

News Feeder Refresh Daemon (nfrd)

Generated by Doxygen 1.8.1.1

Tue Oct 23 2012 16:48:36

Contents

1	Details in configuration file	1
1.1	Introduction	1
1.2	Format	1
1.3	master - General configuration	2
1.3.1	log - Log file	2
1.4	module - Module loading list	2
1.5	AdminService - Module configuration	2
1.5.1	port - TCP Listening port number	2
1.5.2	username - Username for authentication	2
1.5.3	password - Password for authentication	2
1.5.4	timeout - Timeout on socket communicating	3
1.6	Crawler - Module configuration	3
1.6.1	thread - Number of crawling thread	3
1.6.2	sleep - Time to sleep for each round	3
1.7	Statistics - Module configuration	3
1.7.1	period - Time to update or record	3
1.8	mysql - Database connector configuration	3
1.8.1	host - Address of the host	3
1.8.2	port - Mysql port of the host	3
1.8.3	database database - Database schema	3
1.8.4	username - Username for authentication	4
1.8.5	password - Password for authentication	4
2	Protocol talks to Front-end	5
2.1	Introduction	5
2.1.1	Sockets	5
2.2	Login	5
2.2.1	Success case	6
2.2.2	Fail case	6
2.3	status - Check the status	6
2.3.1	crawler - crawler status	6

2.3.2	uptime - Back-end uptime	7
2.4	crawler - Control the crawler	7
2.4.1	start - Start the crawler	7
2.4.2	status - Check the status of the crawler	8
2.4.3	stop - Stop the crawler	8
2.5	config - Configure back-end	8
2.5.1	add - Add/Change an attribute	9
2.5.2	get - Return an attribute	9
2.5.3	load - Load configuration file	10
2.5.4	save - Save configuration file	10
2.6	shutdown - Shutdown the Back-End	11
2.7	quit - Close connection	11
2.8	Unknown command handling	12
2.9	Examples	12
2.9.1	Restart the crawler	12
2.9.2	Change the number of thread in crawler	12
3	Command line usage	13
3.1	Introduction	13
3.2	Manual	13
4	Namespace Documentation	15
4.1	nfrd Namespace Reference	15
4.1.1	Detailed Description	15
4.2	nfrd::config Namespace Reference	15
4.2.1	Detailed Description	16
4.3	nfrd::log Namespace Reference	16
4.3.1	Detailed Description	16
4.4	nfrd::misc Namespace Reference	16
4.4.1	Detailed Description	17
4.5	nfrd::misc::Utility Namespace Reference	17
4.5.1	Detailed Description	17
4.5.2	Function Documentation	18
4.5.2.1	GetArguments	18
4.5.2.2	Read	18
4.5.2.3	Read	18
4.5.2.4	Read	18
4.5.2.5	ToBool	19
4.5.2.6	ToInt	19
4.5.2.7	Trim	19
4.6	nfrd::module Namespace Reference	19

4.6.1	Detailed Description	20
4.7	nfrd::parser Namespace Reference	20
4.7.1	Detailed Description	21
5	Class Documentation	23
5.1	nfrd::module::AdminService Class Reference	23
5.1.1	Detailed Description	24
5.1.2	Constructor & Destructor Documentation	24
5.1.2.1	AdminService	24
5.1.2.2	~AdminService	24
5.1.3	Member Function Documentation	24
5.1.3.1	Authenticate	24
5.1.3.2	Exit	24
5.1.3.3	operator()	25
5.1.3.4	Register	25
5.1.3.5	Stop	25
5.1.4	Member Data Documentation	25
5.1.4.1	acceptor	25
5.1.4.2	condition	25
5.1.4.3	master	25
5.1.4.4	password	25
5.1.4.5	thread_mutex	25
5.1.4.6	threads	25
5.1.4.7	username	26
5.2	nfrd::module::AdminServiceThread Class Reference	26
5.2.1	Detailed Description	27
5.2.2	Constructor & Destructor Documentation	27
5.2.2.1	AdminServiceThread	27
5.2.2.2	~AdminServiceThread	27
5.2.3	Member Function Documentation	27
5.2.3.1	config_main	27
5.2.3.2	crawler_main	28
5.2.3.3	GetStream	28
5.2.3.4	Handle	28
5.2.3.5	operator()	28
5.2.3.6	shutdown_main	28
5.2.3.7	sin	28
5.2.3.8	sout	29
5.2.3.9	status_main	29
5.2.3.10	status_send_module_status	29

5.2.4	Member Data Documentation	29
5.2.4.1	dynamic	29
5.2.4.2	master	29
5.2.4.3	parent	29
5.2.4.4	remote_address	29
5.2.4.5	stream	29
5.2.4.6	timeout	30
5.3	nfrd::parser::AtomItem Class Reference	30
5.3.1	Detailed Description	31
5.3.2	Constructor & Destructor Documentation	31
5.3.2.1	AtomItem	31
5.3.2.2	~AtomItem	31
5.3.3	Member Function Documentation	31
5.3.3.1	GetAuthor	31
5.3.3.2	GetContent	32
5.3.3.3	GetPostDate	32
5.3.3.4	GetTitle	32
5.3.3.5	GetURL	32
5.3.3.6	HasAuthor	33
5.3.3.7	HasContent	33
5.3.3.8	HasPostDate	33
5.3.3.9	HasTitle	33
5.3.3.10	HasURL	34
5.3.3.11	SetAuthor	34
5.3.3.12	SetContent	34
5.3.3.13	SetContent	34
5.3.3.14	SetPostDate	34
5.3.3.15	SetTitle	34
5.3.3.16	SetURL	35
5.3.4	Member Data Documentation	35
5.3.4.1	author	35
5.3.4.2	content	35
5.3.4.3	postDate	35
5.3.4.4	title	35
5.3.4.5	url	35
5.4	nfrd::parser::AtomParser Class Reference	35
5.4.1	Detailed Description	36
5.4.2	Constructor & Destructor Documentation	36
5.4.2.1	AtomParser	36
5.4.2.2	~AtomParser	36

5.4.3	Member Function Documentation	36
5.4.3.1	GetLastBuildDate	36
5.4.3.2	ReadDom	37
5.4.4	Member Data Documentation	37
5.4.4.1	lastBuildDate	37
5.5	nfrd::misc::AutoDB<_Tp> Class Template Reference	37
5.5.1	Detailed Description	38
5.5.2	Member Typedef Documentation	38
5.5.2.1	element_type	38
5.5.3	Constructor & Destructor Documentation	39
5.5.3.1	AutoDB	39
5.5.3.2	AutoDB	39
5.5.3.3	AutoDB	39
5.5.3.4	~AutoDB	39
5.5.3.5	AutoDB	39
5.5.4	Member Function Documentation	39
5.5.4.1	get	40
5.5.4.2	operator AutoDB<_Tp1>	40
5.5.4.3	operator AutoDBRef<_Tp1>	40
5.5.4.4	operator*	40
5.5.4.5	operator->	40
5.5.4.6	operator=	40
5.5.4.7	operator=	40
5.5.4.8	operator=	41
5.5.4.9	release	41
5.5.4.10	reset	41
5.5.5	Member Data Documentation	41
5.5.5.1	_M_ptr	41
5.6	nfrd::misc::AutoDBRef<_Tp1> Struct Template Reference	41
5.6.1	Detailed Description	42
5.6.2	Constructor & Destructor Documentation	42
5.6.2.1	AutoDBRef	42
5.6.3	Member Data Documentation	42
5.6.3.1	_M_ptr	42
5.7	nfrd::config::ConfigException Class Reference	42
5.7.1	Detailed Description	43
5.7.2	Constructor & Destructor Documentation	43
5.7.2.1	ConfigException	43
5.7.2.2	~ConfigException	43
5.7.3	Member Function Documentation	43

5.7.3.1	what	43
5.7.4	Member Data Documentation	43
5.7.4.1	msg	43
5.8	nfrd::config::ConfigManager Class Reference	43
5.8.1	Detailed Description	44
5.8.2	Constructor & Destructor Documentation	44
5.8.2.1	ConfigManager	44
5.8.2.2	ConfigManager	45
5.8.2.3	~ConfigManager	45
5.8.3	Member Function Documentation	45
5.8.3.1	operator[]	45
5.8.3.2	operator[]	45
5.8.3.3	Read	45
5.8.3.4	Read	46
5.8.3.5	SetFileName	46
5.8.3.6	Write	46
5.8.3.7	Write	46
5.8.4	Member Data Documentation	46
5.8.4.1	configFile	46
5.8.4.2	value	46
5.9	nfrd::config::ConfigSector Class Reference	47
5.9.1	Detailed Description	47
5.9.2	Member Typedef Documentation	48
5.9.2.1	const_iterator	48
5.9.2.2	iterator	48
5.9.3	Constructor & Destructor Documentation	48
5.9.3.1	ConfigSector	48
5.9.3.2	~ConfigSector	48
5.9.4	Member Function Documentation	48
5.9.4.1	begin	48
5.9.4.2	begin	48
5.9.4.3	end	48
5.9.4.4	end	48
5.9.4.5	operator[]	49
5.9.4.6	operator[]	49
5.9.4.7	Read	49
5.9.4.8	Write	49
5.9.5	Member Data Documentation	49
5.9.5.1	name	50
5.9.5.2	value	50

5.10 nfrd::module::Crawler Class Reference	50
5.10.1 Detailed Description	51
5.10.2 Constructor & Destructor Documentation	51
5.10.2.1 Crawler	51
5.10.2.2 ~Crawler	51
5.10.3 Member Function Documentation	51
5.10.3.1 InitialiseQueue	51
5.10.3.2 MainLoop	52
5.10.3.3 operator()	52
5.10.3.4 PersistQueue	52
5.10.3.5 RunMaintenanceTasks	52
5.10.3.6 StartThreads	52
5.10.3.7 Stop	52
5.10.4 Member Data Documentation	52
5.10.4.1 crawlers	52
5.10.4.2 healthMutex	52
5.10.4.3 healthThreadCondition	52
5.10.4.4 isAlive	52
5.10.4.5 priorityQueue	52
5.10.4.6 threads	53
5.10.4.7 WAITTIME	53
5.11 nfrd::module::CrawlerThread Class Reference	53
5.11.1 Detailed Description	54
5.11.2 Constructor & Destructor Documentation	54
5.11.2.1 CrawlerThread	54
5.11.2.2 ~CrawlerThread	54
5.11.3 Member Function Documentation	54
5.11.3.1 Crawl	54
5.11.3.2 GetId	54
5.11.3.3 Initialise	54
5.11.3.4 Request	54
5.11.3.5 Stop	54
5.11.3.6 UpdateItem	55
5.11.4 Member Data Documentation	55
5.11.4.1 config	55
5.11.4.2 id	55
5.11.4.3 isAlive	55
5.11.4.4 log	55
5.11.4.5 priorityQueue	55
5.12 nfrd::misc::DateTime Class Reference	55

5.12.1 Detailed Description	57
5.12.2 Constructor & Destructor Documentation	57
5.12.2.1 DateTime	57
5.12.2.2 DateTime	57
5.12.2.3 ~DateTime	57
5.12.3 Member Function Documentation	57
5.12.3.1 ExportToMySQL	57
5.12.3.2 GetDay	58
5.12.3.3 GetHour	58
5.12.3.4 GetMinute	58
5.12.3.5 GetMonth	58
5.12.3.6 GetSecond	58
5.12.3.7 GetYear	58
5.12.3.8 ImportFromMySQL	59
5.12.3.9 JustifyDate	59
5.12.3.10 JustifyTime	59
5.12.3.11 operator!=	59
5.12.3.12 operator<	59
5.12.3.13 operator==	60
5.12.3.14 operator>	60
5.12.3.15 ParseFromRFC3339	60
5.12.3.16 ParseFromRFC822	60
5.12.3.17 ParseFromString	61
5.12.3.18 Set	61
5.12.3.19 SetDateOffset	61
5.12.3.20 SetDay	62
5.12.3.21 SetHour	62
5.12.3.22 SetMinute	62
5.12.3.23 SetMonth	63
5.12.3.24 SetSecond	63
5.12.3.25 SetTimeOffset	63
5.12.3.26 SetYear	63
5.12.4 Member Data Documentation	64
5.12.4.1 day	64
5.12.4.2 hour	64
5.12.4.3 minute	64
5.12.4.4 month	64
5.12.4.5 second	64
5.12.4.6 year	64
5.13 nfrd::parser::FeedItem Class Reference	64

5.13.1 Detailed Description	65
5.13.2 Constructor & Destructor Documentation	66
5.13.2.1 FeedItem	66
5.13.2.2 ~FeedItem	66
5.13.3 Member Function Documentation	66
5.13.3.1 AddImage	66
5.13.3.2 ClearImage	66
5.13.3.3 GetGeoLocation	66
5.13.3.4 GetImageList	66
5.13.3.5 HasFullContent	67
5.13.3.6 HasGeoLocation	67
5.13.3.7 HasImageList	67
5.13.3.8 RemoveImage	67
5.13.3.9 SetContent	67
5.13.3.10 SetFullContent	67
5.13.3.11 SetGeoLocation	68
5.13.4 Member Data Documentation	68
5.13.4.1 full_content	68
5.13.4.2 geo_location	68
5.13.4.3 image_list	68
5.14 nfrd::parser::FeedParser Class Reference	68
5.14.1 Detailed Description	69
5.14.2 Constructor & Destructor Documentation	69
5.14.2.1 FeedParser	69
5.14.2.2 ~FeedParser	69
5.14.3 Member Function Documentation	69
5.14.3.1 GetItemList	69
5.14.3.2 ReadDom	70
5.14.3.3 ReadURL	70
5.14.4 Member Data Documentation	70
5.14.4.1 item	70
5.14.4.2 url	70
5.15 nfrd::module::FeedPriorityQueue Class Reference	70
5.15.1 Detailed Description	72
5.15.2 Constructor & Destructor Documentation	72
5.15.2.1 FeedPriorityQueue	72
5.15.2.2 ~FeedPriorityQueue	73
5.15.3 Member Function Documentation	73
5.15.3.1 CleanupQueue	73
5.15.3.2 GetAllItems	73

5.15.3.3	GetThread	73
5.15.3.4	IterateOnce	73
5.15.3.5	MainLoop	73
5.15.3.6	PopFeed	74
5.15.3.7	PushFeed	74
5.15.3.8	SetNumberOfUsersInSystem	74
5.15.3.9	Start	74
5.15.3.10	StartThreads	74
5.15.3.11	Stop	74
5.15.4	Member Data Documentation	74
5.15.4.1	config	74
5.15.4.2	heapMutex	74
5.15.4.3	incomingHasChanged	74
5.15.4.4	incomingMutex	75
5.15.4.5	incomingQueue	75
5.15.4.6	isAlive	75
5.15.4.7	itemQueue	75
5.15.4.8	itemsPopped	75
5.15.4.9	log	75
5.15.4.10	MAXPOP	75
5.15.4.11	numberOfUsersInSystem	75
5.15.4.12	outgoingHasChanged	75
5.15.4.13	outgoingMutex	75
5.15.4.14	outgoingQueue	76
5.15.4.15	outgoingQueueCondition	76
5.15.4.16	priorityQueueThread	76
5.15.4.17	queueUpdateCondition	76
5.15.4.18	tempOutgoingQueue	76
5.15.4.19	updatedMutex	76
5.16	nfrd::parser::FeedXParser Class Reference	76
5.16.1	Detailed Description	78
5.16.2	Constructor & Destructor Documentation	78
5.16.2.1	FeedXParser	78
5.16.2.2	~FeedXParser	78
5.16.3	Member Function Documentation	78
5.16.3.1	AdvancedMatchDivide	78
5.16.3.2	AdvancedMatchLine	78
5.16.3.3	Construct	79
5.16.3.4	ExtractGeoLocation	79
5.16.3.5	ExtractImages	79

5.16.3.6	GetItemList	79
5.16.3.7	GetLastBuildDate	79
5.16.3.8	GetLongestText	80
5.16.3.9	PatchAndMatch	80
5.16.3.10	ReadURL	80
5.16.3.11	RefineFeed	80
5.16.3.12	TextSize	81
5.16.3.13	TrimEmptyTag	81
5.16.4	Member Data Documentation	81
5.16.4.1	allowed_tags	81
5.16.4.2	parser	81
5.16.4.3	trimmed_tags	81
5.17	nfrd::misc::Image::File Class Reference	81
5.17.1	Detailed Description	82
5.17.2	Constructor & Destructor Documentation	82
5.17.2.1	File	82
5.17.2.2	~File	82
5.17.2.3	File	82
5.17.3	Member Function Documentation	83
5.17.3.1	GetData	83
5.17.3.2	GetSize	83
5.17.3.3	GetType	83
5.17.3.4	operator=	83
5.17.4	Member Data Documentation	84
5.17.4.1	data	84
5.17.4.2	size	84
5.17.4.3	type	84
5.18	nfrd::parser::HasNoValue Class Reference	84
5.18.1	Detailed Description	84
5.18.2	Constructor & Destructor Documentation	85
5.18.2.1	HasNoValue	85
5.18.2.2	~HasNoValue	85
5.19	nfrd::misc::Image Class Reference	85
5.19.1	Detailed Description	86
5.19.2	Member Enumeration Documentation	86
5.19.2.1	Type	86
5.19.3	Constructor & Destructor Documentation	86
5.19.3.1	Image	86
5.19.3.2	Image	87
5.19.3.3	~Image	87

5.19.4	Member Function Documentation	87
5.19.4.1	ExportJpeg	87
5.19.4.2	ExportPng	87
5.19.4.3	FitSize	88
5.19.4.4	GetHeight	88
5.19.4.5	GetWidth	88
5.19.4.6	Load	88
5.19.4.7	Load	88
5.19.4.8	Load	89
5.19.4.9	Load	89
5.19.4.10	operator=	89
5.19.5	Member Data Documentation	90
5.19.5.1	im	90
5.20	nfrd::misc::ImageException Class Reference	90
5.20.1	Detailed Description	90
5.20.2	Constructor & Destructor Documentation	90
5.20.2.1	ImageException	90
5.20.2.2	~ImageException	90
5.20.3	Member Function Documentation	90
5.20.3.1	what	91
5.20.4	Member Data Documentation	91
5.20.4.1	msg	91
5.21	nfrd::parser::InvalidSource Class Reference	91
5.21.1	Detailed Description	91
5.21.2	Constructor & Destructor Documentation	91
5.21.2.1	InvalidSource	91
5.21.2.2	~InvalidSource	92
5.22	nfrd::config::IOException Class Reference	92
5.22.1	Detailed Description	92
5.22.2	Constructor & Destructor Documentation	92
5.22.2.1	IOException	92
5.22.2.2	~IOException	92
5.23	nfrd::log::IOException Class Reference	93
5.23.1	Detailed Description	93
5.23.2	Constructor & Destructor Documentation	93
5.23.2.1	IOException	93
5.23.2.2	~IOException	93
5.24	nfrd::parser::Item Class Reference	93
5.24.1	Detailed Description	95
5.24.2	Member Typedef Documentation	95

5.24.2.1	Image	95
5.24.3	Constructor & Destructor Documentation	95
5.24.3.1	~Item	95
5.24.4	Member Function Documentation	95
5.24.4.1	GetAuthor	95
5.24.4.2	GetContent	95
5.24.4.3	GetGeoLocation	96
5.24.4.4	GetImageList	96
5.24.4.5	GetPostDate	96
5.24.4.6	GetTitle	96
5.24.4.7	GetURL	97
5.24.4.8	HasAuthor	97
5.24.4.9	HasContent	97
5.24.4.10	HasGeoLocation	97
5.24.4.11	HasImageList	97
5.24.4.12	HasPostDate	98
5.24.4.13	HasTitle	98
5.24.4.14	HasURL	98
5.25	nfrd::config::ItemNotFound Class Reference	98
5.25.1	Detailed Description	99
5.25.2	Constructor & Destructor Documentation	99
5.25.2.1	ItemNotFound	99
5.25.2.2	~ItemNotFound	99
5.26	nfrd::parser::FeedXParser::IteratorPair Struct Reference	99
5.26.1	Detailed Description	99
5.26.2	Constructor & Destructor Documentation	100
5.26.2.1	IteratorPair	100
5.26.2.2	IteratorPair	100
5.26.3	Member Data Documentation	100
5.26.3.1	end	100
5.26.3.2	is_allowed_tag	100
5.26.3.3	it	100
5.27	nfrd::log::LogException Class Reference	100
5.27.1	Detailed Description	101
5.27.2	Constructor & Destructor Documentation	101
5.27.2.1	LogException	101
5.27.2.2	~LogException	101
5.27.3	Member Function Documentation	101
5.27.3.1	what	101
5.27.4	Member Data Documentation	101

5.27.4.1	msg	101
5.28	nfrd::log::LogManager Class Reference	102
5.28.1	Detailed Description	102
5.28.2	Member Enumeration Documentation	102
5.28.2.1	Type	102
5.28.3	Constructor & Destructor Documentation	103
5.28.3.1	LogManager	103
5.28.3.2	~LogManager	103
5.28.4	Member Function Documentation	103
5.28.4.1	Disable	103
5.28.4.2	Enable	103
5.28.4.3	isEnabled	103
5.28.4.4	operator()	103
5.28.4.5	operator()	104
5.28.5	Member Data Documentation	104
5.28.5.1	enabled	104
5.28.5.2	fout	104
5.28.5.3	io_mutex	104
5.29	nfrd::Master Class Reference	104
5.29.1	Detailed Description	106
5.29.2	Member Typedef Documentation	106
5.29.2.1	Task	106
5.29.3	Constructor & Destructor Documentation	106
5.29.3.1	Master	106
5.29.3.2	~Master	106
5.29.4	Member Function Documentation	106
5.29.4.1	GetConfig	106
5.29.4.2	GetModule	106
5.29.4.3	GetStartTime	107
5.29.4.4	GetVersion	107
5.29.4.5	IsOnline	107
5.29.4.6	LoadModule	107
5.29.4.7	LoadModules	107
5.29.4.8	Main	107
5.29.4.9	SetModule	108
5.29.4.10	SetModule	108
5.29.4.11	Terminate	108
5.29.4.12	UnloadModule	108
5.29.4.13	UnloadModules	108
5.29.5	Member Data Documentation	108

