t time\) const =0](#) documentation

Member [PredictorAlgorithm::getNextContext\(\) const =0](#) documentation

Member [PredictorAlgorithm::getNextContexts\(\) const =0](#) documentation

Class [PredWrapper](#) documentation

Member [PredWrapper::pred](#) documentation

Member [PredWrapper::PredWrapper\(PredictorAlgorithm \\*pred\)](#) documentation

Member [PredWrapper::getAccuracy\(\)](#) documentation

Member [PredWrapper::getPred\(\)](#) documentation

Member [PredWrapper::predictAndUpdate\(unsigned long curContext\)](#) documentation

Member [PredWrapper::predictAndUpdateMultiStep\(vector< unsigned int > \\*values, unsigned int curPos, unsigned int curContext\)](#) documentation

Class [ReplayFeatureContainer](#) documentation

Member [ReplayFeatureContainer::in](#) documentation

Member [ReplayFeatureContainer::nextSample\(\)](#) care about init lines !

Class [Scanner](#) documentation

Class [ServiceHost](#) documentation

Class [SingleStepDurationPredictor](#) documentation

Member [SingleStepDurationPredictor::durationfrequencies](#) documentation



Member [SingleStepDurationPredictor::frequencies](#) documentation

Member [SingleStepDurationPredictor::SingleStepDurationPredictor\(predictorparams &params\)](#)  
documentation

Member [SingleStepDurationPredictor::SingleStepDurationPredictor\(predictorparams &params\)](#)  
check parameter !!  
check parameter !!  
check parameter !!

Member [SingleStepDurationPredictor::getContextTrajectory\(unsigned int start, unsigned int end\) const](#)  
this has to be enhanced

Member [SingleStepDurationPredictor::getCurrentContextDuration\(\) const](#) documentation

Member [SingleStepDurationPredictor::getNextContextsWithDuration\(unsigned int offset=0\) const](#)  
documentation

Member [SingleStepDurationPredictor::getNextContextsWithDuration\(unsigned int offset=0\) const](#)  
if this turns out to be too inefficient, cache it in the normal data structures

Class [SingleStepDurationPredictor::contextinfo](#) documentation

Member [SingleStepDurationPredictor::contextinfo::transitions](#) documentation

Class [splaytree< comparable >](#) documentation

Member [splaytree::iterator](#) documentation

Member [splaytree::sort\(\)](#) implement splaytree sorting

Member [splaytree::ITEM\\_NOT\\_FOUND](#) documentation

Class [splaytree\\_iterator< comparable >](#) documentation

Member [splaytree\\_iterator::splaytree\\_iterator\(\)](#) documentation

Member [splaytree\\_iterator::iterations](#) documentation

Class [SystemCommandStringListFeature](#) documentation

Member [SystemCommandStringListFeature::SystemCommandStringListFeature](#)(stringcode \*code, long \*maxlen, con documentation

Member [SystemCommandStringListFeature::SystemCommandStringListFeature](#)(stringcode \*code, long \*maxlen, con documentation

Member [SystemCommandStringListFeature::featureName](#) documentation

Member [SystemCommandStringListFeature::providerName](#) documentation

Member [SystemCommandStringListFeatureProvider::providerName](#) documentation

Member [SystemCommandStringListFeatureProvider::SystemCommandStringListFeatureProvider](#)(string featureProvi check parameter

Member [Thread::Lock::Lock](#)() leaking 24 bytes

Member [Thread::ThreadHandle::ThreadHandle](#)() leaking 4 bytes

Class [TimeFeature](#) documentation

Member [TimeFeature::minval](#) documentation

Member [TimeFeature::TimeFeature](#)(FeatureKind kind, time\_t \*minval, time\_t \*maxval) documentation

Member [TimeFeature::TimeFeature](#)(FeatureKind kind, time\_t \*minval, time\_t \*maxval, time\_t timestamp) documentation

Class [TimeFeatureProvider](#) documentation

Member [TimeFeatureProvider::timeFeature\\_maxval](#) documentation

Member [TimeFeatureProvider::TimeFeatureProvider](#)(providerparams &params) documentation

Class [Trie<\\_type>](#) documentation

Member [Trie::Trie\(\)](#) documentation

Member [Trie::~~Trie\(\)](#) documentation

Member [Trie::calculateProbability\(const list< \\_type > \\*context, const \\_type nextValue\)](#) const  
documentation

Member [Trie::findSequence\(const list< \\_type > \\*sequence\)](#) const documentation

Member [Trie::insertSequence\(const list< \\_type > \\*sequence, bool updateFrequency=false\)](#)  
documentation

Member [Trie::navigateTree\(const list< \\_type > \\*sequence\)](#) const documentation

Member [Trie::serialize\(\)](#) const documentation

Member [Trie::toString\(\)](#) const documentation

Member [Trie::unserialize\(string data\)](#) documentation

Member [Trie::root](#) documentation

Class [Trie< \\_type >::Node](#) documentation

Member [Trie::Node::value](#) documentation

Member [Trie::Node::Node\(\)](#) documentation

Member [Trie::Node::Node\(const \\_type val, const Node \\*parent\)](#) documentation

Member [Trie::Node::toString\(unsigned int level\)](#) const documentation

Class [Unit](#) documentation

Member [Unit::all\\_units](#) documentation

Member [Unit::Unit\(unit\\_list const \\*const units, const featurevector pos\)](#) documentation

Member **Unit::Unit**(unit\_list const \*const units, const featurevector pos, unsigned long id) documentation

Member **Unit::addEdge**(Unit \*u) documentation

Member **Unit::ageEdge**(Unit \*u) documentation

Member **Unit::ageEdges**() documentation

Member **Unit::ageEdges**() optimize here ! we will have splits with new meta IDs just before removing those newly split units again (in **GNG::nextSample**) !

Member **Unit::calculateLocalSimilarity**() const documentation

Member **Unit::clearMetaIds**() documentation

Member **Unit::decreaseInsertionThreshold**() documentation

Member **Unit::decreaseYouth**() documentation

Member **Unit::getDistance**(const featurevector \*sample) const documentation

Member **Unit::getDistance**(const featurevector \*sample) const implement the distance from heterogenous kohonen !

Member **Unit::getEdgeCount**() const documentation

Member **Unit::getEdges**() documentation

Member **Unit::getInsertionError**() const documentation

Member **Unit::getInsertionThreshold**() const documentation

Member **Unit::getLearningRate**(bool isWinner) const documentation

Member **Unit::getLearningRate**(bool isWinner) const implement getActivation()

Member **Unit::getLocalSimilarity()** **const** documentation

Member **Unit::getLongTermError()** **const** documentation

Member **Unit::getMaxMetaId()** documentation

Member **Unit::getMetaId()** **const** documentation

Member **Unit::getNumMetaIds()** documentation

Member **Unit::getPosition()** **const** documentation

Member **Unit::getQualityInsertion()** **const** documentation

Member **Unit::getQualityLearning()** **const** documentation

Member **Unit::getShortTermError()** **const** documentation

Member **Unit::getYouth()** **const** documentation

Member **Unit::moveTowards(const featurevector \*sample)** documentation

Member **Unit::operator!=(Unit \*u)** documentation

Member **Unit::operator<(Unit \*u)** documentation

Member **Unit::operator==(Unit \*u)** documentation

Member **Unit::operator>(Unit \*u)** documentation

Member **Unit::releaseMetaId(unsigned long meta\_id)** documentation

Member **Unit::removeEdge(Unit \*u)** documentation

Member **Unit::reserveMetaId(unsigned long meta\_id)** documentation

Member [Unit::serialize\(\)](#) **const** documentation

Member [Unit::setEdgeAge\(\)](#) ([Unit](#) \*u, unsigned long age) documentation

Member [Unit::setInsertionError\(\)](#) (double error) documentation

Member [Unit::setInsertionThreshold\(\)](#) (double threshold) documentation

Member [Unit::setLongTermError\(\)](#) (double error) documentation

Member [Unit::setShortTermError\(\)](#) (double error) documentation

Member [Unit::setYouth\(\)](#) (double youth) documentation

Member [Unit::splitUnit\(\)](#) documentation

Member [Unit::splitUnit\(\)](#) According to the paper, the quality should be `_larger_` then zero. Check why `getQualityInsertion` can return a zero quality for the neighbors of the winner (!!!) and revert this check to `cur_qual > quality` ! UPDATE: `getQualityInsertion` can of course return zero for the first two units, when `error_short` and `error_long` are both zero. Hmm, check the paper again.  
is this OK ?????