5.29.5.1	condition	108
5.29.5.2	config	109
5.29.5.3	log	109
5.29.5.4	module	109
5.29.5.5	mutex	109
5.29.5.6	online	109
5.29.5.7	start_time	109
5.29.5.8	task_queue	109
5.30	nfrd::module::Module Class Reference	109
5.30.1	Detailed Description	110
5.30.2	Member Enumeration Documentation	110
5.30.2.1	Status	110
5.30.3	Constructor & Destructor Documentation	111
5.30.3.1	Module	111
5.30.3.2	Module	111
5.30.3.3	~Module	111
5.30.4	Member Function Documentation	111
5.30.4.1	GetName	111
5.30.4.2	GetStatus	111
5.30.4.3	GetStatusString	111
5.30.4.4	GetThread	112
5.30.4.5	operator()	112
5.30.4.6	Start	112
5.30.4.7	Stop	112
5.30.5	Member Data Documentation	112
5.30.5.1	config	112
5.30.5.2	log	112
5.30.5.3	name	112
5.30.5.4	status	112
5.30.5.5	thread	112
5.31	nfrd::module::ModuleException Class Reference	113
5.31.1	Detailed Description	113
5.31.2	Constructor & Destructor Documentation	113
5.31.2.1	ModuleException	113
5.31.2.2	~ModuleException	113
5.31.3	Member Function Documentation	113
5.31.3.1	what	113
5.31.4	Member Data Documentation	113
5.31.4.1	msg	114
5.32	nfrd::parser::Parser Class Reference	114

5.32.1	Detailed Description	114
5.32.2	Constructor & Destructor Documentation	114
5.32.2.1	~Parser	114
5.32.3	Member Function Documentation	115
5.32.3.1	GetItemList	115
5.32.3.2	GetLastBuildDate	115
5.32.3.3	ReadURL	115
5.33	nfrd::parser::ParserException Class Reference	115
5.33.1	Detailed Description	116
5.33.2	Constructor & Destructor Documentation	116
5.33.2.1	ParserException	116
5.33.2.2	~ParserException	116
5.33.3	Member Function Documentation	116
5.33.3.1	what	116
5.33.4	Member Data Documentation	117
5.33.4.1	msg	117
5.34	nfrd::module::QueueItem Class Reference	117
5.34.1	Constructor & Destructor Documentation	118
5.34.1.1	QueueItem	118
5.34.1.2	QueueItem	118
5.34.2	Member Function Documentation	118
5.34.2.1	CalculateConstant	118
5.34.2.2	feed_comparer	118
5.34.2.3	GetFeedID	118
5.34.2.4	IncrementPriority	118
5.34.2.5	ResetPriority	118
5.34.2.6	SetContentUpdateAverage	119
5.34.2.7	SetFeedType	119
5.34.2.8	SetUsersAffected	119
5.34.3	Member Data Documentation	119
5.34.3.1	contentUpdateAverage	119
5.34.3.2	feedType	119
5.34.3.3	id	119
5.34.3.4	priority	119
5.34.3.5	priorityConstant	119
5.34.3.6	timeSpentInQueue	119
5.34.3.7	usersAffected	119
5.35	nfrd::parser::RSSItem Class Reference	120
5.35.1	Detailed Description	121
5.35.2	Constructor & Destructor Documentation	121

5.35.2.1	RSSItem	121
5.35.2.2	~RSSItem	121
5.35.3	Member Function Documentation	121
5.35.3.1	GetAuthor	121
5.35.3.2	GetContent	121
5.35.3.3	GetPostDate	122
5.35.3.4	GetTitle	122
5.35.3.5	GetURL	122
5.35.3.6	HasAuthor	122
5.35.3.7	HasContent	123
5.35.3.8	HasPostDate	123
5.35.3.9	HasTitle	123
5.35.3.10	HasURL	123
5.35.3.11	SetAuthor	123
5.35.3.12	SetContent	124
5.35.3.13	SetContent	124
5.35.3.14	SetPostDate	124
5.35.3.15	SetTitle	124
5.35.3.16	SetURL	124
5.35.4	Member Data Documentation	124
5.35.4.1	author	125
5.35.4.2	content	125
5.35.4.3	postDate	125
5.35.4.4	title	125
5.35.4.5	url	125
5.36	nfrd::parser::RSSParser Class Reference	125
5.36.1	Detailed Description	126
5.36.2	Constructor & Destructor Documentation	126
5.36.2.1	RSSParser	126
5.36.2.2	~RSSParser	126
5.36.3	Member Function Documentation	126
5.36.3.1	GetLastBuildDate	126
5.36.3.2	GetValue	127
5.36.3.3	ReadDom	127
5.36.4	Member Data Documentation	127
5.36.4.1	buffer	127
5.36.4.2	lastBuildDate	127
5.37	nfrd::module::Statistics Class Reference	128
5.37.1	Detailed Description	128
5.37.2	Constructor & Destructor Documentation	128

5.37.2.1	Statistics	128
5.37.2.2	~Statistics	128
5.37.3	Member Function Documentation	129
5.37.3.1	operator()	129
5.37.3.2	Stop	129
5.37.4	Member Data Documentation	129
5.37.4.1	period	129
6	File Documentation	131
6.1	docs/config.dox File Reference	131
6.2	docs/namespace.dox File Reference	131
6.3	docs/protocol.dox File Reference	131
6.4	docs/usage.dox File Reference	131
6.5	include/nfrd/AdminService.h File Reference	131
6.5.1	Detailed Description	132
6.5.2	DESCRIPTION	132
6.6	include/nfrd/AdminServiceThread.h File Reference	132
6.6.1	Detailed Description	133
6.6.2	DESCRIPTION	133
6.7	include/nfrd/AtomParser.h File Reference	133
6.7.1	Detailed Description	133
6.7.2	DESCRIPTION	133
6.8	include/nfrd/AutoDB.h File Reference	134
6.8.1	Detailed Description	134
6.8.2	DESCRIPTION	134
6.9	include/nfrd/ConfigManager.h File Reference	134
6.9.1	Detailed Description	135
6.9.2	DESCRIPTION	135
6.10	include/nfrd/Crawler.h File Reference	135
6.10.1	Detailed Description	136
6.10.2	DESCRIPTION	136
6.11	include/nfrd/CrawlerThread.h File Reference	136
6.11.1	Detailed Description	137
6.11.2	DESCRIPTION	137
6.12	include/nfrd/DateTime.h File Reference	137
6.12.1	Detailed Description	137
6.12.2	DESCRIPTION	137
6.13	include/nfrd/FeedParser.h File Reference	137
6.13.1	Detailed Description	138
6.13.2	DESCRIPTION	138

6.14	include/nfrd/FeedPriorityQueue.h File Reference	138
6.14.1	Detailed Description	139
6.14.2	DESCRIPTION	139
6.15	include/nfrd/FeedXParser.h File Reference	139
6.15.1	Detailed Description	140
6.15.2	DESCRIPTION	140
6.16	include/nfrd/Image.h File Reference	140
6.16.1	Detailed Description	140
6.16.2	DESCRIPTION	141
6.17	include/nfrd/LogManager.h File Reference	141
6.17.1	Detailed Description	141
6.17.2	DESCRIPTION	141
6.18	include/nfrd/Master.h File Reference	142
6.18.1	Detailed Description	142
6.18.2	DESCRIPTION	142
6.19	include/nfrd/Module.h File Reference	142
6.19.1	Detailed Description	143
6.19.2	DESCRIPTION	143
6.20	include/nfrd/Parser.h File Reference	143
6.20.1	Detailed Description	144
6.20.2	DESCRIPTION	144
6.21	include/nfrd/QueueItem.h File Reference	144
6.21.1	Detailed Description	144
6.21.2	DESCRIPTION	144
6.22	include/nfrd/RSSParser.h File Reference	145
6.22.1	Detailed Description	145
6.22.2	DESCRIPTION	145
6.23	include/nfrd/Statistics.h File Reference	145
6.23.1	Detailed Description	146
6.23.2	DESCRIPTION	146
6.24	include/nfrd/Utility.h File Reference	146
6.24.1	Detailed Description	147
6.24.2	DESCRIPTION	147
6.25	src/AdminService.cpp File Reference	147
6.25.1	Detailed Description	147
6.25.2	DESCRIPTION	147
6.26	src/AdminServiceThread.cpp File Reference	147
6.26.1	Detailed Description	148
6.26.2	DESCRIPTION	148
6.27	src/AtomParser.cpp File Reference	148

6.27.1 Detailed Description	148
6.27.2 DESCRIPTION	148
6.28 src/ConfigManager.cpp File Reference	148
6.28.1 Detailed Description	149
6.28.2 DESCRIPTION	149
6.28.3 Variable Documentation	149
6.28.3.1 WHITESPACES	149
6.29 src/Crawler.cpp File Reference	149
6.29.1 Detailed Description	149
6.29.2 DESCRIPTION	149
6.30 src/CrawlerThread.cpp File Reference	150
6.30.1 Detailed Description	150
6.30.2 DESCRIPTION	150
6.31 src/DateTime.cpp File Reference	150
6.31.1 Detailed Description	150
6.31.2 DESCRIPTION	151
6.32 src/FeedParser.cpp File Reference	151
6.32.1 Detailed Description	151
6.32.2 DESCRIPTION	151
6.33 src/FeedPriorityQueue.cpp File Reference	151
6.33.1 Detailed Description	151
6.33.2 DESCRIPTION	151
6.34 src/FeedXParser.cpp File Reference	152
6.34.1 Detailed Description	152
6.34.2 DESCRIPTION	152
6.34.3 Variable Documentation	152
6.34.3.1 ALLOWED_TAGS	152
6.34.3.2 SIZE_MODIFIER	153
6.34.3.3 TRIMMED_TAGS	153
6.35 src/Image.cpp File Reference	153
6.35.1 Detailed Description	153
6.35.2 DESCRIPTION	153
6.36 src/LogManager.cpp File Reference	153
6.36.1 Detailed Description	154
6.36.2 DESCRIPTION	154
6.37 src/Master.cpp File Reference	154
6.37.1 Detailed Description	154
6.37.2 DESCRIPTION	154
6.38 src/Module.cpp File Reference	154
6.38.1 Detailed Description	154

6.38.2 DESCRIPTION	155
6.39 src/nfrd.cpp File Reference	155
6.39.1 Detailed Description	155
6.39.2 DESCRIPTION	155
6.39.3 Function Documentation	155
6.39.3.1 main	155
6.39.3.2 PrintUsage	156
6.39.3.3 Terminate	156
6.39.4 Variable Documentation	156
6.39.4.1 tracker	156
6.40 src/Parser.cpp File Reference	156
6.40.1 Detailed Description	156
6.40.2 DESCRIPTION	156
6.41 src/QueueItem.cpp File Reference	156
6.41.1 Detailed Description	157
6.41.2 DESCRIPTION	157
6.42 src/RSSParser.cpp File Reference	157
6.42.1 Detailed Description	157
6.42.2 DESCRIPTION	157
6.43 src/Statistics.cpp File Reference	157
6.43.1 Detailed Description	157
6.43.2 DESCRIPTION	158
6.44 src/Utility.cpp File Reference	158
6.44.1 Detailed Description	158
6.44.2 DESCRIPTION	158
6.44.3 Function Documentation	158
6.44.3.1 _write_data	158
6.44.4 Variable Documentation	159
6.44.4.1 WHITESPACES	159

Chapter 1

Details in configuration file

Author

Shiwei Zhang sz653@uow.edu.au

Date

23/04/12

Version

0.1

1.1 Introduction

This page contains the format of the configuration file and explains every attributes in the config.

The nfrd program will use "nfrd.cfg" as configuration file by default.

To use the specified config file for nfrd, please refer to [Manual](#).

Note: Comments in the configuration file are not allowed.

See also

[nfrd::config::ConfigManager](#)

1.2 Format

The config file currently has two levels. One is called **Sector** and another is called **Attribute**.

A config file may have many sectors and a sector may have many attributes. For example:

```
[example]
name      = example
description = just a simple one
whitespace = " bla bla bla "
quote     = "what do you mean by "quote" "
```

As shown above, there is one sector called "example" and this sector has 4 attributes: **name**, **description**, **whitespace** and **quote**.

For parsing the sector name, every thing in `[]` is considered as the name of the sector.

For parsing the attribute, its name and its value are separated by a `=`. Although ConfigManager supports white-spaces in attribute, it is not recommended to do so. The leading and trailing white-spaces of the attribute name is trimmed. For the value of attribute, same rule is applied. However, it can handle white-spaces. Two quotes `""` are used to indicate the value of the attribute if there are leading or trailing white-spaces. With quotes, the value of **whitespace** is (*bla bla bla*). Without quotes, it will be parsed as (*bla bla bla*). The quotes in the quotes is considered as a part of the value. For example, the value of **quote** is *what do you mean by "quote"*

Note: All values are considered as strings.

1.3 master - General configuration

```
[master]
log      = nfrd.log
```

1.3.1 log - Log file

This attribute specifies where nfrd writes logs to.

The path can be an abstract path or a relative path. For relative path, it is not related to where the nfrd program is, but the path the nfrd program is working. To disable logging, just delete this attribute or leave it blank.

1.4 module - Module loading list

```
[module]
AdminService    = 1
Crawler         = 0
```

To enable a module, set the module name with a positive number (normally "1").

To disable a module, delete the attribute for the module or set a zero to it.

1.5 AdminService - Module configuration

```
[AdminService]
port          = 6373
username      = username
password      = password
timeout       = 60
```

1.5.1 port - TCP Listening port number

This attribute specifies which port the AdminService Module of nfrd listens.

1.5.2 username - Username for authentication

This attribute provides the username for authentication

1.5.3 password - Password for authentication

This attribute provides the password for authentication

1.5.4 timeout - Timeout on socket communicating

This optional attribute specifies the time limit that AdminService Module of nfrd on a single operation, especially in socket reading and writing. By default, it is specified as 60 seconds.

1.6 Crawler - Module configuration

```
[Crawler]
thread   = 5
sleep    = 1
```

1.6.1 thread - Number of crawling thread

This attribute specifies the number of thread Crawler Module of nfrd used to crawl feeds.

1.6.2 sleep - Time to sleep for each round

This attribute specifies the time to sleep for each round threads crawl the database feed table.

1.7 Statistics - Module configuration

```
[Statistics]
period      = 3600
```

1.7.1 period - Time to update or record

This attribute specifies the period of updating or recording (in seconds). For example, by setting period to 3600, the statistics will be updated for each hour.

1.8 mysql - Database connector configuration

```
[mysql]
host      = 127.0.0.1
port      = 3306
database  = newsfeeder
username  = username
password  = password
```

1.8.1 host - Address of the host

This attribute specifies the address of the host. This value can be either host name or IP address.

1.8.2 port - Mysql port of the host

This optional attribute specifies the mysql port of the host. Default value is 3306.

1.8.3 database database - Database schema

This optional attribute specifies which database schema to use

1.8.4 username - Username for authentication

This attribute provides the username for authentication

1.8.5 password - Password for authentication

This attribute provides the password for authentication

Chapter 2

Protocol talks to Front-end

Author

Shiwei Zhang sz653@uow.edu.au

Date

01/05/12

Version

0.2

2.1 Introduction

This page contains protocol definition that talks to the front-end of News Feeder.

For this version of protocol, the protocol is **plaintext** which is more readable for human beings but less efficient and security. In the next version of this protocol will based on binary which is more efficient but less human readable.

In later versions, the protocol will be encapsulated by a security layer.

Warning

The protocol details may be changed during development because we are adding news features and implementation issues.

2.1.1 Sockets

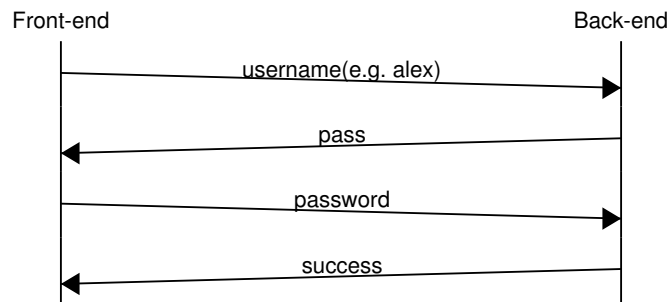
The back-end listens to both unix socket and tcp socket as defined by the config file. The user could turn off unix socket listening or tcp socket listening by modifying the config file.

Basically, for a server that installed both front-end and back-end, unix socket is suggested and turn of the tcp socket in order to prevent external attacks. On the hand, if front-end and back-end are installed in different servers, tcp socket is suggested and turn of the unix socket.

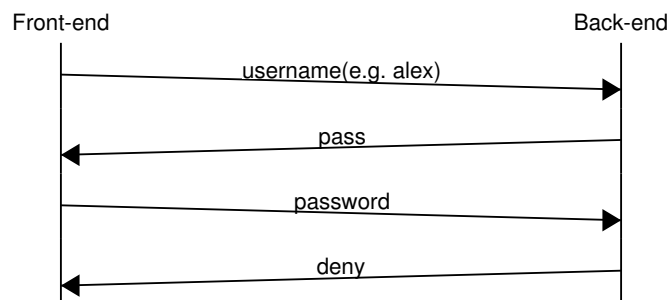
2.2 Login

Before the front-end could communicate with and control back-end, it should login first and be authenticated in order to provide basic security. The front-end must provide a username and password for authentication.

2.2.1 Success case



2.2.2 Fail case

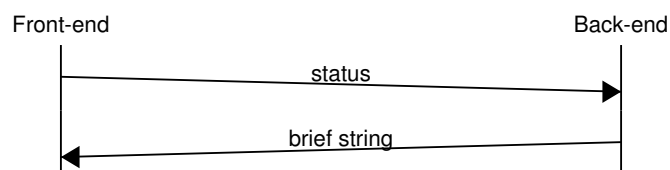


2.3 status - Check the status

The front-end can check the status of the back-end via "status" and combine sub-commands as following:

```
status [sub-command]
```

If sub-command is not provide, the back-end will send a brief string to the front-end. This string may not be parsed by the front-end but is human readable.

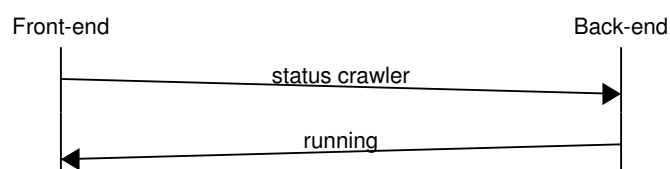


In the later version, statistical information will be available.

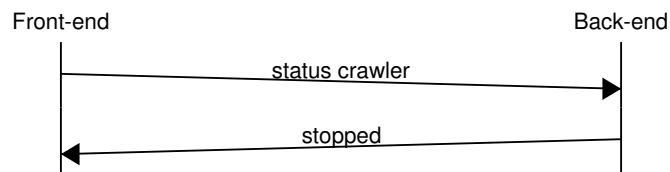
2.3.1 crawler - crawler status

This sub-command "crawler" will tell whether the crawler is running or not.

If the crawler is currently running, it will return "running":



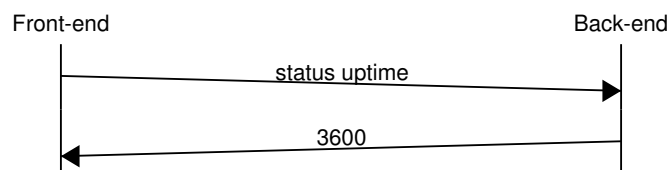
If the crawler is currently stopped, it will return "stopped":



Note: There are 4 statuses: running, stopped, stopping and starting.

2.3.2 uptime - Back-end uptime

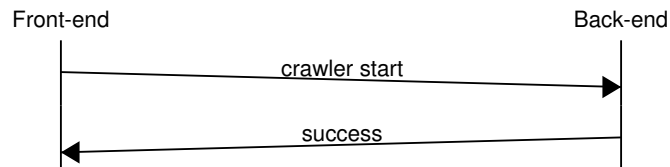
This sub-command "uptime" will return back how many seconds that the back-end has run. For example: (has run for 1 hour)



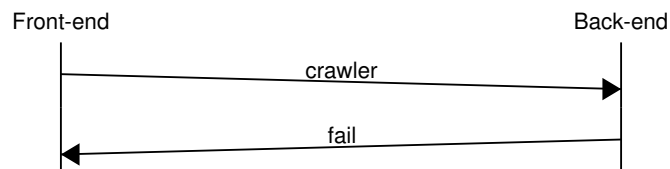
2.4 crawler - Control the crawler

The crawler command in this version of protocol is under the assumption that the back-end is only running one crawler (since the back-end is designed to handle multiple crawler at one time).

Same as command [status - Check the status](#), this command is used with sub-commands while a sub-command is required. For example:

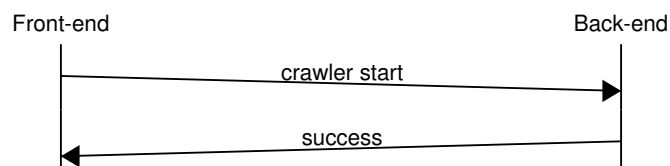


If no sub-command is attached to the command "crawler", the back-end will return a "fail" as following:

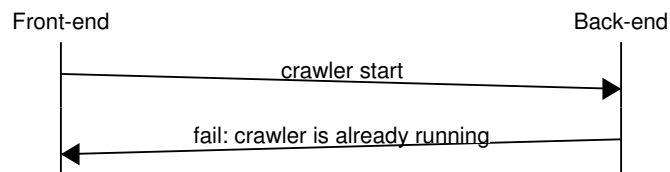


2.4.1 start - Start the crawler

This sub-command "start" tells the back-end to start the crawler:



If exceptions are thrown and caught by the back-end, it will return a "fail" with error message followed by a ": ":



2.4.2 status - Check the status of the crawler

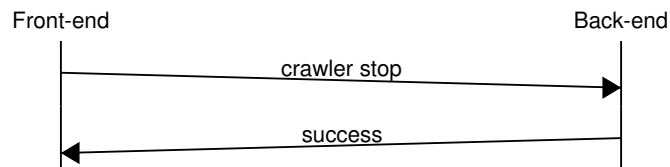
This sub-command "status" is exactly the same as command "status crawler".

See also

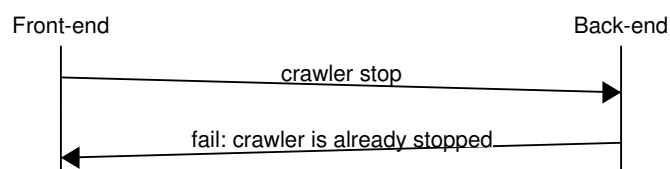
[crawler - crawler status](#)

2.4.3 stop - Stop the crawler

This sub-command "stop" tells the back-end to stop the crawler:

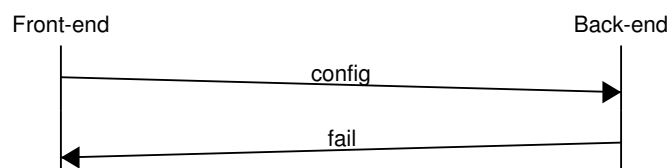


If exceptions are thrown and caught by the back-end, it will return a "fail" with error message followed by a ": ":



2.5 config - Configure back-end

The "config" command provide the direct control of the config information at the back-end. Like [crawler - Control the crawler](#), a sub-command is required when using this command.



See also

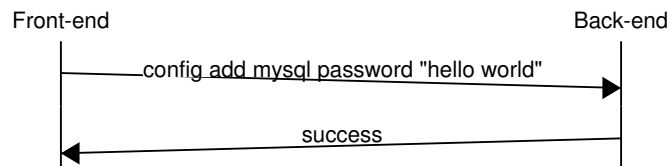
`config::ConfigManager`

2.5.1 add - Add/Change an attribute

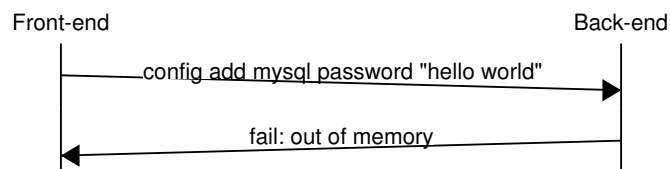
This sub-command "add" tells the back-end to add or change an attribute. The grammar for this command is:

```
config add sector attribute value
```

The sector and attribute should not contain whitespaces. It will take everything after attribute as the value of that attribute. For example:



In this example, "mysql" is parsed as sector name, "password" is parsed as attribute name, and "hello world" is parsed as its value. This command may fail (because of memory problem or others) and it will return a "fail" with error message followed by a ": ":



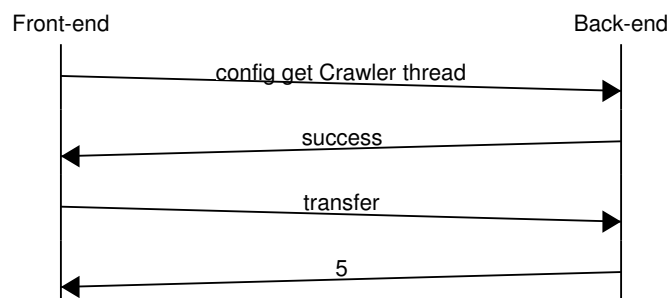
If the required sector or attribute does not exist, it will automatically add a new one. If the attribute is already there, the back-end will change it.

2.5.2 get - Return an attribute

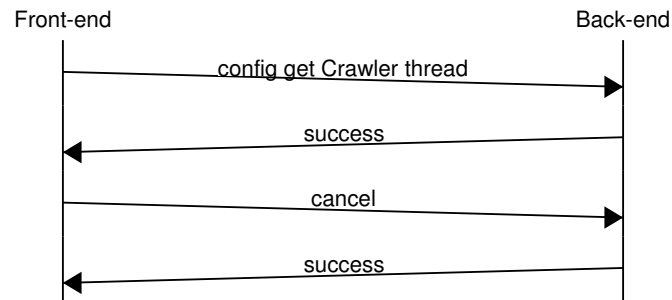
This sub-command "get" tells the back-end to get the value of an attribute. The grammar for this command is:

```
config get sector attribute
```

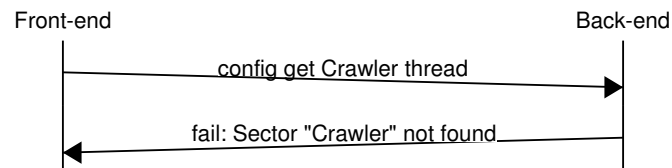
The back-end will search for this attribute as requested. if request is successful, the back-end will return "success". Otherwise, it will return "fail", followed by an error message. Once success, the front-end can get the value back by sending "transfer".



Of course, the front-end can cancel this by sending "cancel" or rubbish.



Also, here is the situation if the sector or the attribute is not available:



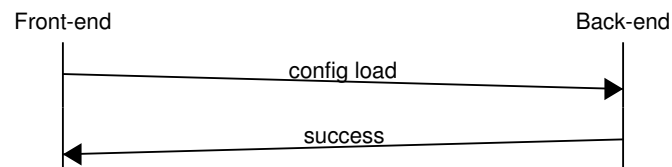
2.5.3 load - Load configuration file

This sub-command "load" tells the back-end to load all configurations from a config file. The grammar for this command is:

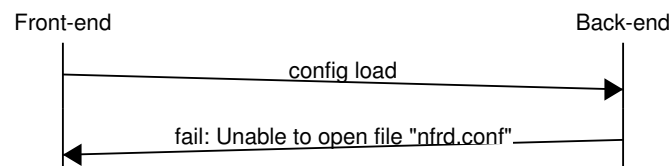
```
config load [file]
```

where "file" is optional. Note: The default config file is "nfrd.conf" which is hard coded in the nfrd program. It will return "success" on success or "fail" followed by error message on failure.

Success case:



Fail case:



See also

[config::ConfigManager::Read\(\)](#)

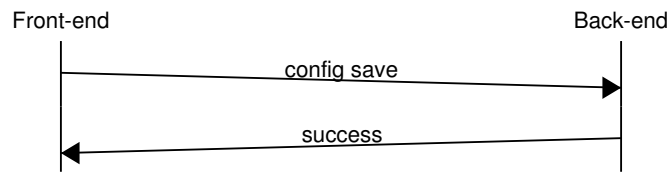
2.5.4 save - Save configuration file

This sub-command "save" tells the back-end to save all configurations to a config file. The grammar for this command is:

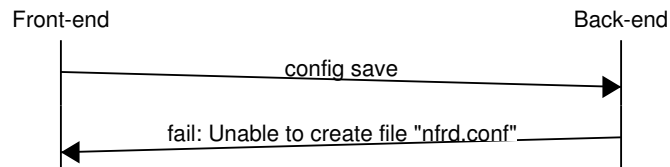
```
config save [file]
```

where "file" is optional. Note: The default config file is "nfrd.conf" which is hard coded in the nfrd program. It will return "success" on success or "fail" followed by error message on failure.

Success case:



Fail case:

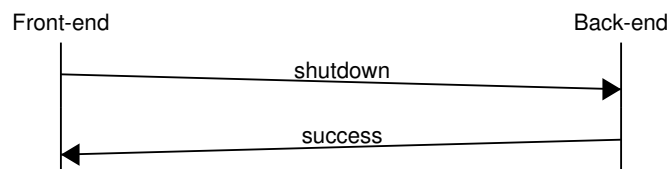


See also

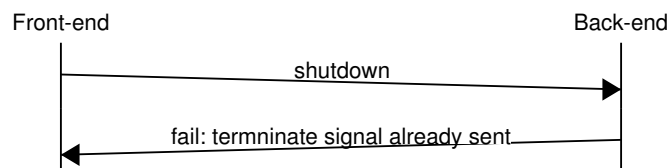
`config::ConfigManager::Write()`

2.6 shutdown - Shutdown the Back-End

Shutdown the Back-End process in a safe way. Stopping all modules and unload them then terminate itself. Success case:



Fail case:

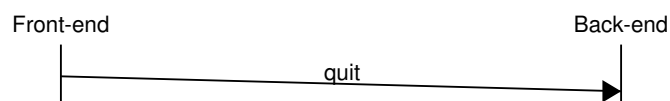


Warning

This command should be used for maintenance only.

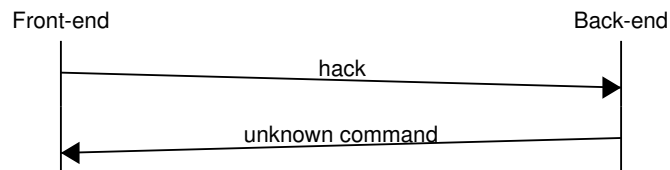
2.7 quit - Close connection

After all communication with the back-end, the front-end should say good-bye, which is "quit" in this case, to the back-end to close the connection between them.



2.8 Unknown command handling

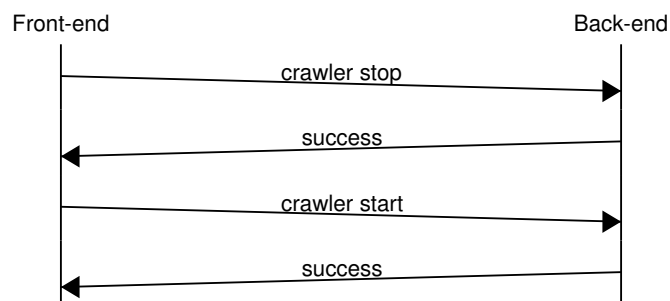
If a command is not supported by the back-end, a message of "unknown command" will be returned.



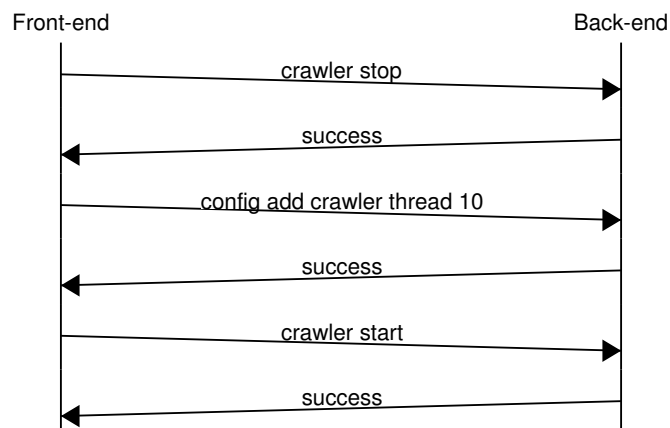
2.9 Examples

Here is some examples for advanced operations.

2.9.1 Restart the crawler



2.9.2 Change the number of thread in crawler



Chapter 3

Command line usage

Author

Shiwei Zhang sz653@uow.edu.au

Date

24/04/12

Version

0.2

3.1 Introduction

This page contains how to run nfrd in command line

3.2 Manual

```
usage: nfrd [options]
options:
  -d          run in daemon mode
  -f file     use specified config file
  -h          show this help
  -v          print the version
```


Chapter 4

Namespace Documentation

4.1 nfrd Namespace Reference

Contains all classes unique to News Feeder Refresh Daemon.

Namespaces

- namespace [config](#)
Contains classes related to configuration.
- namespace [log](#)
Contains classes related to log.
- namespace [misc](#)
Contains classes that commonly used in other classes.
- namespace [module](#)
Contains classes that control different modules.
- namespace [parser](#)
Contains classes related to all kinds of feed parsers.

Classes

- class [Master](#)
Manage, contain and access all components of nfrd.

4.1.1 Detailed Description

Contains all classes unique to News Feeder Refresh Daemon.

Author

Shiwei Zhang sz653@uow.edu.au

Date

21/03/12

4.2 nfrd::config Namespace Reference

Contains classes related to configuration.

Classes

- class [ConfigSector](#)
A part of [ConfigManager](#) (as a container)
- class [ConfigManager](#)
Manages config files (core class).
- class [ConfigException](#)
General exception for config.
- class [IOException](#)
Input/Output exception for config.
- class [ItemNotFound](#)
Item not found.

4.2.1 Detailed Description

Contains classes related to configuration.

4.3 nfrd::log Namespace Reference

Contains classes related to log.

Classes

- class [LogManager](#)
Manage logs.
- class [LogException](#)
General exception for log.
- class [IOException](#)
Input/Output exception for config.

4.3.1 Detailed Description

Contains classes related to log.

4.4 nfrd::misc Namespace Reference

Contains classes that commonly used in other classes.

Namespaces

- namespace [Utility](#)
Contains all utility functions.

Classes

- struct [AutoDBRef](#)
A wrapper class to provide [AutoDB](#) with reference semantics.
- class [AutoDB](#)
A class to mimic `std::AutoDB` specified for `nfdb` usage: `std::AutoDB(std::vector<int>);` When out of scope, this class will deallocate the `int*` in the container automatically.*
- class [DateTime](#)
A class to store date and time.
- class [Image](#)
A class to store image and process image.
- class [ImageException](#)
General exception for [Image](#).

4.4.1 Detailed Description

Contains classes that commonly used in other classes.

4.5 nfrd::misc::Utility Namespace Reference

Contains all utility functions.

Functions

- bool [Read](#) (const char *url, std::vector< char > &container)
Read the content of the url and write to the container.
- bool [Read](#) (const std::string &url, std::vector< char > &container)
This function overloads and calls bool [nfrd::misc::Utility::Read](#)(const char url, std::vector<char> & container).*
- std::auto_ptr< std::vector< char > > [Read](#) (const std::string &url)
This function overloads and calls bool [nfrd::misc::Utility::Read](#)(const char url, std::vector<char> & container).*
- int [ToInt](#) (const std::string &str)
String to integer.
- bool [ToBool](#) (const std::string &str)
String to boolean.
- void [GetArguments](#) (std::istream &in, std::vector< std::string > &args)
Get arguments from a line of a stream.
- std::string [Trim](#) (const std::string &str)
Trim the trailing and ending whitespaces and return a new string.

4.5.1 Detailed Description

Contains all utility functions.

4.5.2 Function Documentation

4.5.2.1 void nfrd::misc::Utility::GetArguments (std::istream & *in*, std::vector< std::string > & *args*)

Get arguments from a line of a stream.

It works like a bash interpreter on Unix in operations of quotes (" and ') and backslash Note: *args* will be truncated

Parameters

<i>in</i>	whether the arguments comes from
<i>args</i>	where the arguments writes to

4.5.2.2 bool nfrd::misc::Utility::Read (const char * *url*, std::vector< char > & *container*)

Read the content of the url and write to the container.

Note: The container will not be cleared by this function. All it does is to append data to the container.

This function will follow the url redirections (max -> 255).

Parameters

<i>url</i>	URL of the file
<i>container</i>	where the data write to

Returns

true if success
false if fails to obtain the file

4.5.2.3 bool nfrd::misc::Utility::Read (const std::string & *url*, std::vector< char > & *container*)

This function overloads and calls bool [nfrd::misc::Utility::Read](#)(const char* url, std::vector<char>& container).

It takes std::string for url instead of const char*

Parameters

<i>url</i>	URL of the file
<i>container</i>	where the data write to

Returns

true if success
false if fails to obtain the file

4.5.2.4 std::auto_ptr<std::vector<char> > nfrd::misc::Utility::Read (const std::string & *url*)

This function overloads and calls bool [nfrd::misc::Utility::Read](#)(const char* url, std::vector<char>& container).

Instead of passing container to the function, it returns the data via an auto pointer.

Parameters

<i>url</i>	URL of the file
------------	-----------------

Returns

the file data

Exceptions

<i>length_error</i>	if fail to fetch the data that url points to
---------------------	--

4.5.2.5 bool nfrd::misc::Utility::ToBool (const std::string & *str*)

String to boolean.

This function is equivalent to (bool)atoi(str.c_str());

Parameters

<i>str</i>	source string
------------	---------------

Returns

converted boolean

4.5.2.6 int nfrd::misc::Utility::ToInt (const std::string & *str*)

String to integer.

This function is equivalent to atoi(str.c_str());

Parameters

<i>str</i>	source string
------------	---------------

Returns

converted integer

4.5.2.7 std::string nfrd::misc::Utility::Trim (const std::string & *str*)

Trim the trailing and ending whitespaces and return a new string.

Parameters

<i>str</i>	source string
------------	---------------

Returns

trimmed string

4.6 nfrd::module Namespace Reference

Contains classes that control different modules.

Classes

- class [AdminService](#)
Manage sockets talk to the front end and interacts with other components.
- class [AdminServiceThread](#)
Handle sockets talk to the front end and interacts with other components.
- class [Crawler](#)
The main class representing the crawler module The responsibilities of this class:
- class [CrawlerThread](#)
A worker class representing a thread.
- class [FeedPriorityQueue](#)
Implements a queueing/threading model.
- class [Module](#)
A generalised module interface class, providing all the interfaces of a module that start or stop.
- class [ModuleException](#)
General exception for module.
- class [QueueItem](#)
- class [Statistics](#)
Periodly record and update the statistics information.

4.6.1 Detailed Description

Contains classes that control different modules.

4.7 nfrd::parser Namespace Reference

Contains classes related to all kinds of feed parsers.

Classes

- class [AtomItem](#)
A class to store details of an item obtained by the [AtomParser](#).
- class [AtomParser](#)
A parser to parse Atom feeds.
- class [FeedItem](#)
A class to store details of an item obtained by the [FeedParser](#).
- class [FeedParser](#)
A parser to parse web feeds.
- class [FeedXParser](#)
A parser to parse web feeds with Patch and Match feature.
- class [Item](#)
A class to store details of an item obtained by the [Parser](#).
- class [Parser](#)
A generalised parser interface class, providing all the interfaces of a class that reads resource from an URL and parse it into a list of [Item](#).
- class [ParserException](#)
General exception for parser.
- class [HasNoValue](#)
Has no value exception.
- class [InvalidSource](#)

Invalid source exception.

- class [RSSItem](#)

A class to store details of an item obtained by the [RSSParser](#).

- class [RSSParser](#)

A parser to parse RSS feeds.

4.7.1 Detailed Description

Contains classes related to all kinds of feed parsers.

Chapter 5

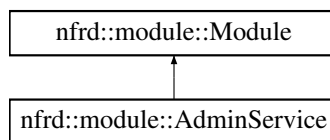
Class Documentation

5.1 nfrd::module::AdminService Class Reference

Manage sockets talk to the front end and interacts with other components.

```
#include <AdminService.h>
```

Inheritance diagram for nfrd::module::AdminService:



Public Member Functions

- [AdminService](#) ([Master](#) &[master](#), const [config::ConfigManager](#) &[config](#), const [log::LogManager](#) &[log](#))
Initialising Constructor for AdminService.
- [~AdminService](#) ()
Delete all dynamic memory, if any.
- void [operator](#)() ()
Start the service/module in current thread.
- void [Stop](#) ()
Stop the service/module, joining the thread.
- bool [Authenticate](#) (const std::string &[username](#), const std::string &[password](#)) const
Authenticate the username and password.
- void [Register](#) ([AdminServiceThread](#) *[thread](#)) const
Register the thread in the member: threads.
- void [Exit](#) ([AdminServiceThread](#) *[thread](#), bool dynamic=true) const
This function should be call when a thread exits.

Private Attributes

- [Master](#) & [master](#)
A reference to Master. Hence, be able to control Master.
- boost::asio::ip::tcp::acceptor * [acceptor](#)
A pointer to the acceptor used in operator()/()

- `std::set< AdminServiceThread * > threads`
Track all detached threads;.
- `boost::mutex thread_mutex`
A mutex for operating on the member: threads.
- `boost::condition condition`
Condition variable that used in [Stop\(\)](#) for waiting all AdminServiceThreads.
- `std::string username`
Authentication username.
- `std::string password`
Authentication password.

Additional Inherited Members

5.1.1 Detailed Description

Manage sockets talk to the front end and interacts with other components.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 `AdminService::AdminService (Master & master, const config::ConfigManager & config, const log::LogManager & log)`

Initialising Constructor for [AdminService](#).

Parameters

<i>master</i>	Master of nfrd
<i>config</i>	config manager
<i>log</i>	logger

5.1.2.2 `AdminService::~~AdminService ()`

Delete all dynamic memory, if any.

5.1.3 Member Function Documentation

5.1.3.1 `bool AdminService::Authenticate (const std::string & username, const std::string & password) const`

Authenticate the username and password.

Parameters

<i>username</i>	username from front-end
<i>password</i>	password from front-end

Returns

true if passed. Otherwise, fail returned

5.1.3.2 `void AdminService::Exit (AdminServiceThread * thread, bool dynamic = true) const`

This function should be call when a thread exits.

If dynamic is true, then no more operations are allowed on the instance.

Parameters

<i>thread</i>	finished AdminServiceThread
<i>dynamic</i>	whether the thread is dynamic allocated if true. delete is called

5.1.3.3 void AdminService::operator()() [virtual]

Start the service/module in current thread.

Implements [nfrd::module::Module](#).

5.1.3.4 void AdminService::Register (AdminServiceThread * thread) const

Register the thread in the member: threads.

Parameters

<i>thread</i>	AdminServiceThread just started
---------------	---

5.1.3.5 void AdminService::Stop () [virtual]

Stop the service/module, joining the thread.

Reimplemented from [nfrd::module::Module](#).

5.1.4 Member Data Documentation

5.1.4.1 boost::asio::ip::tcp::acceptor* nfrd::module::AdminService::acceptor [private]

A pointer to the acceptor used in [operator>\(\)](#)

5.1.4.2 boost::condition nfrd::module::AdminService::condition [mutable], [private]

Condition variable that used in [Stop\(\)](#) for waiting all AdminServiceThreads.

5.1.4.3 Master& nfrd::module::AdminService::master [private]

A reference to [Master](#). Hence, be able to control [Master](#).

5.1.4.4 std::string nfrd::module::AdminService::password [private]

Authentication password.

5.1.4.5 boost::mutex nfrd::module::AdminService::thread_mutex [mutable], [private]

A mutex for operating on the member: threads.

5.1.4.6 std::set<AdminServiceThread*> nfrd::module::AdminService::threads [mutable], [private]

Track all detached threads;.

5.1.4.7 `std::string nfrd::module::AdminService::username` [private]

Authentication username.

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

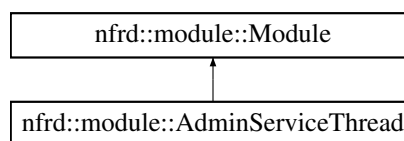
- include/nfrd/[AdminService.h](#)
- src/[AdminService.cpp](#)

5.2 `nfrd::module::AdminServiceThread` Class Reference

Handle sockets talk to the front end and interacts with other components.

```
#include <AdminServiceThread.h>
```

Inheritance diagram for `nfrd::module::AdminServiceThread`:



Public Member Functions

- [AdminServiceThread](#) (const [AdminService](#) &parent, [Master](#) &master, unsigned int timeout=60, bool dynamic=true)
Initialising Constructor for AdminServiceThread.
- [~AdminServiceThread](#) ()
Delete all dynamic memory, if any.
- void [operator](#)() ()
Handle basic communication and do authentication.
- boost::asio::ip::tcp::iostream & [GetStream](#) ()
Get the stream associated in this thread.