Member [Unit::unserialize\(\)](#) (const [featurevector](#) &features, const string data) documentation

Member [Unit::updateErrors\(\)](#) (const [featurevector](#) \*sample) documentation

Member [Unit::updateInsertionThreshold\(\)](#) documentation

Class [unittree](#) < [comparator](#) > documentation

Member [VideoFeatureProvider::VideoFeatureProvider\(\)](#) (providerparams &params) symbian stl fix  
check parameter !!  
check parameter !!  
check parameter !!  
check parameter !!

Class [VideoFeatureProvider::SampleData](#) documentation

Class [WlanAccessPointFeatureProvider](#) documentation

Member [WlanAccessPointFeatureProvider::features](#) documentation

Member [WlanAccessPointFeatureProvider::WlanAccessPointFeatureProvider\(featureparams &params\)](#)  
documentation

Member [WlanAccessPointFeatureProvider::WlanAccessPointFeatureProvider\(featureparams &params\)](#)  
check parameter !!

Member [WlanActiveEssidFeature::WlanActiveEssidFeature\(stringcode \\*code, long \\*maxlen, const string essid\)](#)  
documentation

Class [WlanActiveModeFeature](#) documentation

Member [WlanActiveModeFeature::FeatureKind](#) documentation

Member [WlanActiveModeFeature::mode](#) documentation

Class [WlanActiveSignalLevelFeature](#) documentation

Class [WlanFeatureProvider](#) documentation

Member [WlanFeatureProvider::activePeer](#) documentation

Member [WlanFeatureProvider::features](#) documentation

Member [WlanFeatureProvider::WlanFeatureProvider\(providerparams &params\)](#) documentation

Member [WlanFeatureProvider::WlanFeatureProvider\(providerparams &params\)](#) provide a sym-  
bian limits header file

Member [WlanFeatureProvider::FeatureKind](#) documentation

Member [WlanFeatureProvider::nextSample\(clock\\_t checkpoint\)](#) clean up

Class [WlanLinuxFeatureProvider](#) documentation

Member [WlanLinuxFeatureProvider::initSocket\(string interfaceName\)](#) documentation

Member [WlanLinuxFeatureProvider::ifname](#) documentation

Member [WlanLinuxFeatureProvider::WlanLinuxFeatureProvider\(featureparams &params\)](#)  
documentation

Class [WlanNumPeersFeature](#) documentation

Class [WlanPeerInfo](#) documentation

Member [WlanPeerInfo::essid](#) documentation

Class [WlanPeersFeature](#) documentation

Member [\\_logfoo\(const char \\*,...\)](#) gcc supports 'args...' macros, afaik ms doesn't - check, would be a nicer hack

Member [featureparams](#) documentation

Member [unserializeFeatureVector\(const featurevector &features, featurevector &samples, char \\*data\)](#)  
catch "[invalid]" entries in the file and de-escape ";", "\", "[" and "]"

Member [meta\\_cluster\\_list](#) documentation

Member [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeGetAddressesInRange\(JNIEnv \\*jenv, jobject\)](#)  
documentation

Member [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeGetHardwareAddress\(JNIEnv \\*jenv, jobject\)](#)  
documentation

Member [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeGetLastMessage\(JNIEnv \\*, jobject\)](#)  
documentation

Member [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeInit\(JNIEnv \\*, jobject\)](#)  
documentation

Member [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeIsAddressValid\(JNIEnv \\*jenv, jobject, jstring pAdr\)](#)  
documentation



Member [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeStart](#)(JNIEnv \*, jobject)  
documentation