Private Member Functions

- void [Handle](#) ()
Handle commands request from the front-end.
- std::istream & [sin](#) ()
Use the socket stream as istream, refreshing time expire.
- std::ostream & [sout](#) ()
Use the socket stream as ostream, refreshing time expire.
- void [status_main](#) (const std::vector< std::string > &args)
status - Check the status
- void [crawler_main](#) (const std::vector< std::string > &args)
crawler - Control the crawler
- void [config_main](#) (const std::vector< std::string > &args)
config - Configure back-end
- void [shutdown_main](#) (const std::vector< std::string > &args)
shutdown - Shutdown back-end
- void [status_send_module_status](#) (const std::string &name)
Used by status_main(), sending status of a module.

Private Attributes

- boost::asio::ip::tcp::iostream [stream](#)
Socket stream.
- const [AdminService](#) & [parent](#)
Keep reference to its parrent.
- [Master](#) & [master](#)
Keep reference to the [Master](#) to be handled.
- boost::posix_time::seconds [timeout](#)
General operation time out limit.
- bool [dynamic](#)
Whether this instance is dynamically allocated.
- std::string [remote_address](#)
Keep track of the address of the remote host.

Additional Inherited Members

5.2.1 Detailed Description

Handle sockets talk to the front end and interacts with other components.

See also

[Protocol talks to Front-end](#)

5.2.2 Constructor & Destructor Documentation

5.2.2.1 `AdminServiceThread::AdminServiceThread (const AdminService & parent, Master & master, unsigned int timeout = 60, bool dynamic = true)`

Initialising Constructor for [AdminServiceThread](#).

Parameters

<i>parent</i>	parent of this thread
<i>master</i>	Master of nfrd to be controled
<i>timeout</i>	timeout on socket operation
<i>dynamic</i>	is this instance dynamically allocated

5.2.2.2 `AdminServiceThread::~~AdminServiceThread ()`

Delete all dynamic memory, if any.

5.2.3 Member Function Documentation

5.2.3.1 `void AdminServiceThread::config_main (const std::vector< std::string > & args) [private]`

config - Configure back-end

Parameters

<i>args</i>	argument list
-------------	---------------

See also

[config - Configure back-end](#)

5.2.3.2 `void AdminServiceThread::crawler_main (const std::vector< std::string > & args)` `[private]`

crawler - Control the crawler

Parameters

<i>args</i>	argument list
-------------	---------------

See also

[crawler - Control the crawler](#)

5.2.3.3 `boost::asio::ip::tcp::iostream & AdminServiceThread::GetStream ()`

Get the stream associated in this thread.

Returns

socket stream

5.2.3.4 `void AdminServiceThread::Handle ()` `[private]`

Handle commands request from the front-end.

5.2.3.5 `void AdminServiceThread::operator()()` `[virtual]`

Handle baisc communication and do authentication.

Implements [nfrd::module::Module](#).

5.2.3.6 `void AdminServiceThread::shutdown_main (const std::vector< std::string > & args)` `[private]`

shutdown - Shutdown back-end

Parameters

<i>args</i>	argument list
-------------	---------------

See also

[shutdown - Shutdown the Back-End](#)

5.2.3.7 `std::istream & AdminServiceThread::sin ()` `[private]`

Use the socket stream as istream, refreshing time expire.

Returns

istream of stream

5.2.3.8 `std::ostream & AdminServiceThread::sout ()` [private]

Use the socket stream as ostream, refreshing time expire.

Returns

ostream of stream

5.2.3.9 `void AdminServiceThread::status_main (const std::vector< std::string > & args)` [private]

status - Check the status

Parameters

<i>args</i>	argument list
-------------	---------------

See also

[status - Check the status](#)

5.2.3.10 `void AdminServiceThread::status_send_module_status (const std::string & name)` [private]

Used by [status_main\(\)](#), sending status of a module.

For example: name: unloaded

Parameters

<i>name</i>	module name
-------------	-------------

5.2.4 Member Data Documentation

5.2.4.1 `bool nfrd::module::AdminServiceThread::dynamic` [private]

Whether this instance is dynamically allocated.

5.2.4.2 `Master& nfrd::module::AdminServiceThread::master` [private]

Keep reference to the [Master](#) to be handled.

5.2.4.3 `const AdminService& nfrd::module::AdminServiceThread::parent` [private]

Keep reference to its parent.

5.2.4.4 `std::string nfrd::module::AdminServiceThread::remote_address` [private]

Keep track of the address of the remote host.

5.2.4.5 `boost::asio::ip::tcp::iostream nfrd::module::AdminServiceThread::stream` [private]

Socket stream.

5.2.4.6 boost::posix_time::seconds nfrd::module::AdminServiceThread::timeout [private]

General operation time out limit.

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

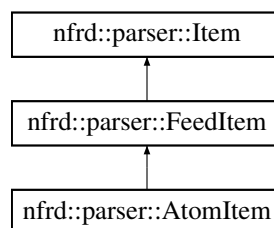
- include/nfrd/AdminServiceThread.h
- src/AdminServiceThread.cpp

5.3 nfrd::parser::AtomItem Class Reference

A class to store details of an item obtained by the [AtomParser](#).

```
#include <AtomParser.h>
```

Inheritance diagram for nfrd::parser::AtomItem:



Public Member Functions

- [AtomItem](#) ()
Initialising Constructor for [AtomItem](#).
- [~AtomItem](#) ()
Delete all dynamic memory, if any.
- const std::string & [GetTitle](#) () const
Get the title of the item.
- const std::string & [GetURL](#) () const
Get the URL where full edition of the item is.
- const std::string & [GetContent](#) () const
Get the content of the item.
- const [misc::DateTime](#) & [GetPostDate](#) () const
Get the post date of the item.
- const std::string & [GetAuthor](#) () const
Get the author of the item.
- void [SetTitle](#) (const char *source)
Set the title of the item.
- void [SetURL](#) (const char *source)
Set the url of the item.
- void [SetContent](#) (const char *source)
Set the content of the item.
- void [SetContent](#) (const std::string &source)
Set the content of the item.
- void [SetPostDate](#) (const [misc::DateTime](#) *source)
Set the post date of the item.
- void [SetAuthor](#) (const char *source)

- *Set the author of the item.*
- bool [HasTitle](#) () const
Test the item has title or not.
- bool [HasURL](#) () const
Test the item has URL or not.
- bool [HasContent](#) () const
Test the item has content or not.
- bool [HasPostDate](#) () const
Test the item has post date or not.
- bool [HasAuthor](#) () const
Test the item has author or not.

Private Attributes

- std::string * [title](#)
Title of the item.
- std::string * [url](#)
URL of the item.
- std::string * [content](#)
Content of the item.
- [misc::DateTime](#) * [postDate](#)
Post date of the item.
- std::string * [author](#)
Author of the item.

5.3.1 Detailed Description

A class to store details of an item obtained by the [AtomParser](#).

Note: All item components are actually stored in the `AtomParser::doc`

See also

[FeedItem](#)

5.3.2 Constructor & Destructor Documentation

5.3.2.1 AtomItem::AtomItem ()

Initialising Constructor for [AtomItem](#).

Initialise everything to zero/null.

5.3.2.2 AtomItem::~~AtomItem ()

Delete all dynamic memory, if any.

5.3.3 Member Function Documentation

5.3.3.1 const string & AtomItem::GetAuthor () const [virtual]

Get the author of the item.

Returns

author

Exceptions

<i>HasNoValue</i>	if the item has no author or if this function is not overridden
-------------------	---

Reimplemented from [nfrd::parser::Item](#).

5.3.3.2 const string & AtomItem::GetContent () const [virtual]

Get the content of the item.

Returns

content

Exceptions

<i>HasNoValue</i>	if the item has no content or if this function is not overridden
-------------------	--

Reimplemented from [nfrd::parser::Item](#).

5.3.3.3 const DateTime & AtomItem::GetPostDate () const [virtual]

Get the post date of the item.

Returns

post date

Exceptions

<i>HasNoValue</i>	if the item has no post date or if this function is not overridden
-------------------	--

Reimplemented from [nfrd::parser::Item](#).

5.3.3.4 const string & AtomItem::GetTitle () const [virtual]

Get the title of the item.

Returns

title

Exceptions

<i>HasNoValue</i>	if the item has no title or if this function is not overridden
-------------------	--

Reimplemented from [nfrd::parser::Item](#).

5.3.3.5 const string & AtomItem::GetURL () const [virtual]

Get the URL where full edition of the item is.

Returns

URL

Exceptions

<i>HasNoValue</i>	if the item has no URL or if this function is not overridden
-------------------	--

Reimplemented from [nfrd::parser::Item](#).

5.3.3.6 bool AtomItem::HasAuthor () const [virtual]

Test the item has author or not.

Returns

true If the item has author
false If the item has no title or this function is not overridden

Reimplemented from [nfrd::parser::Item](#).

5.3.3.7 bool AtomItem::HasContent () const [virtual]

Test the item has content or not.

Returns

true If the item has content
false If the item has no content or this function is not overridden

Reimplemented from [nfrd::parser::Item](#).

5.3.3.8 bool AtomItem::HasPostDate () const [virtual]

Test the item has post date or not.

Returns

true If the item has post date
false If the item has no post date or this function is not overridden

Reimplemented from [nfrd::parser::Item](#).

5.3.3.9 bool AtomItem::HasTitle () const [virtual]

Test the item has title or not.

Returns

true If the item has title
false If the item has no title or this function is not overridden

Reimplemented from [nfrd::parser::Item](#).

5.3.3.10 `bool AtomItem::HasURL () const` `[virtual]`

Test the item has URL or not.

Returns

true If the item has URL
false If the item has no URL or this function is not overridden

Reimplemented from [nfrd::parser::Item](#).

5.3.3.11 `void AtomItem::SetAuthor (const char * source)`

Set the author of the item.

Parameters

<i>source</i>	author of the item
---------------	--------------------

5.3.3.12 `void AtomItem::SetContent (const char * source)`

Set the content of the item.

Parameters

<i>source</i>	content of the item
---------------	---------------------

5.3.3.13 `void nfrd::parser::AtomItem::SetContent (const std::string & source)` `[virtual]`

Set the content of the item.

Parameters

<i>source</i>	content of the item
---------------	---------------------

See also

[FeedItem](#)

Implements [nfrd::parser::FeedItem](#).

5.3.3.14 `void AtomItem::SetPostDate (const misc::DateTime * source)`

Set the post date of the item.

Parameters

<i>source</i>	post date of the item
---------------	-----------------------

5.3.3.15 `void AtomItem::SetTitle (const char * source)`

Set the title of the item.

Parameters

<i>source</i>	title of the item
---------------	-------------------

5.3.3.16 void AtomItem::SetURL (const char * *source*)

Set the url of the item.

Parameters

<i>source</i>	url of the item
---------------	-----------------

5.3.4 Member Data Documentation

5.3.4.1 std::string* nfrd::parser::AtomItem::author [private]

Author of the item.

Required in Atom. Original tag in Atom: author | name

5.3.4.2 std::string* nfrd::parser::AtomItem::content [private]

Content of the item.

Optional in Atom. Original tag in Atom: content

5.3.4.3 misc::DateTime* nfrd::parser::AtomItem::postDate [private]

Post date of the item.

Required in Atom. Original tag in Atom: updated

5.3.4.4 std::string* nfrd::parser::AtomItem::title [private]

Title of the item.

Required in Atom. Original tag in Atom: title

5.3.4.5 std::string* nfrd::parser::AtomItem::url [private]

URL of the item.

Optional in Atom. Original tag in Atom: link rel="alternate"

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

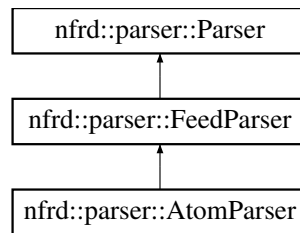
- include/nfrd/[AtomParser.h](#)
- src/[AtomParser.cpp](#)

5.4 nfrd::parser::AtomParser Class Reference

A parser to parse Atom feeds.

```
#include <AtomParser.h>
```

Inheritance diagram for nfrd::parser::AtomParser:



Public Member Functions

- [AtomParser](#) ()
Initialising Constructor for [AtomParser](#).
- virtual [~AtomParser](#) ()
Delete all dynamic memory, if any.
- void [ReadDom](#) (const rapidxml::xml_document<> &doc)
Parse feed from a dom tree (xml document) into a list of [Item](#).
- const [misc::DateTime](#) & [GetLastBuildDate](#) () const
Get the last build date of the feed resource.

Protected Attributes

- [misc::DateTime](#) * [lastBuildDate](#)
Last build date of the Atom feed Required in Atom.

5.4.1 Detailed Description

A parser to parse Atom feeds.

Standard: <http://tools.ietf.org/html/rfc4287>

See also

[Parser](#)

5.4.2 Constructor & Destructor Documentation

5.4.2.1 AtomParser::AtomParser ()

Initialising Constructor for [AtomParser](#).

5.4.2.2 AtomParser::~~AtomParser () [virtual]

Delete all dynamic memory, if any.

5.4.3 Member Function Documentation

5.4.3.1 const DateTime & AtomParser::GetLastBuildDate () const [virtual]

Get the last build date of the feed resource.

Usually, this data is provided in the feed resource, telling when the feed resource is generated. Some subclasses may use pseudo-LastBuildDate that the date is the post date of the latest item.

Returns

last build date of the feed resource

Exceptions

<i>HasNoValue</i>	if the item has no last build date
-----------------------------------	------------------------------------

Reimplemented from [nfrd::parser::Parser](#).

5.4.3.2 void AtomParser::ReadDom (const rapidxml::xml.document<> & *doc*) [virtual]

Parse feed from a dom tree (xml document) into a list of [Item](#).

Parameters

<i>doc</i>	parsed xml document of the feed resource
------------	--

Exceptions

<i>InvalidSource</i>	if the dom or the feed resource is invalid
--------------------------------------	--

Implements [nfrd::parser::FeedParser](#).

5.4.4 Member Data Documentation

5.4.4.1 misc::DateTime* nfrd::parser::AtomParser::lastBuildDate [protected]

Last build date of the Atom feed Required in Atom.

Original tag in Atom: updated

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

- include/nfrd/[AtomParser.h](#)
- src/[AtomParser.cpp](#)

5.5 nfrd::misc::AutoDB< _Tp > Class Template Reference

A class to mimic std::AutoDB specified for nfdb usage: std::AutoDB(std::vector<int *>); When out of scope, this class will deallocate the int* in the container automatically.

```
#include <AutoDB.h>
```

Public Types

- typedef [_Tp element_type](#)
The pointed-to type.

Public Member Functions

- [AutoDB](#) ([element_type](#) *__p=0) throw ()
An AutoDB is usually constructed from a raw pointer.
- [AutoDB](#) ([AutoDB](#) &__a) throw ()

- An AutoDB can be constructed from another AutoDB.*
- `template<typename _Tp1 >`
`AutoDB (AutoDB< _Tp1 > &__a) throw ()`
An AutoDB can be constructed from another AutoDB.
- `AutoDB & operator= (AutoDB &__a) throw ()`
AutoDB assignment operator.
- `template<typename _Tp1 >`
`AutoDB & operator= (AutoDB< _Tp1 > &__a) throw ()`
AutoDB assignment operator.
- `~AutoDB ()`
When the AutoDB goes out of scope, the object it owns is deleted.
- `element_type & operator* () const throw ()`
Smart pointer dereferencing.
- `element_type * operator-> () const throw ()`
Smart pointer dereferencing.
- `element_type * get () const throw ()`
Bypassing the smart pointer.
- `element_type * release () throw ()`
Bypassing the smart pointer.
- `void reset (element_type * __p=0) throw ()`
Forcibly deletes the managed object.
- `AutoDB (AutoDBRef< element_type > __ref) throw ()`
Automatic conversions.
- `AutoDB & operator= (AutoDBRef< element_type > __ref) throw ()`
- `template<typename _Tp1 >`
`operator AutoDBRef< _Tp1 > () throw ()`
- `template<typename _Tp1 >`
`operator AutoDB< _Tp1 > () throw ()`

Private Attributes

- `_Tp * _M_ptr`

5.5.1 Detailed Description

`template<typename _Tp>class nfrd::misc::AutoDB< _Tp >`

A class to mimic `std::AutoDB` specified for `nfdb` usage: `std::AutoDB(std::vector<int *>)`; When out of scope, this class will deallocate the `int*` in the container automatically.

Warning

this class does not suitable for those containers which takes more than 1 types, such as `std::map`

Note

this class is derived from the source of `std::AutoDB` the Copyright is refer to GNU license

5.5.2 Member Typedef Documentation

5.5.2.1 `template<typename _Tp> typedef _Tp nfrd::misc::AutoDB< _Tp >::element_type`

The pointed-to type.

5.5.3 Constructor & Destructor Documentation

5.5.3.1 `template<typename _Tp> nfrd::misc::AutoDB<_Tp>::AutoDB (element_type * __p = 0) throw ()`
`[inline], [explicit]`

An AutoDB is usually constructed from a raw pointer.

Parameters

<i>p</i>	A pointer (defaults to NULL).
----------	-------------------------------

This object now *owns* the object pointed to by *p*.

5.5.3.2 `template<typename _Tp> nfrd::misc::AutoDB<_Tp>::AutoDB (AutoDB<_Tp> & __a) throw ()`
`[inline]`

An AutoDB can be constructed from another AutoDB.

Parameters

<i>a</i>	Another AutoDB of the same type.
----------	----------------------------------

This object now *owns* the object previously owned by *a*, which has given up ownership.

5.5.3.3 `template<typename _Tp> template<typename _Tp1> nfrd::misc::AutoDB<_Tp>::AutoDB (AutoDB<_Tp1> & __a) throw ()` `[inline]`

An AutoDB can be constructed from another AutoDB.

Parameters

<i>a</i>	Another AutoDB of a different but related type.
----------	---

A pointer-to-Tp1 must be convertible to a pointer-to-Tp/element_type.

This object now *owns* the object previously owned by *a*, which has given up ownership.

5.5.3.4 `template<typename _Tp> nfrd::misc::AutoDB<_Tp>::~~AutoDB ()` `[inline]`

When the AutoDB goes out of scope, the object it owns is deleted.

If it no longer owns anything (i.e., `get ()` is NULL), then this has no effect.

5.5.3.5 `template<typename _Tp> nfrd::misc::AutoDB<_Tp>::AutoDB (AutoDBRef<element_type> __ref)`
`throw ()` `[inline]`

Automatic conversions.

These operations convert an AutoDB into and from an [AutoDBRef](#) automatically as needed. This allows constructs such as

```
AutoDB<Derived> func_returning_AutoDB(....);
...
AutoDB<Base> ptr = func_returning_AutoDB(....);
```

5.5.4 Member Function Documentation

5.5.4.1 `template<typename _Tp> element_type* nfrd::misc::AutoDB<_Tp>::get () const throw () [inline]`

Bypassing the smart pointer.

Returns

The raw pointer being managed.

You can get a copy of the pointer that this object owns, for situations such as passing to a function which only accepts a raw pointer.

Note

This AutoDB still owns the memory.

5.5.4.2 `template<typename _Tp> template<typename _Tp1 > nfrd::misc::AutoDB<_Tp>::operator AutoDB<_Tp1> () throw () [inline]`

5.5.4.3 `template<typename _Tp> template<typename _Tp1 > nfrd::misc::AutoDB<_Tp>::operator AutoDBRef<_Tp1> () throw () [inline]`

5.5.4.4 `template<typename _Tp> element_type& nfrd::misc::AutoDB<_Tp>::operator* () const throw () [inline]`

Smart pointer dereferencing.

If this AutoDB no longer owns anything, then this operation will crash. (For a smart pointer, "no longer owns anything" is the same as being a null pointer, and you know what happens when you dereference one of those...)

5.5.4.5 `template<typename _Tp> element_type* nfrd::misc::AutoDB<_Tp>::operator-> () const throw () [inline]`

Smart pointer dereferencing.

This returns the pointer itself, which the language then will automatically cause to be dereferenced.

5.5.4.6 `template<typename _Tp> AutoDB& nfrd::misc::AutoDB<_Tp>::operator= (AutoDB<_Tp> & __a) throw () [inline]`

AutoDB assignment operator.

Parameters

<i>a</i>	Another AutoDB of the same type.
----------	----------------------------------

This object now *owns* the object previously owned by *a*, which has given up ownership. The object that this one *used* to own and track has been deleted.

5.5.4.7 `template<typename _Tp> template<typename _Tp1 > AutoDB& nfrd::misc::AutoDB<_Tp>::operator= (AutoDB<_Tp1> & __a) throw () [inline]`

AutoDB assignment operator.

Parameters

<i>a</i>	Another AutoDB of a different but related type.
----------	---

A pointer-to-Tp1 must be convertible to a pointer-to-Tp/element_type.

This object now *owns* the object previously owned by *a*, which has given up ownership. The object that this one *used* to own and track has been deleted.

```
5.5.4.8 template<typename _Tp> AutoDB& nfrd::misc::AutoDB< _Tp >::operator= ( AutoDBRef< element_type
    > __ref ) throw () [inline]
```

```
5.5.4.9 template<typename _Tp> element_type* nfrd::misc::AutoDB< _Tp >::release ( ) throw () [inline]
```

Bypassing the smart pointer.

Returns

The raw pointer being managed.

You can get a copy of the pointer that this object owns, for situations such as passing to a function which only accepts a raw pointer.

Note

This AutoDB no longer owns the memory. When this object goes out of scope, nothing will happen.

```
5.5.4.10 template<typename _Tp> void nfrd::misc::AutoDB< _Tp >::reset ( element_type * __p = 0 ) throw ()
    [inline]
```

Forcibly deletes the managed object.

Parameters

<i>p</i>	A pointer (defaults to NULL).
----------	-------------------------------

This object now *owns* the object pointed to by *p*. The previous object has been deleted.

5.5.5 Member Data Documentation

```
5.5.5.1 template<typename _Tp> _Tp* nfrd::misc::AutoDB< _Tp >::M_ptr [private]
```

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

- include/nfrd/[AutoDB.h](#)

5.6 nfrd::misc::AutoDBRef< _Tp1 > Struct Template Reference

A wrapper class to provide [AutoDB](#) with reference semantics.

```
#include <AutoDB.h>
```

Public Member Functions

- [AutoDBRef](#) (_Tp1 * __p)

Public Attributes

- `_Tp1 * _M_ptr`

5.6.1 Detailed Description

```
template<typename _Tp1> struct nfrd::misc::AutoDBRef< _Tp1 >
```

A wrapper class to provide [AutoDB](#) with reference semantics.