Member [Java\\_at\\_jku\\_bluetooth\\_LinuxBluetoothAdapter\\_nativeStop](#)(JNIEnv \*, jobject)  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_context\\_Context\\_nativeGetClusterMembership](#)(JNIEnv \*jenv, jclass, jlongArray cla  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_context\\_Context\\_nativeGetNumClusters](#)(JNIEnv \*, jclass)  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_context\\_Context\\_nativeInit](#)(JNIEnv \*jenv, jclass, jstring configFilename, jstring per  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_context\\_Context\\_nativeNextSample](#)(JNIEnv \*, jclass)  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_samples\\_NumericalContinuousSample\\_nativeGetVal](#)(JNIEnv \*, jobject, jint objRef)  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_samples\\_NumericalDiscreteSample\\_nativeGetVal](#)(JNIEnv \*, jobject, jint objRef)  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_samples\\_Sample\\_nativeFreeObject](#)(JNIEnv \*, jobject, jint objRef)  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_samples\\_Sample\\_nativeGetName](#)(JNIEnv \*jenv, jobject, jint objRef)  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_samples\\_Sample\\_nativeGetPosition](#)(JNIEnv \*, jobject, jint objRef)  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_samples\\_Sample\\_nativeGetType](#)(JNIEnv \*, jobject, jint objRef)  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_samples\\_SampleContainer\\_nativeGetSampleVector](#)(JNIEnv \*jenv, jclass)  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_samples\\_SampleContainer\\_nativeInit](#)(JNIEnv \*jenv, jclass, jstring configFilename, j  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_samples\\_SampleContainer\\_nativeNextSamples](#)(JNIEnv \*, jclass)  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_samples\\_StringListSample\\_nativeGetValues](#)(JNIEnv \*jenv, jobject, jint objRef)  
documentation

Member [Java\\_at\\_jku\\_intelligence\\_samples\\_StringSample\\_nativeGetVal](#)(JNIEnv \*jenv, jobject, jint objRef)  
documentation

Member [cluster\\_list](#) documentation

Member [keepRunning](#) documentation

Member [call\\_function](#)(const char \*file, const char \*func) strlen , Z\_STRLEN(retval));

Member [init\\_php](#)() fix that;

Member [php\\_move\\_towards](#)(const char \*file, PHPFeatureType type, void \*a, void \*b) foo  
COPY CTOR

Member [php\\_next\\_sample](#)(const char \*file, PHPFeatureType type) foo

Member [contexttrajectory](#) documentation

Member [DEFAULT\\_PEAK\\_THRESHOLD](#) documentation

Member [DEFAULT\\_SMOOTHING\\_FACTOR](#) documentation

Member [DEFAULT\\_SMOOTHING\\_WINDOW\\_WIDTH](#) documentation

Member [membershiplist\\_duration](#) documentation

Member [new](#) documentation

Member [edge\\_map](#) documentation

# Index

- ~Feature
  - Feature, [10](#)
- ~FeatureProvider
  - FeatureProvider, [15](#)
- ~PersistantFeature
  - PersistantFeature, [29](#)
- acfLagSums
  - Statistics, [34](#)
- acfSums
  - Statistics, [34](#)
- acfWindow
  - Statistics, [34](#)
- at::jku::intelligence::context::Context, [9](#)
- at::jku::intelligence::samples::Numerical-ContinuousSample, [23](#)
- at::jku::intelligence::samples::Numerical-DiscreteSample, [28](#)
- at::jku::intelligence::samples::Sample, [31](#)
- at::jku::intelligence::samples::SampleContainer, [32](#)
- at::jku::intelligence::samples::StringListSample, [35](#)
- at::jku::intelligence::samples::StringSample, [36](#)
- ClassifierAlgorithm
  - nextSample, [7](#)
- ClassifierAlgorithm, [7](#)
  - getClusterMembership, [8](#)
  - init, [8](#)
  - nextSample, [8](#)
- clone
  - Feature, [12](#)
  - NumericalContinuousFeature, [20](#)
  - NumericalDiscreteFeature, [25](#)
- externalize
  - Feature, [12](#)
- extnlze
  - Feature, [14](#)
- Feature, [10](#)
  - ~Feature, [10](#)
  - clone, [12](#)
  - externalize, [12](#)
  - extnlze, [14](#)
  - Feature, [11](#)
  - getDistance, [12](#)
  - getName, [12](#)
  - getPosition, [12](#)
  - getType, [13](#)
  - moveTowards, [13](#)
  - serialize, [13](#)
  - unserialize, [11](#)
- FeatureProvider
  - ~FeatureProvider, [15](#)
  - FeatureProvider, [16](#)
  - features, [16](#)
  - name, [16](#)
- FeatureProvider, [15](#)
  - FeatureProvider, [16](#)
  - getFeature, [16](#)
  - getFeatureList, [17](#)
  - getName, [17](#)
  - getSample, [17](#)
  - nextSample, [17](#)
- features
  - FeatureProvider, [16](#)
- getAcf
  - Statistics, [33](#)
- getClusterMembership
  - ClassifierAlgorithm, [8](#)
- getDistance
  - Feature, [12](#)
  - NumericalContinuousFeature, [20](#)
  - NumericalDiscreteFeature, [24](#)
- getFeature
  - FeatureProvider, [16](#)
  - SystemCommandStringListFeature-Provider, [38](#)
- getFeatureList
  - FeatureProvider, [17](#)
- getMean
  - Statistics, [33](#)
- getName
  - Feature, [12](#)
  - FeatureProvider, [17](#)
  - NumericalContinuousFeature, [20](#)
  - NumericalDiscreteFeature, [25](#)
- getNum

- Statistics, 33
- getPosition
  - Feature, 12
  - NumericalContinuousFeature, 21
  - NumericalDiscreteFeature, 24
- getSample
  - FeatureProvider, 17
  - SystemCommandStringListFeature-Provider, 38
- getType
  - Feature, 13
  - NumericalContinuousFeature, 21
  - NumericalDiscreteFeature, 25
- getVal
  - NumericalDiscreteFeature, 27
- getVariance
  - Statistics, 33
- init
  - ClassifierAlgorithm, 8
- maxAcfOrder
  - Statistics, 34
- maxval
  - NumericalDiscreteFeature, 25
- minval
  - NumericalDiscreteFeature, 25
- moveTowards
  - Feature, 13
  - NumericalContinuousFeature, 21
  - NumericalDiscreteFeature, 25
- name
  - FeatureProvider, 16
- nextSample
  - ClassifierAlgorithm, 7, 8
  - FeatureProvider, 17
  - Statistics, 33
  - SystemCommandStringListFeature-Provider, 39
- num
  - Statistics, 33
- NumericalContinuousFeature
  - read, 20
  - unserialize, 19
- NumericalContinuousFeature, 19
  - clone, 20
  - getDistance, 20
  - getName, 20
  - getPosition, 21
  - getType, 21
  - moveTowards, 21
  - serialize, 22
  - write, 22

- NumericalDiscreteFeature
  - clone, 25
  - getDistance, 24
  - getName, 25
  - getPosition, 24
  - getType, 25
  - maxval, 25
  - minval, 25
  - moveTowards, 25
  - NumericalDiscreteFeature, 26
  - read, 25
  - serialize, 25
  - unserialize, 25
  - val, 25
  - write, 25
- NumericalDiscreteFeature, 24
  - getVal, 27
  - NumericalDiscreteFeature, 26
- PersistentFeature
  - ~PersistentFeature, 29
  - PersistentFeature, 30
  - read, 29
- PersistentFeature, 29
  - PersistentFeature, 30
  - v, 30
  - write, 30
- read
  - NumericalContinuousFeature, 20
  - NumericalDiscreteFeature, 25
  - PersistentFeature, 29
- serialize
  - Feature, 13
  - NumericalContinuousFeature, 22
  - NumericalDiscreteFeature, 25
- squaresum
  - Statistics, 33
- Statistics, 33
  - acfLagSums, 34
  - acfSums, 34
  - acfWindow, 34
  - getAcf, 33
  - getMean, 33
  - getNum, 33
  - getVariance, 33
  - maxAcfOrder, 34
  - nextSample, 33
  - num, 33
  - squaresum, 33
  - Statistics, 34
  - sum, 33
- sum

- Statistics, [33](#)
- systemCommand
  - SystemCommandStringListFeature-  
Provider, [38](#)
- SystemCommandStringListFeatureProvider
  - systemCommand, [38](#)
  - SystemCommandStringListFeature-  
Provider, [37](#)
- SystemCommandStringListFeatureProvider, [37](#)
  - getFeature, [38](#)
  - getSample, [38](#)
  - nextSample, [39](#)
- unserialize
  - Feature, [11](#)
  - NumericalContinuousFeature, [19](#)
  - NumericalDiscreteFeature, [25](#)
- v
  - PersistentFeature, [30](#)
- val
  - NumericalDiscreteFeature, [25](#)
- write
  - NumericalContinuousFeature, [22](#)
  - NumericalDiscreteFeature, [25](#)
  - PersistentFeature, [30](#)