For example, an [AutoDB](#) can be assigned (or constructed from) the result of a function which returns an [AutoDB](#) by value.

All the [AutoDBRef](#) stuff should happen behind the scenes.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 `template<typename _Tp1> nfrd::misc::AutoDBRef< _Tp1 >::AutoDBRef (_Tp1 * __p) [inline], [explicit]`

5.6.3 Member Data Documentation

5.6.3.1 `template<typename _Tp1> _Tp1* nfrd::misc::AutoDBRef< _Tp1 >::_M_ptr`

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

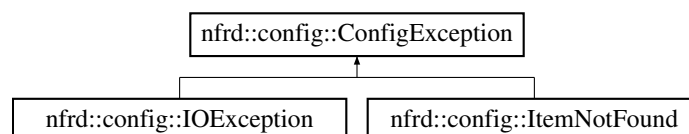
- `include/nfrd/AutoDB.h`

5.7 nfrd::config::ConfigException Class Reference

General exception for config.

```
#include <ConfigManager.h>
```

Inheritance diagram for `nfrd::config::ConfigException`:



Public Member Functions

- [ConfigException](#) (const std::string &message)
Default Constructor for [ConfigException](#), recording the error message.
- virtual [~ConfigException](#) () throw ()
Delete dynamic memories, if any.
- virtual const char * [what](#) () const throw ()
Return error message.

Private Attributes

- `std::string msg`
Error message.

5.7.1 Detailed Description

General exception for config.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 ConfigException::ConfigException (const std::string & *message*) [explicit]

Default Constructor for [ConfigException](#), recording the error message.

Parameters

<i>message</i>	Error message
----------------	---------------

5.7.2.2 ConfigException::~ConfigException () throw () [virtual]

Delete dynamic memories, if any.

5.7.3 Member Function Documentation

5.7.3.1 const char * ConfigException::what () const throw () [virtual]

Return error message.

Returns

Error message

5.7.4 Member Data Documentation

5.7.4.1 std::string nfrd::config::ConfigException::msg [private]

Error message.

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

- `include/nfrd/ConfigManager.h`
- `src/ConfigManager.cpp`

5.8 nfrd::config::ConfigManager Class Reference

Manages config files (core class).

```
#include <ConfigManager.h>
```

Public Member Functions

- [ConfigManager](#) ()
Default Constructor for [ConfigManager](#).
- [ConfigManager](#) (const std::string &filename)
Initialising Constructor for [ConfigManager](#).
- [~ConfigManager](#) ()
Delete all dynamic memory, if any.
- void [Write](#) () const
Write out [ConfigManager](#) to file with default settings.
- void [Write](#) (const std::string &filename) const
Write out [ConfigManager](#) to file.
- void [Read](#) ()
Read in [ConfigManager](#) from file with default settings.
- void [Read](#) (const std::string &filename)
Read in [ConfigManager](#) from file.
- void [SetFileName](#) (const std::string &filename)
Externally set the file name of the config file.
- [ConfigSector](#) & [operator\[\]](#) (const std::string §or)
Fetch the sector of the config If the sector is not existed, a new entry will be created.
- const [ConfigSector](#) & [operator\[\]](#) (const std::string §or) const
Fetch the sector of the config in const form If the sector is not existed, a new entry will be created.

Private Attributes

- std::string [configFile](#)
File name of the config file.
- std::map< std::string,
[ConfigSector](#) > [value](#)
A container contains all [ConfigSector](#) instances.

5.8.1 Detailed Description

Manages config files (core class).

Note: The config is stored in the memory.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 [ConfigManager::ConfigManager](#) ()

Default Constructor for [ConfigManager](#).

This constructor does not sepecify the filename of the config. Hence, [SetFileName\(\)](#) should be called befor [Write\(\)](#) or [Read\(\)](#)

See also

[Write\(\)](#)
[Read\(\)](#)
[SetFileName\(\)](#)

5.8.2.2 ConfigManager::ConfigManager (const std::string & filename)

Initialising Constructor for [ConfigManager](#).

Parameters

<i>filename</i>	Default filename to Write() or Read()
-----------------	---

5.8.2.3 ConfigManager::~~ConfigManager ()

Delete all dynamic memory, if any.

5.8.3 Member Function Documentation

5.8.3.1 ConfigSector & ConfigManager::operator[] (const std::string & sector)

Fetch the sector of the config If the sector is not existed, a new entry will be created.

Parameters

<i>sector</i>	Sector name
---------------	-------------

Returns

A [ConfigSector](#) instance of the sector

5.8.3.2 const ConfigSector & ConfigManager::operator[] (const std::string & sector) const

Fetch the sector of the config in const form If the sector is not existed, a new entry will be created.

Parameters

<i>sector</i>	Sector name
---------------	-------------

Returns

A [ConfigSector](#) instance of the sector

Exceptions

ItemNotFound	If the sector does not exist
------------------------------	------------------------------

5.8.3.3 void ConfigManager::Read ()

Read in [ConfigManager](#) from file with default settings.

Exceptions

IOException	If filename is not specified or file does not exist or file corrupted.
-----------------------------	--

5.8.3.4 void ConfigManager::Read (const std::string & filename)

Read in [ConfigManager](#) from file.

Parameters

<i>filename</i>	The name of the file to read
-----------------	------------------------------

Exceptions

IOException	If file does not exist or file corrupted.
-----------------------------	---

5.8.3.5 void ConfigManager::SetFileName (const std::string & filename)

Externally set the file name of the config file.

Parameters

<i>filename</i>	The name of the file that will be used by Write() and Read() as default
-----------------	---

5.8.3.6 void ConfigManager::Write () const

Write out [ConfigManager](#) to file with default settings.

Exceptions

IOException	If filename is not specified or unable to write file.
-----------------------------	---

5.8.3.7 void ConfigManager::Write (const std::string & filename) const

Write out [ConfigManager](#) to file.

Parameters

<i>filename</i>	The name of the file to be writted
-----------------	------------------------------------

Exceptions

IOException	If unable to write file.
-----------------------------	--------------------------

5.8.4 Member Data Documentation

5.8.4.1 std::string nfrd::config::ConfigManager::configFile [private]

File name of the config file.

5.8.4.2 std::map<std::string, ConfigSector> nfrd::config::ConfigManager::value [private]

A container contains all [ConfigSector](#) instances.

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

- include/nfrd/[ConfigManager.h](#)

- src/[ConfigManager.cpp](#)

5.9 nfrd::config::ConfigSector Class Reference

A part of [ConfigManager](#) (as a container)

```
#include <ConfigManager.h>
```

Public Types

- typedef std::map< std::string, std::string >::iterator iterator
Iteraor.
- typedef std::map< std::string, std::string >::const_iterator const_iterator
Constant iterator.

Public Member Functions

- [ConfigSector](#) ()
Initialising Constructor for [ConfigSector](#).
- [~ConfigSector](#) ()
Delete all dynamic memory, if any.
- void [Write](#) (std::ostream &out) const
Write out [ConfigSector](#).
- void [Read](#) (std::istream &in)
Read in [ConfigSector](#).
- std::string & [operator\[\]](#) (const std::string &arg)
Fetch the content of the sector.
- const std::string & [operator\[\]](#) (const std::string &arg) const
Fetch the content of the sector.
- [iterator begin](#) ()
Returns an iterator referring to the first element in the [ConfigSector](#) container.
- [const_iterator begin](#) () const
Returns an constant iterator referring to the first element in the [ConfigSector](#) container.
- [iterator end](#) ()
Returns an iterator referring to the past-the-end element in the [ConfigSector](#) container.
- [const_iterator end](#) () const
Returns an constant iterator referring to the past-the-end element in the [ConfigSector](#) container.

Private Attributes

- std::string [name](#)
Sector name.
- std::map< std::string, std::string > [value](#)
a container to store all arguments with their values

5.9.1 Detailed Description

A part of [ConfigManager](#) (as a container)

5.9.2 Member Typedef Documentation

5.9.2.1 `typedef std::map<std::string, std::string>::const_iterator nfrd::config::ConfigSector::const_iterator`

Constant iterator.

5.9.2.2 `typedef std::map<std::string, std::string>::iterator nfrd::config::ConfigSector::iterator`

Iteraor.

5.9.3 Constructor & Destructor Documentation

5.9.3.1 `ConfigSector::ConfigSector ()`

Initialising Constructor for [ConfigSector](#).

5.9.3.2 `ConfigSector::~~ConfigSector ()`

Delete all dynamic memory, if any.

5.9.4 Member Function Documentation

5.9.4.1 `ConfigSector::iterator ConfigSector::begin ()`

Returns an iterator referring to the first element in the [ConfigSector](#) container.

Returns

an iterator referring to the first element in the container

5.9.4.2 `ConfigSector::const_iterator ConfigSector::begin () const`

Returns an constant iterator referring to the first element in the [ConfigSector](#) container.

Returns

an constant iterator referring to the first element in the container

5.9.4.3 `ConfigSector::iterator ConfigSector::end ()`

Returns an iterator referring to the past-the-end element in the [ConfigSector](#) container.

Returns

an iterator to the element past the end of the container

5.9.4.4 `ConfigSector::const_iterator ConfigSector::end () const`

Returns an constant iterator referring to the past-the-end element in the [ConfigSector](#) container.

Returns

an constant iterator to the element past the end of the container

5.9.4.5 `std::string & ConfigSector::operator[] (const std::string & arg)`

Fetch the content of the sector.

If the argument name is not existed, a new entry will be created.

Parameters

<i>arg</i>	Argument name
------------	---------------

Returns

Value of fetched argument

5.9.4.6 `const std::string & ConfigSector::operator[] (const std::string & arg) const`

Fetch the content of the sector.

Parameters

<i>arg</i>	Argument name
------------	---------------

Returns

Value of fetched argument

Exceptions

<i>ItemNotFound</i>	If the argument does not exist
-------------------------------------	--------------------------------

5.9.4.7 `void ConfigSector::Read (std::istream & in)`

Read in [ConfigSector](#).

Parameters

<i>in</i>	Read ConfigSector from this stream
-----------	--

Exceptions

<i>IOException</i>	If argument is not correctly formatted
------------------------------------	--

5.9.4.8 `void ConfigSector::Write (std::ostream & out) const`

Write out [ConfigSector](#).

Parameters

<i>out</i>	Write ConfigSector to this stream
------------	---

5.9.5 Member Data Documentation

5.9.5.1 `std::string nfrd::config::ConfigSector::name` [private]

Sector name.

5.9.5.2 `std::map<std::string, std::string> nfrd::config::ConfigSector::value` [private]

a container to store all arguments with their values

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

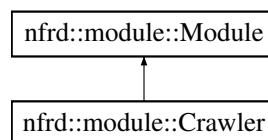
- [include/nfrd/ConfigManager.h](#)
- [src/ConfigManager.cpp](#)

5.10 `nfrd::module::Crawler` Class Reference

The main class representing the crawler module The responsibilities of this class:

```
#include <Crawler.h>
```

Inheritance diagram for `nfrd::module::Crawler`:



Public Member Functions

- [Crawler](#) (const [config::ConfigManager](#) &[config](#), const [log::LogManager](#) &[log](#))
Initialising Constructor for [Crawler](#).
- [~Crawler](#) ()
Delete all dynamic memory, if any.
- void [operator\(\)](#) ()
Called on module start, sets up and runs the queue.
- void [Stop](#) ()
[Module](#) stop, shuts down all activities safely.

Private Member Functions

- void [MainLoop](#) ()
The main loop for this thread, ensure that the queue items are kept up to date and that any new items are placed into the queue.
- void [RunMaintenanceTasks](#) ()
Run some maintenance tasks on the priority queue.
- void [StartThreads](#) ()
Setup and run all queue worker threads ([CrawlerThread](#))
- void [PersistQueue](#) ()
Persist the queue back to the database.
- void [InitialiseQueue](#) ()
Initialise the queue by pulling out a list of all items from the DB.

Private Attributes

- boost::thread_group [threads](#)
Group of worker threads.
- std::list< [CrawlerThread](#) * > [crawlers](#)
[CrawlerThread](#) classes, each one represents a worker thread.
- [FeedPriorityQueue](#) * [priorityQueue](#)
The priority queue used for delegating work.
- boost::mutex [healthMutex](#)
Mutex mainly used for the [healthThreadCondition](#).
- boost::condition_variable [healthThreadCondition](#)
Condition for running maintenance tasks.
- bool [isAlive](#)
Flag for whether the crawler thread is alive.
- const int [WAITTIME](#)
The time, in milliseconds to wait until health tasks are executed.

Additional Inherited Members

5.10.1 Detailed Description

The main class representing the crawler module The responsibilities of this class:

- Management of priority queue
- Management of worker threads
- Responsible for maintaining priority queue health

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Crawler::Crawler (const config::ConfigManager & config, const log::LogManager & log)

Initialising Constructor for [Crawler](#).

Parameters

<i>config</i>	The config manager to use
<i>log</i>	The log manager to use

5.10.2.2 Crawler::~Crawler ()

Delete all dynamic memory, if any.

5.10.3 Member Function Documentation

5.10.3.1 void Crawler::InitialiseQueue () [private]

Initialise the queue by pulling out a list of all items from the DB.

5.10.3.2 void Crawler::MainLoop () [private]

The main loop for this thread, ensure that the queue items are kept up to date and that any new items are placed into the queue.

5.10.3.3 void Crawler::operator()() [virtual]

Called on module start, sets up and runs the queue.

Implements [nfrd::module::Module](#).

5.10.3.4 void Crawler::PersistQueue () [private]

Persist the queue back to the database.

5.10.3.5 void Crawler::RunMaintenanceTasks () [private]

Run some maintenance tasks on the priority queue.

5.10.3.6 void Crawler::StartThreads () [private]

Setup and run all queue worker threads ([CrawlerThread](#))

5.10.3.7 void Crawler::Stop () [virtual]

[Module](#) stop, shuts down all activities safely.

Reimplemented from [nfrd::module::Module](#).

5.10.4 Member Data Documentation

5.10.4.1 std::list<CrawlerThread*> nfrd::module::Crawler::crawlers [private]

[CrawlerThread](#) classes, each one represents a worker thread.

5.10.4.2 boost::mutex nfrd::module::Crawler::healthMutex [mutable], [private]

Mutex mainly used for the healthThreadCondition.

5.10.4.3 boost::condition_variable nfrd::module::Crawler::healthThreadCondition [mutable], [private]

Condition for running maintenance tasks.

5.10.4.4 bool nfrd::module::Crawler::isAlive [private]

Flag for whether the crawler thread is alive.

5.10.4.5 FeedPriorityQueue* nfrd::module::Crawler::priorityQueue [private]

The priority queue used for delegating work.

5.10.4.6 `boost::thread_group nfrd::module::Crawler::threads` `[mutable], [private]`

Group of worker threads.

5.10.4.7 `const int nfrd::module::Crawler::WAITTIME` `[private]`

The time, in milliseconds to wait until health tasks are executed.

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

- `include/nfrd/Crawler.h`
- `src/Crawler.cpp`

5.11 nfrd::module::CrawlerThread Class Reference

A worker class representing a thread.

```
#include <CrawlerThread.h>
```

Public Member Functions

- `CrawlerThread` (const `config::ConfigManager` &`config`, const `log::LogManager` &`log`, int `id`, `FeedPriorityQueue` *`priorityqueue`)
Initialising Constructor for Crawler.
- `~CrawlerThread` ()
Free any dynamic memory.
- void `Initialise` ()
Initialise this crawler thread for processing.
- void `Stop` ()
Stop this crawler thread from processing.
- int `GetId` ()
Get the Id of this crawler thread.

Private Member Functions

- void `UpdateItem` (`QueueItem` *`item`, `nfdb::Feed` *`feed`)
Updated the item with details from the feed.
- void `Request` ()
Main loop, continuously request for more feeds.
- void `Crawl` (`nfdb::FeedController` *`feedDataController`, `nfdb::Feed` *`feed`)
Crawl the given feed.

Private Attributes

- `FeedPriorityQueue` * `priorityQueue`
The priority queue to request more work from.
- int `id`
The id of this crawler thread.
- const `config::ConfigManager` & `config`
A config to use to obtain configuration settings.
- const `log::LogManager` & `log`

A log to use for logging out errors.

- bool `isAlive`

Whether the crawler thread is alive or not.

5.11.1 Detailed Description

A worker class representing a thread.

This class will poll

the priority queue for more work and then execute the work.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 `CrawlerThread::CrawlerThread (const config::ConfigManager & config, const log::LogManager & log, int id, FeedPriorityQueue * priorityqueue)`

Initialising Constructor for `Crawler`.

Parameters

<i>config</i>	The config manager to use
<i>log</i>	The log manager to use
<i>crawler</i>	The crawler to use (priority queue)

5.11.2.2 `CrawlerThread::~~CrawlerThread ()`

Free any dynamic memory.

5.11.3 Member Function Documentation

5.11.3.1 `void CrawlerThread::Crawl (nfdb::FeedController * feedDataController, nfdb::Feed * feed) [private]`

Crawl the given feed.

5.11.3.2 `int CrawlerThread::GetId ()`

Get the Id of this crawler thread.

5.11.3.3 `void CrawlerThread::Initialise ()`

Initialise this crawler thread for processing.

5.11.3.4 `void CrawlerThread::Request () [private]`

Main loop, continuously request for more feeds.

5.11.3.5 `void CrawlerThread::Stop ()`

Stop this crawler thread from processing.

5.11.3.6 `void CrawlerThread::UpdateItem (QueueItem * item, nfdb::Feed * feed) [private]`

Updated the item with details from the feed.

Parameters

<i>item</i>	Item to update
<i>feed</i>	Feed to use when updating item

5.11.4 Member Data Documentation

5.11.4.1 `const config::ConfigManager& nfrd::module::CrawlerThread::config [private]`

A config to use to obtain configuration settings.

5.11.4.2 `int nfrd::module::CrawlerThread::id [private]`

The id of this crawler thread.

5.11.4.3 `bool nfrd::module::CrawlerThread::isAlive [private]`

Whether the crawler thread is alive or not.

5.11.4.4 `const log::LogManager& nfrd::module::CrawlerThread::log [private]`

A log to use for logging out errors.

5.11.4.5 `FeedPriorityQueue* nfrd::module::CrawlerThread::priorityQueue [private]`

The priority queue to request more work from.

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

- [include/nfrd/CrawlerThread.h](#)
- [src/CrawlerThread.cpp](#)

5.12 nfrd::misc::DateTime Class Reference

A class to store date and time.

```
#include <DateTime.h>
```

Public Member Functions

- [DateTime \(\)](#)
Default Constructor for Datetime.
- [DateTime \(const DateTime &rhs\)](#)
Initialising Constructor for DateTime.
- [~DateTime \(\)](#)
Delete all dynamic memory, if any.
- `std::string ExportToMySQL \(\) const`
Export a string with the format for MySQL datetime type.

- void [ImportFromMySQL](#) (const std::string &source)
Parse date time from a string from MySQL.
- void [ParseFromString](#) (const std::string &source)
Parse date time from a string by auto detecting standard.
- void [ParseFromRFC822](#) (const std::string &source)
Parse date time from a string using the RFC822 standard.
- void [ParseFromRFC3339](#) (const std::string &source)
Parse date time from a string using the RFC3339 standard.
- int [GetSecond](#) () const
Return the second part of the [DateTime](#).
- int [GetMinute](#) () const
Return the minute part of the [DateTime](#).
- int [GetHour](#) () const
Return the hour part of the [DateTime](#).
- int [GetDay](#) () const
Return the day part of the [DateTime](#).
- int [GetMonth](#) () const
Return the month part of the [DateTime](#).
- int [GetYear](#) () const
Return the year part of the [DateTime](#).
- void [SetSecond](#) (int sec)
Set the second part of the [DateTime](#).
- void [SetMinute](#) (int min)
Set the minute part of the [DateTime](#).
- void [SetHour](#) (int h)
Set the hour part of the [DateTime](#).
- void [SetDay](#) (int d)
Set the day part of the [DateTime](#).
- void [SetMonth](#) (int m)
Set the month part of the [DateTime](#).
- void [SetYear](#) (int y)
Set the year part of the [DateTime](#).
- void [Set](#) (int y, int n, int d, int h, int m, int s)
Set up all data in the [DateTime](#).
- void [SetTimeOffset](#) (int h, int m=0, int s=0)
Add time to the current [DateTime](#).
- void [SetDateOffset](#) (int d, int m=0, int y=0)
Add date to the current [DateTime](#).
- bool [operator<](#) (const [DateTime](#) &rhs) const
Compare two [DateTime](#) instances whether the current [DateTime](#) is before the specified [DateTime](#).
- bool [operator>](#) (const [DateTime](#) &rhs) const
Compare two [DateTime](#) instances whether the current [DateTime](#) is after the specified [DateTime](#).
- bool [operator==](#) (const [DateTime](#) &rhs) const
Compare two [DateTime](#) instances whether the current [DateTime](#) is equal to the specified [DateTime](#).
- bool [operator!=](#) (const [DateTime](#) &rhs) const
Compare two [DateTime](#) instances whether the current [DateTime](#) is not equal to the specified [DateTime](#).

Private Member Functions

- void [JustifyTime](#) ()
Rounding the Time part of the [DateTime](#).
- void [JustifyDate](#) ()
Rounding the Date part of the [DateTime](#).

Private Attributes

- int [second](#)
Second part of [DateTime](#).
- int [minute](#)
Minute part of [DateTime](#).
- int [hour](#)
Hour part of [DateTime](#).
- int [day](#)
Day part of [DateTime](#).
- int [month](#)
Month part of [DateTime](#).
- int [year](#)
Year part of [DateTime](#).

5.12.1 Detailed Description

A class to store date and time.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 [DateTime::DateTime](#) ()

Default Constructor for Datatime.

The time will be initialised to 01/01/0000

5.12.2.2 [DateTime::DateTime](#) (const [DateTime](#) & *rhs*)

Initialising Constructor for [DateTime](#).

Parameters

<i>rhs</i>	Source instance to copy
------------	-------------------------

5.12.2.3 [DateTime::~~DateTime](#) ()

Delete all dynamic memory, if any.

5.12.3 Member Function Documentation

5.12.3.1 [string DateTime::ExportToMySQL](#) () const

Export a string with the format for MySQL datetime type.

e.g. 2000-01-01 00:00:00 or 2000-1-1 0:0:0 both are OK.

This format can be parsed by mysql in a single SQL statement.

Returns

Time string suitable for mysql

5.12.3.2 int DateTime::GetDay () const

Return the day part of the [DateTime](#).

Returns

day part

5.12.3.3 int DateTime::GetHour () const

Return the hour part of the [DateTime](#).

Returns

hour part

5.12.3.4 int DateTime::GetMinute () const

Return the minute part of the [DateTime](#).

Returns

minute part

5.12.3.5 int DateTime::GetMonth () const

Return the month part of the [DateTime](#).

Returns

month part

5.12.3.6 int DateTime::GetSecond () const

Return the second part of the [DateTime](#).

Returns

second part

5.12.3.7 int DateTime::GetYear () const

Return the year part of the [DateTime](#).

Returns

year part

5.12.3.8 void DateTime::ImportFromMySQL (const std::string & source)

Parse date time from a string from MySQL.

Sample: 2000-01-01 00:00:00

Parameters

<i>source</i>	original date time string
---------------	---------------------------

Exceptions

<i>std::domain_error</i>	if
--------------------------	----

Parameters

<i>source</i>	is corrupted
---------------	--------------

5.12.3.9 void DateTime::JustifyDate () [private]

Rounding the Date part of the [DateTime](#).

5.12.3.10 void DateTime::JustifyTime () [private]

Rounding the Time part of the [DateTime](#).

If necessary, it will call JusityDate() via [SetDateOffset\(\)](#) for rounding the Date part.

5.12.3.11 bool DateTime::operator!= (const DateTime & rhs) const

Compare two [DateTime](#) instances whether the current [DateTime](#) is not equal to the specified [DateTime](#).

Parameters

<i>rhs</i>	DateTime instance to be compared
------------	--

Returns

true If the current [DateTime](#) is not equal to the specified [DateTime](#).

5.12.3.12 bool DateTime::operator< (const DateTime & rhs) const

Compare two [DateTime](#) instances whether the current [DateTime](#) is before the specified [DateTime](#).

Parameters

<i>rhs</i>	DateTime instance to be compared
------------	--

Returns

true If the current [DateTime](#) is before the specified [DateTime](#).

5.12.3.13 `bool DateTime::operator==(const DateTime & rhs) const`

Compare two `DateTime` instances whether the current `DateTime` is equal to the specified `DateTime`.

Parameters

<i>rhs</i>	<code>DateTime</code> instance to be compared
------------	---

Returns

true If the current `DateTime` is equal to the specified `DateTime`.

5.12.3.14 `bool DateTime::operator> (const DateTime & rhs) const`

Compare two `DateTime` instances whether the current `DateTime` is after the specified `DateTime`.

Parameters

<i>rhs</i>	<code>DateTime</code> instance to be compared
------------	---

Returns

true If the current `DateTime` is after the specified `DateTime`.

5.12.3.15 `void DateTime::ParseFromRFC3339 (const std::string & source)`

Parse date time from a string using the RFC3339 standard.

Note: The date time will be forced coverneted to UTC+0000. Sample: 1937-01-01T12:00:27.87+00:20

Parameters

<i>source</i>	original date time string
---------------	---------------------------

Exceptions

<code>std::domain_error</code>	if
--------------------------------	----

Parameters

<i>source</i>	is corrupted
---------------	--------------

See also

<http://www.ietf.org/rfc/rfc3339.txt>

5.12.3.16 `void DateTime::ParseFromRFC822 (const std::string & source)`

Parse date time from a string using the RFC822 standard.

Note: The date time will be forced coverneted to UTC+0000. Sample: Sat, 07 Sep 2002 09:42:31 +0000

Parameters

<i>source</i>	original date time string
---------------	---------------------------

Exceptions

<code>std::domain_error</code>	if
--------------------------------	----

Parameters

<code>source</code>	is corrupted
---------------------	--------------

See also

<http://www.ietf.org/rfc/rfc822.txt>

5.12.3.17 void DateTime::ParseFromString (const std::string & *source*)

Parse date time from a string by auto detecting standard.

Note: The date time will be forced converted to UTC+0000.

Parameters

<code>source</code>	original date time string
---------------------	---------------------------

Exceptions

<code>std::domain_error</code>	if
--------------------------------	----

Parameters

<code>source</code>	is corrupted
---------------------	--------------

5.12.3.18 void DateTime::Set (int *y*, int *n*, int *d*, int *h*, int *m*, int *s*)

Set up all data in the [DateTime](#).

This function will DO the roundings by calling [JustifyTime\(\)](#) and [JustifyDate\(\)](#). Hence, There is no domain restricted.

Parameters

<code>y</code>	year
<code>n</code>	month
<code>d</code>	day
<code>h</code>	hour
<code>m</code>	minute
<code>s</code>	second

5.12.3.19 void DateTime::SetDateOffset (int *d*, int *m* = 0, int *y* = 0)

Add date to the current [DateTime](#).

e.g. For [DateTime](#) 25/12/2011, SetDateOffset(10,1,0) will result in 04/02/2012.

The function will do roundings automatically.

Note: Negative numbers are accepted as subtraction.

Parameters

<i>d</i>	day
<i>m</i>	month
<i>y</i>	yeaar

5.12.3.20 void DateTime::SetDay (int *d*)

Set the day part of the [DateTime](#).

Note: The function does NOT do the roundings.

The domain range of this attribute is as following.

Month	Range
1, 3, 5, 7, 8, 10, 12	1-31
4, 6, 9, 11	1-30
2 (leap year)	1-29
2 (not leap year)	1-28

If the day to be set is not sure for the month or year, use [Set\(\)](#) or [SetDateOffset\(\)](#) to safely set the day.

Parameters

<i>d</i>	day part
----------	----------

Exceptions

<i>std::domain_error</i>	for wrong domain
--------------------------	------------------

See also

[Set\(\)](#)
[SetDateOffset\(\)](#)

5.12.3.21 void DateTime::SetHour (int *h*)

Set the hour part of the [DateTime](#).

Note: The function does NOT do the roundings. The domain range of this attribute is 0-23.

Parameters

<i>h</i>	hour part
----------	-----------

Exceptions

<i>std::domain_error</i>	for wrong domain
--------------------------	------------------

5.12.3.22 void DateTime::SetMinute (int *min*)

Set the minute part of the [DateTime](#).

Note: The function does NOT do the roundings. The domain range of this attribute is 0-59.

Parameters

<i>min</i>	minute part
------------	-------------

Exceptions

<i>std::domain_error</i>	for wrong domain
--------------------------	------------------

5.12.3.23 void DateTime::SetMonth (int *m*)

Set the month part of the [DateTime](#).

Note: The function does NOT do the roundings. The domain range of this attribute is 1-12. The day part of the Datetime will automatically rounded. e.g. 30/03/2012 will become 01/03/2012 if month is set to 2.

Parameters

<i>m</i>	month part
----------	------------

Exceptions

<i>std::domain_error</i>	for wrong domain
--------------------------	------------------

5.12.3.24 void DateTime::SetSecond (int *sec*)

Set the second part of the [DateTime](#).

Note: The function does NOT do the roundings. The domain range of this attribute is 0-59.

Parameters

<i>sec</i>	second part
------------	-------------

Exceptions

<i>std::domain_error</i>	for wrong domain
--------------------------	------------------

5.12.3.25 void DateTime::SetTimeOffset (int *h*, int *m* = 0, int *s* = 0)

Add time to the current [DateTime](#).

e.g. For [DateTime](#) 12:30:00, SetTimeOffset(2,45,30) will result in 15:15:30.

The function will do roundings automatically.

Note: Negative numbers are accepted as subtraction.

Parameters

<i>h</i>	hour
<i>m</i>	minute
<i>s</i>	second

5.12.3.26 void DateTime::SetYear (int *y*)

Set the year part of the [DateTime](#).

The year could be negative which means BC.

Parameters

<code>y</code>	year part
----------------	-----------

5.12.4 Member Data Documentation

5.12.4.1 `int nfrd::misc::DateTime::day` `[private]`

Day part of [DateTime](#).

5.12.4.2 `int nfrd::misc::DateTime::hour` `[private]`

Hour part of [DateTime](#).

5.12.4.3 `int nfrd::misc::DateTime::minute` `[private]`

Minute part of [DateTime](#).

5.12.4.4 `int nfrd::misc::DateTime::month` `[private]`

Month part of [DateTime](#).

5.12.4.5 `int nfrd::misc::DateTime::second` `[private]`

Second part of [DateTime](#).

5.12.4.6 `int nfrd::misc::DateTime::year` `[private]`

Year part of [DateTime](#).

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

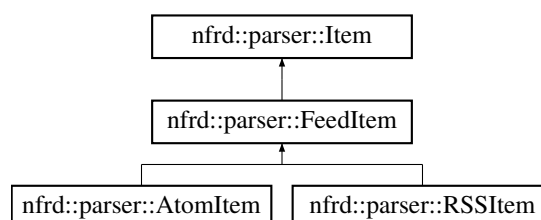
- `include/nfrd/DateTime.h`
- `src/DateTime.cpp`

5.13 nfrd::parser::FeedItem Class Reference

A class to store details of an item obtained by the [FeedParser](#).

```
#include <FeedParser.h>
```

Inheritance diagram for `nfrd::parser::FeedItem`:



Public Member Functions

- [FeedItem](#) ()
Initialising Constructor for [FeedItem](#).
- virtual [~FeedItem](#) ()
Delete all dynamic memory, if any.
- const std::string & [GetGeoLocation](#) () const
Get the geographical location of the item.
- const std::list< [Image](#) > & [GetImageList](#) () const
Get the image list of the item.
- virtual void [SetContent](#) (const std::string &source)=0
Set the content of the item.
- void [SetGeoLocation](#) (const std::string &source)
Set the geographical location of the item.
- void [SetFullContent](#) (bool hasFullContent)
Set whether the content is the full version.
- void [AddImage](#) (const std::string &source)
Add an URL of an image to the image list, auto generating thumbnail.
- void [RemoveImage](#) (const std::string &source)
Remove an URL of an image from the image list.
- void [ClearImage](#) ()
Clear the image list.
- bool [HasGeoLocation](#) () const
Test the item has geographical location or not.
- bool [HasImageList](#) () const
Test the item has image list or not.
- bool [HasFullContent](#) () const
Whether the content is the full version or not.

Private Attributes

- std::string * [geo_location](#)
Geographical location of the item.
- std::list< [Image](#) > * [image_list](#)
Images in the item Optional in RSS.
- bool [full_content](#)
Whether the content is the full version.

Additional Inherited Members

5.13.1 Detailed Description

A class to store details of an item obtained by the [FeedParser](#).

See also

[Item](#)

5.13.2 Constructor & Destructor Documentation

5.13.2.1 `FeedItem::FeedItem ()`

Initialising Constructor for [FeedItem](#).

Initialise everything to zero/null.

5.13.2.2 `FeedItem::~~FeedItem ()` [virtual]

Delete all dynamic memory, if any.

5.13.3 Member Function Documentation

5.13.3.1 `void FeedItem::AddImage (const std::string & source)`

Add an URL of an image to the image list, auto generating thumbnail.

Parameters

<i>source</i>	url of an image to be added
---------------	-----------------------------

5.13.3.2 `void FeedItem::ClearImage ()`

Clear the image list.

Note: This operation actually purge the image list.

5.13.3.3 `const std::string & FeedItem::GetGeoLocation () const` [virtual]

Get the geographical location of the item.

Returns

geographical location

Exceptions

HasNoValue	if the item has no geographical location or if this function is not overridden
----------------------------	--

Reimplemented from [nfrd::parser::Item](#).

5.13.3.4 `const list< Item::Image > & FeedItem::GetImageList () const` [virtual]

Get the image list of the item.

The relationship between the item and the image is one-to-many.

Returns

image list

Exceptions

HasNoValue	if the item has no image list or if this function is not overridden
----------------------------	---

Reimplemented from [nfrd::parser::Item](#).

5.13.3.5 bool FeedItem::HasFullContent () const

Whether the content is the full version or not.

Returns

true if the content of the item is the full version.

5.13.3.6 bool FeedItem::HasGeoLocation () const [virtual]

Test the item has geographical location or not.

Returns

true If the item has geographical location
false If the item has no geographical location or this function is not overridden

Reimplemented from [nfrd::parser::Item](#).

5.13.3.7 bool FeedItem::HasImageList () const [virtual]

Test the item has image list or not.

Returns

true If the item has image list
false If the item has no image list or this function is not overridden

Reimplemented from [nfrd::parser::Item](#).

5.13.3.8 void FeedItem::RemoveImage (const std::string & source)

Remove an URL of an image from the image list.

Parameters

<i>source</i>	url of an image to be removed
---------------	-------------------------------

5.13.3.9 virtual void nfrd::parser::FeedItem::SetContent (const std::string & source) [pure virtual]

Set the content of the item.

Parameters

<i>source</i>	content of the item
---------------	---------------------

Implemented in [nfrd::parser::AtomItem](#), and [nfrd::parser::RSSItem](#).

5.13.3.10 void FeedItem::SetFullContent (bool hasFullContent)

Set whether the content is the full version.

Parameters

<i>hasFullContent</i>	whether the content is the full version
-----------------------	---

5.13.3.11 void FeedItem::SetGeoLocation (const std::string & source)

Set the geographical location of the item.

Parameters

<i>source</i>	geographical location of the item
---------------	-----------------------------------

See also

[FeedItem](#)

5.13.4 Member Data Documentation

5.13.4.1 bool nfrd::parser::FeedItem::full_content [private]

Whether the content is the full version.

5.13.4.2 std::string* nfrd::parser::FeedItem::geo_location [private]

Geographical location of the item.

5.13.4.3 std::list<Image>* nfrd::parser::FeedItem::image_list [private]

Images in the item Optional in RSS.

Original tag in RSS: media:content

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

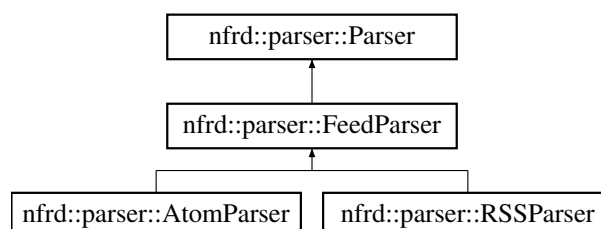
- include/nfrd/[FeedParser.h](#)
- src/[FeedParser.cpp](#)

5.14 nfrd::parser::FeedParser Class Reference

A parser to parse web feeds.

```
#include <FeedParser.h>
```

Inheritance diagram for nfrd::parser::FeedParser:



Public Member Functions

- [FeedParser](#) ()
Initialising Constructor for [RSSParser](#).
- virtual [~FeedParser](#) ()
Delete all dynamic memory, if any.
- virtual void [ReadURL](#) (const std::string &url)
Read resouce from an URL and parse into a list of [Item](#).
- const std::list< [Item](#) * > & [GetItemList](#) () const
Get the item list of parsed feed.
- virtual void [ReadDom](#) (const rapidxml::xml_document<> &doc)=0
Parse feed from a dom tree (xml document) into a list of [Item](#).

Protected Attributes

- std::string * url
URL of the source.
- std::list< [Item](#) * > item
Items of the RSS feed Required in RSS.

5.14.1 Detailed Description

A parser to parse web feeds.

See also

[Parser](#)

5.14.2 Constructor & Destructor Documentation

5.14.2.1 [FeedParser::FeedParser](#) ()

Initialising Constructor for [RSSParser](#).

5.14.2.2 [FeedParser::~~FeedParser](#) () [virtual]

Delete all dynamic memory, if any.

5.14.3 Member Function Documentation

5.14.3.1 const list< [Item](#) * > & [FeedParser::GetItemList](#) () const [virtual]

Get the item list of parsed feed.

Returns

item list

Exceptions

HasNoValue	if the feed has no item list
----------------------------	------------------------------

Reimplemented from [nfrd::parser::Parser](#).

5.14.3.2 `virtual void nfrd::parser::FeedParser::ReadDom (const rapidxml::xml_document<> & doc)` [pure virtual]

Parse feed from a dom tree (xml document) into a list of [Item](#).

Parameters

<i>doc</i>	parsed xml document of the feed resource
------------	--

Exceptions

InvalidSource	if the dom or the feed resource is invalid
-------------------------------	--

Implemented in [nfrd::parser::AtomParser](#), and [nfrd::parser::RSSParser](#).

5.14.3.3 `void FeedParser::ReadURL (const std::string & url)` [virtual]

Read resource from an URL and parse into a list of [Item](#).

Parameters

<i>url</i>	URL address of the feed resource
------------	----------------------------------

Exceptions

InvalidSource	if the url or the feed resource is invalid
-------------------------------	--

Implements [nfrd::parser::Parser](#).

5.14.4 Member Data Documentation

5.14.4.1 `std::list<Item*> nfrd::parser::FeedParser::item` [protected]

Items of the RSS feed Required in RSS.

Original tag in RSS: item

5.14.4.2 `std::string* nfrd::parser::FeedParser::url` [protected]

URL of the source.

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

- include/nfrd/[FeedParser.h](#)
- src/[FeedParser.cpp](#)

5.15 nfrd::module::FeedPriorityQueue Class Reference

Implements a queueing/threading model.

```
#include <FeedPriorityQueue.h>
```

Public Member Functions

- [FeedPriorityQueue](#) (const [nfrd::config::ConfigManager](#) &config, const [nfrd::log::LogManager](#) &log, int reserve=0)
Initialising Constructor for [FeedPriorityQueue](#).
- [~FeedPriorityQueue](#) ()
Delete all dynamic memory, if any.
- void [PopFeed](#) ([QueueItem](#) *&item)
Removes a feed from the prioritised queue for processing.
- void [PushFeed](#) ([QueueItem](#) *item)
Adds a feed on to the queue.
- void [Start](#) ()
Start the priority queue/thread.
- void [Stop](#) ()
Stop the priority queue/thread
Only call priority queue stop when you are 100% sure that nothing else is using the queue.
- boost::thread * [GetThread](#) ()
Get the thread that this priority queue is running under.
- std::vector< [QueueItem](#) * > [GetAllItems](#) ()
Get a vector of all queue items.
- void [SetNumberOfUsersInSystem](#) (int numberOfUsersInSystem)
Set the number of users in the system.

Private Member Functions

- void [MainLoop](#) ()
The main loop for the main queueing thread of the crawler waits for work executes [IterateOnce](#) when it has some.
- void [IterateOnce](#) ()
Processes the priority queue (aka heap, aka vector) and the incoming and outgoing queues to ensure that they are all up to date.
- void [StartThreads](#) ()
Setup and run all queue worker threads ([CrawlerThread](#))
- void [CleanupQueue](#) ()
When this module is stopped, cleanup queue is called to persist items back to the database.

Private Attributes

- boost::mutex [incomingMutex](#)
Mutex to synchronise access to the incoming queue.
- boost::mutex [outgoingMutex](#)
Mutex to synchronise access to the outgoing queue.
- boost::mutex [updatedMutex](#)
Mutex to synchronise access to the outgoingHasChanged and incomingHasChanged variables.
- boost::mutex [heapMutex](#)
Mutex to synchronise access to the internal heap.
- boost::condition_variable [outgoingQueueCondition](#)
Condition for waiting on more items to be available in the outgoing queue for the worker threads (notified by the main queueing thread)

- `boost::condition_variable` [queueUpdateCondition](#)
Condition for the main queueing thread to wait on PushFeed and PopFeed executions (notified by the worker threads)
- `std::vector< QueueItem * >` [itemQueue](#)
The main priority queue (heap) only the main queueing thread is allowed to touch this!
- `std::queue< QueueItem * >` [incomingQueue](#)
A buffer queue for any incoming (PushFeed) items.
- `std::vector< QueueItem * >` [tempOutgoingQueue](#)
A temporary array used for performance reasons by the main queueing thread.
- `std::queue< QueueItem * >` [outgoingQueue](#)
A buffer queue for any outgoing (PopFeed) items.
- `const nfrd::config::ConfigManager &` [config](#)
Configuration access.
- `const nfrd::log::LogManager &` [log](#)
Access to logging system.
- `int` [itemsPopped](#)
Used internally, the number of items which have been popped from the heap and put on the outgoing queue since the heap was last prioritised/sorted.
- `bool` [outgoingHasChanged](#)
Used to help synchronise the main queueing thread so that it only iterates when there is work to do.
- `bool` [incomingHasChanged](#)
Used to help synchronise the main queueing thread so that it only iterates when there is work to do.
- `bool` [isAlive](#)
Used to keep the main queueing thread alive.
- `const int` [MAXPOP](#)
The maximum number of items to pop into the outgoing queue before requiring a prioritise/sort on the heap.
- `int` [numberOfUsersInSystem](#)
The number of users in the system, used to help priority calculations.
- `boost::thread` [priorityQueueThread](#)
The main queueing thread dedicated to this object.

5.15.1 Detailed Description

Implements a queueing/threading model.

The model was designed to sort and prioritise the feeds in the background while providing a fast way of delegating work to background feeds. So to summarise the responsibilities of this class:

- Prioritising all feeds for processing
- Delegating work to threads (in a consumer/producer type model)

5.15.2 Constructor & Destructor Documentation

5.15.2.1 `FeedPriorityQueue::FeedPriorityQueue (const nfrd::config::ConfigManager & config, const nfrd::log::LogManager & log, int reserve = 0)`

Initialising Constructor for [FeedPriorityQueue](#).

Parameters

<i>config</i>	The config manager to use
<i>log</i>	The log manager to use

5.15.2.2 FeedPriorityQueue::~~FeedPriorityQueue ()

Delete all dynamic memory, if any.

5.15.3 Member Function Documentation

5.15.3.1 void FeedPriorityQueue::CleanupQueue () [private]

When this module is stopped, cleanup queue is called to persist items back to the database.

5.15.3.2 vector< QueueItem * > FeedPriorityQueue::GetAllItems ()

Get a vector of all queue items.

Note the queue should be stopped before calling this function

5.15.3.3 boost::thread * FeedPriorityQueue::GetThread ()

Get the thread that this priority queue is running under.

5.15.3.4 void FeedPriorityQueue::IterateOnce () [private]

Processes the priority queue (aka heap, aka vector) and the incoming and outgoing queues to ensure that they are all up to date.

Can be broken down into these operations:

- Process incoming queue
- Re-calculate feed priority and re-sort the queue
- Process outgoing queue

5.15.3.5 void FeedPriorityQueue::MainLoop () [private]

The main loop for the main queueing thread of the crawler waits for work executes IterateOnce when it has some.

"Work" is classified as:

- PopFeed is executed
- PushFeed is executed

5.15.3.6 void FeedPriorityQueue::PopFeed (QueueItem *& item)

Removes a feed from the prioritised queue for processing.

Parameters

<i>item</i>	Pop an item off the heap and populate item
-------------	--

5.15.3.7 void FeedPriorityQueue::PushFeed (QueueItem * item)

Adds a feed on to the queue.

Parameters

<i>item</i>	Put the given item onto the heap
-------------	----------------------------------

5.15.3.8 void FeedPriorityQueue::SetNumberOfUsersInSystem (int numberOfUsersInSystem)

Set the number of users in the system.

5.15.3.9 void FeedPriorityQueue::Start ()

Start the priority queue/thread.

5.15.3.10 void nfrd::module::FeedPriorityQueue::StartThreads () [private]

Setup and run all queue worker threads ([CrawlerThread](#))

5.15.3.11 void FeedPriorityQueue::Stop ()

Stop the priority queue/thread

Only call priority queue stop when you are 100% sure that nothing else is using the queue.

5.15.4 Member Data Documentation

5.15.4.1 const nfrd::config::ConfigManager& nfrd::module::FeedPriorityQueue::config [private]

Configuration access.

5.15.4.2 boost::mutex nfrd::module::FeedPriorityQueue::heapMutex [mutable], [private]

Mutex to synchronise access to the internal heap.

5.15.4.3 bool nfrd::module::FeedPriorityQueue::incomingHasChanged [private]

Used to help synchronise the main queueing thread so that it only iterates when there is work to do.

Updated by PushFeed

5.15.4.4 `boost::mutex nfrd::module::FeedPriorityQueue::incomingMutex` [mutable], [private]

Mutex to synchronise access to the incoming queue.

5.15.4.5 `std::queue<QueueItem*> nfrd::module::FeedPriorityQueue::incomingQueue` [private]

A buffer queue for any incoming (PushFeed) items.

5.15.4.6 `bool nfrd::module::FeedPriorityQueue::isAlive` [private]

Used to keep the main queueing thread alive.

Cannot use the module's "status" as this is being used by another section of the code, if both used the same variable/value then the main queueing thread would not safely stop.

5.15.4.7 `std::vector<QueueItem*> nfrd::module::FeedPriorityQueue::itemQueue` [private]

The main priority queue (heap) only the main queueing thread is allowed to touch this!

5.15.4.8 `int nfrd::module::FeedPriorityQueue::itemsPopped` [private]

Used internally, the number of items which have been popped from the heap and put on the outgoing queue since the heap was last prioritised/sorted.

5.15.4.9 `const nfrd::log::LogManager& nfrd::module::FeedPriorityQueue::log` [private]

Access to logging system.

5.15.4.10 `const int nfrd::module::FeedPriorityQueue::MAXPOP` [private]

The maximum number of items to pop into the outgoing queue before requiring a prioritise/sort on the heap.

Also represents the maximum number of items allowed in the outgoing queue at once. Changing this number to a higher number will be more efficient but will make the overall prioritising algorithm less effective. Making it smaller will in turn, decrease performance. This is configurable in the config, but changes to this are not currently supported during module runtime.

5.15.4.11 `int nfrd::module::FeedPriorityQueue::numberOfUsersInSystem` [private]

The number of users in the system, used to help priority calculations.

5.15.4.12 `bool nfrd::module::FeedPriorityQueue::outgoingHasChanged` [private]

Used to help synchronise the main queueing thread so that it only iterates when there is work to do.

Updated by PopFeed

5.15.4.13 `boost::mutex nfrd::module::FeedPriorityQueue::outgoingMutex` [mutable], [private]

Mutex to synchronise access to the outgoing queue.

5.15.4.14 `std::queue<QueueItem*> nfrd::module::FeedPriorityQueue::outgoingQueue` [private]

A buffer queue for any outgoing (PopFeed) items.

5.15.4.15 `boost::condition_variable nfrd::module::FeedPriorityQueue::outgoingQueueCondition` [mutable], [private]

Condition for waiting on more items to be available in the outgoing queue for the worker threads (notified by the main queueing thread)

5.15.4.16 `boost::thread nfrd::module::FeedPriorityQueue::priorityQueueThread` [private]

The main queueing thread dedicated to this object.

5.15.4.17 `boost::condition_variable nfrd::module::FeedPriorityQueue::queueUpdateCondition` [mutable], [private]

Condition for the main queueing thread to wait on PushFeed and PopFeed executions (notified by the worker threads)

5.15.4.18 `std::vector<QueueItem*> nfrd::module::FeedPriorityQueue::tempOutgoingQueue` [private]

A temporary array used for performance reasons by the main queueing thread.

5.15.4.19 `boost::mutex nfrd::module::FeedPriorityQueue::updatedMutex` [mutable], [private]

Mutex to synchronise access to the outgoingHasChanged and incomingHasChanged variables.

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

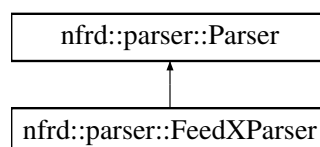
- [include/nfrd/FeedPriorityQueue.h](#)
- [src/FeedPriorityQueue.cpp](#)

5.16 nfrd::parser::FeedXParser Class Reference

A parser to parse web feeds with Patch and Match feature.

```
#include <FeedXParser.h>
```

Inheritance diagram for `nfrd::parser::FeedXParser`:



Classes

- struct [IteratorPair](#)

A iterator pair structure used in [Construct\(\)](#) to store iterators for parent nodes.

Public Member Functions

- [FeedXParser](#) ()
Initialising Constructor for [FeedParser](#).
- [~FeedXParser](#) ()
Delete all dynamic memory, if any.
- void [ReadURL](#) (const std::string &url)
Read resource from an URL and parse into a list of [Item](#).
- const [misc::DateTime](#) & [GetLastBuildDate](#) () const
Get the last build date of the feed resource.
- const std::list< [Item](#) * > & [GetItemList](#) () const
Get the item list of parsed feed.

Static Private Member Functions

- static void [PatchAndMatch](#) ([FeedItem](#) *item)
Patch an rss item using Patch and Match (Match and Patch) algorithm.
- static void [Construct](#) (const [IteratorPair](#) &iterator_pair, std::string &content)
Construct html source code from a html tree using all child nodes of a root node.
- static std::string::size_type [TextSize](#) (const tree< htmlcxx::HTML::Node >::iterator &iterator)
Get the total text size of a html tree, counting all child nodes of a root node.
- static std::string [GetLongestText](#) (const std::string &source)
Get the longest text from a html source code.
- static std::string [TrimEmptyTag](#) (const std::string &source)
Trim all empty a or div tags in the source.
- static void [ExtractImages](#) ([FeedItem](#) *item, const [IteratorPair](#) &iterator_pair)
Extract images from the given tree and add to the item.
- static void [ExtractGeoLocation](#) ([FeedItem](#) *item, const [IteratorPair](#) &iterator_pair)
Extract geographical location from the given tree and add to the item.
- static void [RefineFeed](#) ([FeedItem](#) *item)
Refine a feed by stripping tags and extracting images.
- static tree
 < htmlcxx::HTML::Node >
 ::iterator [AdvancedMatchLine](#) (const tree< htmlcxx::HTML::Node > &dom, std::string &match_string)
 Find the matched content block in a dom tree This method will modified match_string to the longest line in match_string.
- static tree
 < htmlcxx::HTML::Node >
 ::iterator [AdvancedMatchDivide](#) (const tree< htmlcxx::HTML::Node > &dom, std::string &match_string)
 Find the matched content block in a dom tree This method will modified match_string to the mid part of match_string.

Private Attributes

- [FeedParser](#) * [parser](#)
Feed parser to parse the actual feed.

Static Private Attributes

- static std::set< std::string > [allowed_tags](#)
The tags that will not be changed.
- static std::set< std::string > [trimmed_tags](#)
The tage that only text part is kept.

5.16.1 Detailed Description

A parser to parse web feeds with Patch and Match feature.

This parser will automatically identify feed type (RSS or Atom).

See also

[FeedParser](#)

5.16.2 Constructor & Destructor Documentation

5.16.2.1 FeedXPather::FeedXPather ()

Initialising Constructor for [FeedParser](#).

5.16.2.2 FeedXPather::~~FeedXPather ()

Delete all dynamic memory, if any.

5.16.3 Member Function Documentation

5.16.3.1 `tree< htmlcxx::HTML::Node >::iterator FeedXPather::AdvancedMatchDivide (const tree< htmlcxx::HTML::Node > & dom, std::string & match_string) [static], [private]`

Find the matched content block in a dom tree This method will modified *match_string* to the mid part of *match_string*.

Parameters

<i>dom</i>	source dom tree
<i>match_string</i>	the string to test the dom tree

Returns

the lowest level of node which match the string
dom.end() if not matched.

5.16.3.2 `tree< htmlcxx::HTML::Node >::iterator FeedXPather::AdvancedMatchLine (const tree< htmlcxx::HTML::Node > & dom, std::string & match_string) [static], [private]`

Find the matched content block in a dom tree This method will modified *match_string* to the longest line in *match_string*.

Parameters

<i>dom</i>	source dom tree
<i>match_string</i>	the string to test the dom tree

Returns

the lowest level of node which match the string
dom.end() if not matched.

5.16.3.3 void FeedXParser::Construct (const IteratorPair & *iterator_pair*, std::string & *content*) [static],
[private]

Construct html source code from a html tree using all child nodes of a root node.

Note: Remember to empty the *content* before call this function

Parameters

<i>iterator_pair</i>	a iterator pair of the root node
<i>content</i>	whether to put constructed html code

5.16.3.4 void FeedXParser::ExtractGeoLocation (FeedItem * *item*, const IteratorPair & *iterator_pair*) [static],
[private]

Extract geographical location from the given tree and add to the item.

Parameters

<i>item</i>	where to add geographical location
<i>iterator_pair</i>	tree source

5.16.3.5 void FeedXParser::ExtractImages (FeedItem * *item*, const IteratorPair & *iterator_pair*) [static],
[private]

Extract images from the given tree and add to the item.

Parameters

<i>item</i>	where to add images
<i>iterator_pair</i>	tree source

5.16.3.6 const std::list< Item * > & FeedXParser::GetItemList () const [virtual]

Get the item list of parsed feed.

Returns

item list

Exceptions

<i>HasNoValue</i>	if the feed has no item list
-----------------------------------	------------------------------

Reimplemented from [nfrd::parser::Parser](#).

5.16.3.7 const DateTime & FeedXParser::GetLastBuildDate () const [virtual]

Get the last build date of the feed resource.

Usually, this data is provided in the feed resource, telling when the feed resource is generated. Some subclasses may use pseudo-LastBuildDate that the date is the post date of the latest item.

Returns

last build date of the feed resource

Exceptions

<i>HasNoValue</i>	if the item has no last build date
-----------------------------------	------------------------------------

Reimplemented from [nfrd::parser::Parser](#).

5.16.3.8 `std::string FeedXPather::GetLongestText (const std::string & source)` `[static], [private]`

Get the longest text from a html source code.

Parameters

<i>source</i>	html source code
---------------	------------------

Returns

the longest text in *source*

5.16.3.9 `void FeedXPather::PatchAndMatch (FeedItem * item)` `[static], [private]`

Patch an rss item using Patch and Match (Match and Patch) algorithm.

Parameters

<i>item</i>	an item to be patched.
-------------	------------------------

5.16.3.10 `void FeedXPather::ReadURL (const std::string & url)` `[virtual]`

Read resource from an URL and parse into a list of [Item](#).

Parameters

<i>url</i>	URL address of the feed resource
------------	----------------------------------

Exceptions

<i>InvalidSource</i>	if the url or the feed resource is invalid
--------------------------------------	--

Implements [nfrd::parser::Parser](#).

5.16.3.11 `void FeedXPather::RefineFeed (FeedItem * item)` `[static], [private]`

Refine a feed by stripping tags and extracting images.

Parameters

<i>item</i>	an item to be refined.
-------------	------------------------

5.16.3.12 `string::size_type FeedXParser::TextSize (const tree< htmlcxx::HTML::Node >::iterator & iterator)` `[static]`, `[private]`

Get the total text size of a html tree, counting all child nodes of a root node.

Parameters

<i>iterator</i>	a iterator of the root node
-----------------	-----------------------------

5.16.3.13 `std::string FeedXParser::TrimEmptyTag (const std::string & source)` `[static]`, `[private]`

Trim all empty a or div tags in the source.

Parameters

<i>source</i>	html source code
---------------	------------------

Returns

trimmed html source code

5.16.4 Member Data Documentation

5.16.4.1 `std::set< std::string > FeedXParser::allowed_tags` `[static]`, `[private]`

The tags that will not be changed.

5.16.4.2 `FeedParser* nfrd::parser::FeedXParser::parser` `[private]`

Feed parser to parse the actual feed.

5.16.4.3 `std::set< std::string > FeedXParser::trimmed_tags` `[static]`, `[private]`

The tage that only text part is kept.

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

- include/nfrd/[FeedXParser.h](#)
- src/[FeedXParser.cpp](#)

5.17 nfrd::misc::Image::File Class Reference

A class to represent an image file (as an ownner)

```
#include <Image.h>
```

Public Member Functions

- [File](#) (void *data, int size, [Type](#) type=UNKNOWN)
Construct image file by owning a block of memory.
- [~File](#) ()
Delete all dynamic memory, if any.

- `const void * GetData () const`
Get the memory block of the file.
- `int GetSize () const`
Get the size of the memory block.
- `Type GetType () const`
Get the image type.

Private Member Functions

- `File (File &file)`
Copy Constructor for [File](#).
- `File & operator= (File &file)`
Copy an image file from another image file.

Private Attributes

- `void * data`
memory block of the file
- `int size`
size of the memory block
- `Type type`
image type

5.17.1 Detailed Description

A class to represent an image file (as an owner)

5.17.2 Constructor & Destructor Documentation

5.17.2.1 `Image::File::File (void * data, int size, Type type = UNKNOWN)`

Construct image file by owning a block of memory.

Warning

this constructor should only be used with libgd

Parameters

<i>data</i>	starting memory address of the file
<i>size</i>	size of the memory block
<i>type</i>	type of the image

5.17.2.2 `Image::File::~~File ()`

Delete all dynamic memory, if any.

5.17.2.3 `nfrd::misc::Image::File::File (File &file) [private]`

Copy Constructor for [File](#).

Warning

copy constructor is disabled

Parameters

<i>file</i>	source file
-------------	-------------

Exceptions

<i>ImageException</i>	if fail to copy
---------------------------------------	-----------------

5.17.3 Member Function Documentation**5.17.3.1 const void * Image::File::GetData () const**

Get the memory block of the file.

Returns

the starting address of the memory

5.17.3.2 int Image::File::GetSize () const

Get the size of the memory block.

Returns

size of the memory block

5.17.3.3 Image::Type Image::File::GetType () const

Get the image type.

Returns

image type

5.17.3.4 File& nfrd::misc::Image::File::operator= (File & *file*) [private]

Copy an image file from another image file.

Warning

assignment operator is disabled

Parameters

<i>file</i>	source file
-------------	-------------

Returns

*this

Exceptions

ImageException	if fail to copy
--------------------------------	-----------------

5.17.4 Member Data Documentation

5.17.4.1 `void* nfrd::misc::Image::File::data` [private]

memory block of the file

5.17.4.2 `int nfrd::misc::Image::File::size` [private]

size of the memory block

5.17.4.3 `Type nfrd::misc::Image::File::type` [private]

image type

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

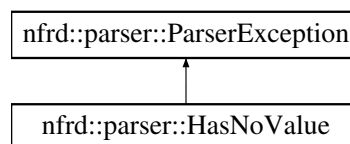
- [include/nfrd/Image.h](#)
- [src/Image.cpp](#)

5.18 nfrd::parser::HasNoValue Class Reference

Has no value exception.

```
#include <Parser.h>
```

Inheritance diagram for nfrd::parser::HasNoValue:



Public Member Functions

- [HasNoValue](#) (const std::string &msg)
Default Constructor for [HasNoValue](#), recording the error message.
- virtual [~HasNoValue](#) () throw ()
Delete dynamic memories, if any.

5.18.1 Detailed Description

Has no value exception.

[HasNoValue](#) is thrown when the attribute has no value.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 HasNoValue::HasNoValue (const std::string & msg) [explicit]

Default Constructor for [HasNoValue](#), recording the error message.

Parameters

<i>msg</i>	Error message
------------	---------------

5.18.2.2 HasNoValue::~HasNoValue () throw () [virtual]

Delete dynamic memories, if any.

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

- include/nfrd/[Parser.h](#)
- src/[Parser.cpp](#)

5.19 nfrd::misc::Image Class Reference

A class to store image and process image.

```
#include <Image.h>
```

Classes

- class [File](#)
A class to represent an image file (as an owner)

Public Types

- enum [Type](#) {
 [UNKNOWN](#), [GD](#), [GD2](#), [GIF](#),
 [JPEG](#), [PNG](#), [WBMP](#) }
Types of [Image](#).

Public Member Functions

- [Image](#) ()
Default Constructor for [Image](#).
- [Image](#) (const [Image](#) &rhs)
Copy Constructor for [Image](#).
- [~Image](#) ()
Delete all dynamic memory, if any.
- void [Load](#) (const [File](#) &file)
Load image from file mapped in memory.
- void [Load](#) (std::string url, [Type](#) type=[UNKNOWN](#))
Load image from specified URL.
- void [Load](#) (const std::vector< char > &data, [Type](#) type=[UNKNOWN](#))
Load image from file mapped in memory.

- void [Load](#) (const void *data, int size, [Type](#) type=[UNKNOWN](#))
Load image from file mapped in memory.
- std::auto_ptr< [File](#) > [ExportJpeg](#) (int quality=80) const
Export the image as a JPEG file.
- std::auto_ptr< [File](#) > [ExportPng](#) (int compression=-1) const
Export the image as a PNG file.
- std::auto_ptr< [Image](#) > [FitSize](#) (int width, int height)
Fit into the specified size, retaining the ratio with black padding.
- int [GetWidth](#) () const
Get the width of the image.
- int [GetHeight](#) () const
Get the height of the image.
- [Image](#) & [operator=](#) (const [Image](#) &image)
Copy an image from another image.

Private Attributes

- gdImagePtr [im](#)
A pointer to gdImage.

5.19.1 Detailed Description

A class to store image and process image.

5.19.2 Member Enumeration Documentation

5.19.2.1 enum nfrd::misc::Image::Type

Types of [Image](#).

Enumerator:

- UNKNOWN** Unknown format.
- GD** GD format: .gd.
- GD2** GD2 format: .gd2.
- GIF** Graphics Interchange format: .gif.
- JPEG** JPEG format: .jpg, .jpeg, .jpe .jif, .jfif, .jfi.
- PNG** Portable Network Graphics format: .png.
- WBMP** Wireless Application Protocol Bitmap format: .wbmp.

5.19.3 Constructor & Destructor Documentation

5.19.3.1 Image::Image ()

Default Constructor for [Image](#).

5.19.3.2 Image::Image (const Image & rhs)

Copy Constructor for [Image](#).

Parameters

<i>rhs</i>	Source instance to copy
------------	-------------------------

Exceptions

ImageException	if fail to copy
--------------------------------	-----------------

5.19.3.3 Image::~Image ()

Delete all dynamic memory, if any.

5.19.4 Member Function Documentation

5.19.4.1 std::auto_ptr< Image::File > Image::ExportJpeg (int *quality* = 80) const

Export the image as a JPEG file.

Parameters

<i>quality</i>	quality of jpeg in range of [0, 95]
----------------	-------------------------------------

Returns

an auto pointer to JPEG file

Exceptions

ImageException	if fail to allocate memory
--------------------------------	----------------------------

5.19.4.2 std::auto_ptr< Image::File > Image::ExportPng (int *compression* = -1) const

Export the image as a PNG file.

Parameters

<i>compression</i>	compression level in range of [0, 9] -1 means default by zlib
--------------------	---

Returns

an auto pointer to PNG file

Exceptions

ImageException	if fail to allocate memory
--------------------------------	----------------------------

5.19.4.3 `std::auto_ptr< Image > Image::FitSize (int width, int height)`

Fit into the specified size, retaining the ratio with black padding.

If the image is smaller than the specified size, the original picture will not be resized.

Parameters

<i>width</i>	specified size
<i>height</i>	specified height

Returns

resized image

Exceptions

<i>ImageException</i>	if fail to allocate memory
---------------------------------------	----------------------------

5.19.4.4 `int Image::GetHeight () const`

Get the height of the image.

Returns

height of the image

5.19.4.5 `int Image::GetWidth () const`

Get the width of the image.

Returns

width of the image

5.19.4.6 `void Image::Load (const File & file)`

Load image from file mapped in memory.

Parameters

<i>file</i>	file that contains a memory block of image
-------------	--

Exceptions

<i>ImageException</i>	if fail to load
---------------------------------------	-----------------

5.19.4.7 `void Image::Load (std::string url, Type type = UNKNOWN)`

Load image from specified URL.

Parameters

<i>url</i>	url address of the image
<i>type</i>	type of the image

Exceptions

<i>ImageException</i>	if fail to load
---------------------------------------	-----------------

5.19.4.8 void Image::Load (const std::vector< char > & data, Type type = UNKNOWN)

Load image from file mapped in memory.

Parameters

<i>data</i>	memory block of the file
<i>type</i>	type of the image

Exceptions

<i>ImageException</i>	if fail to load
---------------------------------------	-----------------

5.19.4.9 void Image::Load (const void * data, int size, Type type = UNKNOWN)

Load image from file mapped in memory.

Parameters

<i>data</i>	starting memory address of the file
<i>size</i>	size of the memory block
<i>type</i>	type of the image

Exceptions

<i>ImageException</i>	if fail to load
---------------------------------------	-----------------

5.19.4.10 Image & Image::operator= (const Image & image)

Copy an image from another image.

Parameters

<i>image</i>	source image
--------------	--------------

Returns

*this

Exceptions

<i>ImageException</i>	if fail to copy
---------------------------------------	-----------------

5.19.5 Member Data Documentation

5.19.5.1 gdImagePtr nfrd::misc::Image::im [mutable], [private]

A pointer to gdImage.

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

- include/nfrd/[Image.h](#)
- src/[Image.cpp](#)

5.20 nfrd::misc::ImageException Class Reference

General exception for [Image](#).

```
#include <Image.h>
```

Public Member Functions

- [ImageException](#) (const std::string &message)
Default Constructor for [ImageException](#), recording the error message.
- virtual [~ImageException](#) () throw ()
Delete dynamic memories, if any.
- virtual const char * [what](#) () const throw ()
Return error message.

Private Attributes

- std::string [msg](#)
Error message.

5.20.1 Detailed Description

General exception for [Image](#).

5.20.2 Constructor & Destructor Documentation

5.20.2.1 ImageException::ImageException (const std::string & message) [explicit]

Default Constructor for [ImageException](#), recording the error message.

Parameters

<i>message</i>	Error message
----------------	---------------

5.20.2.2 ImageException::~ImageException () throw () [virtual]

Delete dynamic memories, if any.

5.20.3 Member Function Documentation

5.20.3.1 `const char * ImageException::what () const throw ()` `[virtual]`

Return error message.

Returns

Error message

5.20.4 Member Data Documentation

5.20.4.1 `std::string nfrd::misc::ImageException::msg` `[private]`

Error message.

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

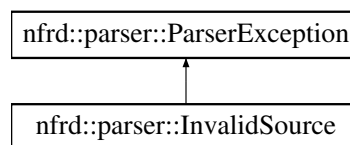
- [include/nfrd/Image.h](#)
- [src/Image.cpp](#)

5.21 nfrd::parser::InvalidSource Class Reference

Invalid source exception.

```
#include <Parser.h>
```

Inheritance diagram for `nfrd::parser::InvalidSource`:



Public Member Functions

- [InvalidSource](#) (const std::string &[msg](#))
Default Constructor for [InvalidSource](#), recording the error message.
- virtual [~InvalidSource](#) () throw ()
Delete dynamic memories, if any.

5.21.1 Detailed Description

Invalid source exception.

[InvalidSource](#) is thrown when the resource cannot be parsed by the parser.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 `InvalidSource::InvalidSource (const std::string & msg)` `[explicit]`

Default Constructor for [InvalidSource](#), recording the error message.

Parameters

<i>msg</i>	Error message
------------	---------------

5.21.2.2 InvalidSource::~InvalidSource () throw () [virtual]

Delete dynamic memories, if any.

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

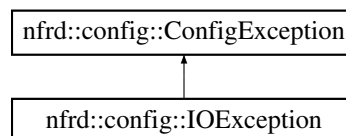
- include/nfrd/[Parser.h](#)
- src/[Parser.cpp](#)

5.22 nfrd::config::IOException Class Reference

Input/Output exception for config.

```
#include <ConfigManager.h>
```

Inheritance diagram for nfrd::config::IOException:



Public Member Functions

- [IOException](#) (const std::string &msg)
Default Constructor for [IOException](#), recording the error message.
- virtual [~IOException](#) () throw ()
Delete dynamic memories, if any.

5.22.1 Detailed Description

Input/Output exception for config.

5.22.2 Constructor & Destructor Documentation

5.22.2.1 IOException::IOException (const std::string & msg) [explicit]

Default Constructor for [IOException](#), recording the error message.

Parameters

<i>msg</i>	Error message
------------	---------------

5.22.2.2 IOException::~~IOException () throw () [virtual]

Delete dynamic memories, if any.

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

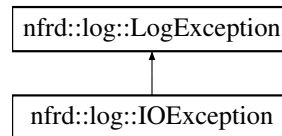
- include/nfrd/[ConfigManager.h](#)
- src/[ConfigManager.cpp](#)

5.23 nfrd::log::IOException Class Reference

Input/Output exception for config.

```
#include <LogManager.h>
```

Inheritance diagram for nfrd::log::IOException:



Public Member Functions

- [IOException](#) (const std::string &msg)
Default Constructor for [IOException](#), recording the error message.
- virtual [~IOException](#) () throw ()
Delete dynamic memories, if any.

5.23.1 Detailed Description

Input/Output exception for config.

5.23.2 Constructor & Destructor Documentation

5.23.2.1 IOException::IOException (const std::string & msg) [explicit]

Default Constructor for [IOException](#), recording the error message.

Parameters

<i>msg</i>	Error message
------------	---------------

5.23.2.2 IOException::~~IOException () throw () [virtual]

Delete dynamic memories, if any.

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

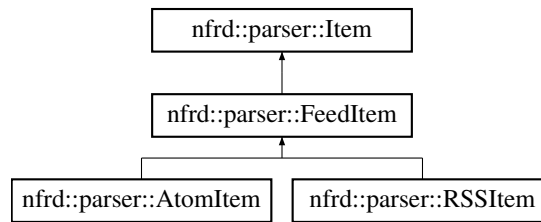
- include/nfrd/[LogManager.h](#)
- src/[LogManager.cpp](#)

5.24 nfrd::parser::Item Class Reference

A class to store details of an item obtained by the [Parser](#).

```
#include <Parser.h>
```

Inheritance diagram for nfrd::parser::Item:



Public Types

- typedef std::pair< std::string, [misc::Image::File](#) * > [Image](#)
A Image type is a pair of URL and image file.

Public Member Functions

- virtual [~Item](#) ()
Delete all dynamic memory, if any.
- virtual const std::string & [GetTitle](#) () const
Get the title of the item.
- virtual const std::string & [GetURL](#) () const
Get the URL where full edition of the item is.
- virtual const std::string & [GetContent](#) () const
Get the content of the item.
- virtual const [misc::DateTime](#) & [GetPostDate](#) () const
Get the post date of the item.
- virtual const std::string & [GetAuthor](#) () const
Get the author of the item.
- virtual const std::string & [GetGeoLocation](#) () const
Get the geographical location of the item.
- virtual const std::list< [Image](#) > & [GetImageList](#) () const
Get the image list of the item.
- virtual bool [HasTitle](#) () const
Test the item has title or not.
- virtual bool [HasURL](#) () const
Test the item has URL or not.
- virtual bool [HasContent](#) () const
Test the item has content or not.
- virtual bool [HasPostDate](#) () const
Test the item has post date or not.
- virtual bool [HasAuthor](#) () const
Test the item has author or not.
- virtual bool [HasGeoLocation](#) () const
Test the item has geographical location or not.
- virtual bool [HasImageList](#) () const
Test the item has image list or not.

5.24.1 Detailed Description

A class to store details of an item obtained by the [Parser](#).

The relationship with [Parser](#) is many-to-one, that [Parser](#) can have many [Item](#). The attribute can be added to this class at anytime and it will not make subclasses unable to compile. Instead, all subclasses will automatical derive new attributes.

5.24.2 Member Typedef Documentation

5.24.2.1 `typedef std::pair<std::string, misc::Image::File*> nfrd::parser::Item::Image`

A Image type is a pair of URL and image file.

Notice: image file part can be null that the image only contains URL, not the file. The image file could be a thumbnail and not the image that the URL points to.

5.24.3 Constructor & Destructor Documentation

5.24.3.1 `Item::~Item () [virtual]`

Delete all dynamic memory, if any.

5.24.4 Member Function Documentation

5.24.4.1 `const string & Item::GetAuthor () const [virtual]`

Get the author of the item.

Returns

author

Exceptions

<i>HasNoValue</i>	if the item has no author or if this function is not overridden
-----------------------------------	---

Reimplemented in [nfrd::parser::AtomItem](#), and [nfrd::parser::RSSItem](#).

5.24.4.2 `const string & Item::GetContent () const [virtual]`

Get the content of the item.

Returns

content

Exceptions

<i>HasNoValue</i>	if the item has no content or if this function is not overridden
-----------------------------------	--

Reimplemented in [nfrd::parser::AtomItem](#), and [nfrd::parser::RSSItem](#).

5.24.4.3 `const string & Item::GetGeoLocation () const` [virtual]

Get the geographical location of the item.

Returns

geographical location

Exceptions

<i>HasNoValue</i>	if the item has no geographical location or if this function is not overridden
-----------------------------------	--

Reimplemented in [nfrd::parser::FeedItem](#).

5.24.4.4 `const list< Item::Image > & Item::GetImageList () const` [virtual]

Get the image list of the item.

The relationship between the item and the image is one-to-many.

Returns

image list

Exceptions

<i>HasNoValue</i>	if the item has no image list or if this function is not overridden
-----------------------------------	---

Reimplemented in [nfrd::parser::FeedItem](#).

5.24.4.5 `const DateTime & Item::GetPostDate () const` [virtual]

Get the post date of the item.

Returns

post date

Exceptions

<i>HasNoValue</i>	if the item has no post date or if this function is not overridden
-----------------------------------	--

Reimplemented in [nfrd::parser::AtomItem](#), and [nfrd::parser::RSSItem](#).

5.24.4.6 `const string & Item::GetTitle () const` [virtual]

Get the title of the item.

Returns

title

Exceptions

<i>HasNoValue</i>	if the item has no title or if this function is not overridden
-----------------------------------	--

Reimplemented in [nfrd::parser::AtomItem](#), and [nfrd::parser::RSSItem](#).

5.24.4.7 `const string & Item::GetURL () const` `[virtual]`

Get the URL where full edition of the item is.

Returns

URL

Exceptions

<i>HasNoValue</i>	if the item has no URL or if this function is not overridden
-----------------------------------	--

Reimplemented in [nfrd::parser::AtomItem](#), and [nfrd::parser::RSSItem](#).

5.24.4.8 `bool Item::HasAuthor () const` `[virtual]`

Test the item has author or not.

Returns

true If the item has author
false If the item has no title or this function is not overridden

Reimplemented in [nfrd::parser::AtomItem](#), and [nfrd::parser::RSSItem](#).

5.24.4.9 `bool Item::HasContent () const` `[virtual]`

Test the item has content or not.

Returns

true If the item has content
false If the item has no content or this function is not overridden

Reimplemented in [nfrd::parser::AtomItem](#), and [nfrd::parser::RSSItem](#).

5.24.4.10 `bool Item::HasGeoLocation () const` `[virtual]`

Test the item has geographical location or not.

Returns

true If the item has geographical location
false If the item has no geographical location or this function is not overridden

Reimplemented in [nfrd::parser::FeedItem](#).

5.24.4.11 `bool Item::HasImageList () const` `[virtual]`

Test the item has image list or not.

Returns

true If the item has image list
false If the item has no image list or this function is not overridden

Reimplemented in [nfrd::parser::FeedItem](#).

5.24.4.12 `bool Item::HasPostDate () const [virtual]`

Test the item has post date or not.

Returns

true If the item has post date
false If the item has no post date or this function is not overridden

Reimplemented in [nfrd::parser::AtomItem](#), and [nfrd::parser::RSSItem](#).

5.24.4.13 `bool Item::HasTitle () const [virtual]`

Test the item has title or not.

Returns

true If the item has title
false If the item has no title or this function is not overridden

Reimplemented in [nfrd::parser::AtomItem](#), and [nfrd::parser::RSSItem](#).

5.24.4.14 `bool Item::HasURL () const [virtual]`

Test the item has URL or not.

Returns

true If the item has URL
false If the item has no URL or this function is not overridden

Reimplemented in [nfrd::parser::AtomItem](#), and [nfrd::parser::RSSItem](#).

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

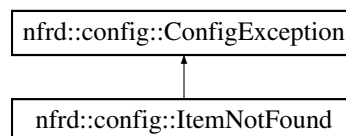
- [include/nfrd/Parser.h](#)
- [src/Parser.cpp](#)

5.25 `nfrd::config::ItemNotFound` Class Reference

Item not found.

```
#include <ConfigManager.h>
```

Inheritance diagram for `nfrd::config::ItemNotFound`:



Public Member Functions

- [ItemNotFound](#) (const std::string &msg)
Default Constructor for [ItemNotFound](#), recording the error message.
- virtual [~ItemNotFound](#) () throw ()
Delete dynamic memories, if any.

5.25.1 Detailed Description

Item not found.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 ItemNotFound::ItemNotFound (const std::string & msg) [explicit]

Default Constructor for [ItemNotFound](#), recording the error message.

Parameters

<i>msg</i>	Error message
------------	---------------

5.25.2.2 ItemNotFound::~ItemNotFound () throw () [virtual]

Delete dynamic memories, if any.

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

- include/nfrd/[ConfigManager.h](#)
- src/[ConfigManager.cpp](#)

5.26 nfrd::parser::FeedXParser::IteratorPair Struct Reference

A iterator pair structure used in [Construct\(\)](#) to store iterators for parent nodes.

Public Member Functions

- [IteratorPair](#) (const tree< htmlcxx::HTML::Node >::iterator &iterator, bool [is_allowed_tag](#)=false)
Initialising Constructor for [IteratorPair](#) using a iterator of a node.
- [IteratorPair](#) (const tree< htmlcxx::HTML::Node > &node, bool [is_allowed_tag](#)=false)
Initialising Constructor for [IteratorPair](#) using a node.

Public Attributes

- tree< htmlcxx::HTML::Node >::iterator [it](#)
Current working iterator.
- tree< htmlcxx::HTML::Node >::iterator [end](#)
The end mark of the current working iterator.
- bool [is_allowed_tag](#)
Whether the parent node is an allowed tag.

5.26.1 Detailed Description

A iterator pair structure used in [Construct\(\)](#) to store iterators for parent nodes.

See also

[Construct\(\)](#)

5.26.2 Constructor & Destructor Documentation

5.26.2.1 `nfrd::parser::FeedXParser::IteratorPair::IteratorPair (const tree< htmlcxx::HTML::Node >::iterator & iterator, bool is_allowed_tag = false)`

Initialising Constructor for [IteratorPair](#) using a iterator of a node.

Parameters

<i>iterator</i>	where it and end are extracted
<i>is_allowed_tag</i>	whether the parent node is an allowed tag

5.26.2.2 `nfrd::parser::FeedXParser::IteratorPair::IteratorPair (const tree< htmlcxx::HTML::Node > & node, bool is_allowed_tag = false)`

Initialising Constructor for [IteratorPair](#) using a node.

Parameters

<i>node</i>	where it and end are extracted
<i>is_allowed_tag</i>	whether the parent node is an allowed tag

5.26.3 Member Data Documentation

5.26.3.1 `tree<htmlcxx::HTML::Node>::iterator nfrd::parser::FeedXParser::IteratorPair::end`

The end mark of the current working iterator.

5.26.3.2 `bool nfrd::parser::FeedXParser::IteratorPair::is_allowed_tag`

Whether the parent node is an allowed tag.

5.26.3.3 `tree<htmlcxx::HTML::Node>::iterator nfrd::parser::FeedXParser::IteratorPair::it`

Current working iterator.

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

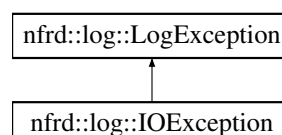
- include/nfrd/[FeedXParser.h](#)

5.27 nfrd::log::LogException Class Reference

General exception for log.

```
#include <LogManager.h>
```

Inheritance diagram for `nfrd::log::LogException`:



Public Member Functions

- [LogException](#) (const std::string &message)
Default Constructor for [LogException](#), recording the error message.
- virtual [~LogException](#) () throw ()
Delete dynamic memories, if any.
- virtual const char * [what](#) () const throw ()
Return error message.

Private Attributes

- std::string [msg](#)
Error message.

5.27.1 Detailed Description

General exception for log.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 [LogException::LogException \(const std::string & message \)](#) `[explicit]`

Default Constructor for [LogException](#), recording the error message.

Parameters

<i>message</i>	Error message
----------------	---------------

5.27.2.2 [LogException::~~LogException \(\) throw \(\)](#) `[virtual]`

Delete dynamic memories, if any.

5.27.3 Member Function Documentation

5.27.3.1 [const char * LogException::what \(\) const throw \(\)](#) `[virtual]`

Return error message.

Returns

Error message

5.27.4 Member Data Documentation

5.27.4.1 [std::string nfrd::log::LogException::msg](#) `[private]`

Error message.

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

- [include/nfrd/LogManager.h](#)
- [src/LogManager.cpp](#)

5.28 nfrd::log::LogManager Class Reference

Manage logs.

```
#include <LogManager.h>
```

Public Types

- enum [Type](#) { [NORMAL](#) = 0, [ERROR](#) = 1, [WARNING](#) = 2 }

Type of log.

Public Member Functions

- [LogManager](#) ()
Initialising Constructor for [LogManager](#).
- [~LogManager](#) ()
Delete all dynamic memory, if any.
- void [Enable](#) (std::string &filename)
Enable logging, writing logs to filename.
- void [Disable](#) ()
Disable logging system.
- bool [isEnabled](#) () const
Test the logging system is enabled or not.
- const [LogManager](#) & [operator\(\)](#) (const std::string &message, [Type](#) type=[NORMAL](#), const char title[]=0)
const
Log the message.
- const [LogManager](#) & [operator\(\)](#) (const std::string &message, const std::string &title, [Type](#) type=[NORMAL](#))
const
Log the message.

Private Attributes

- bool [enabled](#)
Logging is enabled or not.
- std::ofstream [fout](#)
File stream where the logs are written to.
- boost::mutex [io_mutex](#)
I/O mutex.

5.28.1 Detailed Description

Manage logs.

5.28.2 Member Enumeration Documentation

5.28.2.1 enum nfrd::log::LogManager::Type

Type of log.

Enumerator:

[NORMAL](#) Normal log. Marked as [LOG].

ERROR Error log. Marked as [ERR].

WARNING Warning log. Marked as [WRN].

5.28.3 Constructor & Destructor Documentation

5.28.3.1 LogManager::LogManager ()

Initialising Constructor for [LogManager](#).

5.28.3.2 LogManager::~~LogManager ()

Delete all dynamic memory, if any.

5.28.4 Member Function Documentation

5.28.4.1 void LogManager::Disable ()

Disable logging system.

5.28.4.2 void LogManager::Enable (std::string & filename)

Enable logging, writing logs to *filename*.

Parameters

<i>filename</i>	where logs will be written to.
-----------------	--------------------------------

Exceptions

IOException	If unable to write file.
-----------------------------	--------------------------

5.28.4.3 bool LogManager::isEnabled () const

Test the logging system is enabled or not.

5.28.4.4 const LogManager & LogManager::operator() (const std::string & message, Type type = NORMAL, const char title[] = 0) const

Log the message.

For example:

```
LogManager log;
log.Enable("logfile.log");
log("started", NORMAL, "LogManager");
```

will output

[Sat May 20 15:21:51 2000][LOG] [LogManager](#): started

Parameters

<i>message</i>	message to log
<i>type</i>	log type
<i>title</i>	title of the log

Returns

a reference to self

5.28.4.5 `const LogManager & LogManager::operator() (const std::string & message, const std::string & title, Type type = NORMAL) const`

Log the message.

Parameters

<i>message</i>	message to log
<i>title</i>	title (module name) of the log
<i>type</i>	log type

Returns

a reference to self

See also

const [LogManager](#)& operator()(const std::string& message, [Type](#) type = [NORMAL](#), const char title[] = 0) const;

5.28.5 Member Data Documentation

5.28.5.1 `bool nfrd::log::LogManager::enabled` [private]

Logging is enabled or not.

5.28.5.2 `std::ofstream nfrd::log::LogManager::fout` [mutable], [private]

File stream where the logs are written to.

5.28.5.3 `boost::mutex nfrd::log::LogManager::io_mutex` [mutable], [private]

I/O mutex.

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

- include/nfrd/[LogManager.h](#)
- src/[LogManager.cpp](#)

5.29 nfrd::Master Class Reference

Manage, contain and access all components of nfrd.

```
#include <Master.h>
```

Public Member Functions

- [Master](#) ()
Initialising Constructor for [Master](#).
- [~Master](#) ()

- Delete all dynamic memory, if any.*
 - void [Main](#) (const std::string &configFile)
 - Program entrance, loading everything according to the configuration file.*
 - void [Terminate](#) ()
 - Send terminate signal to the [Master](#).*
 - bool [IsOnline](#) () const
 - Test the master is online or not.*
 - const time_t & [GetStartTime](#) () const
 - Get start time of [Master](#).*
 - [config::ConfigManager](#) & [GetConfig](#) ()
 - Get the access to config module.*
 - [module::Module](#) * [GetModule](#) (const std::string &name)
 - Get the access to a module.*
 - void [SetModule](#) (const std::string &name, bool start)
 - Set a module to start or stop.*
 - void [SetModule](#) ([module::Module](#) *module, bool start)
 - Set a module to start or stop.*

Static Public Member Functions

- static const char * [GetVersion](#) ()
 - Get the version of the nfrd.*

Private Types

- typedef std::pair
 - < [module::Module](#) *, bool > [Task](#)
 - Provide convenience to have pair frequently used in task.*

Private Member Functions

- [module::Module](#) * [LoadModule](#) (const std::string &name)
 - Load a specified module according to config.*
- void [UnloadModule](#) (const std::string &name)
 - Unload a specified module.*
- void [LoadModules](#) ()
 - Load modules according to config.*
- void [UnloadModules](#) ()
 - Unload modules.*

Private Attributes

- [config::ConfigManager](#) config
 - Configuration Module (required)*
- [log::LogManager](#) log
 - Logging Module (required)*
- std::map< std::string,
 - [module::Module](#) * > [module](#)
 - Service Modules.*
- std::queue< [Task](#) > [task_queue](#)

- *Task queue.*
- bool [online](#)
Indicate [Master](#) is online or not.
- time_t [start_time](#)
Start time.
- boost::mutex [mutex](#)
Mutex in [Main\(\)](#)
- boost::condition [condition](#)
Condition variable that controls the main thread flow.

5.29.1 Detailed Description

Manage, contain and access all components of nfrd.

5.29.2 Member Typedef Documentation

5.29.2.1 `typedef std::pair<module::Module*, bool> nfrd::Master::Task` [private]

Provide convenience to have pair frequently used in task.

5.29.3 Constructor & Destructor Documentation

5.29.3.1 `Master::Master ()`

Initialising Constructor for [Master](#).

5.29.3.2 `Master::~Master ()`

Delete all dynamic memory, if any.

5.29.4 Member Function Documentation

5.29.4.1 `config::ConfigManager & Master::GetConfig ()`

Get the access to config module.

Returns

config module

5.29.4.2 `module::Module * Master::GetModule (const std::string & name)`

Get the access to a module.

Parameters

<i>name</i>	name of module to be accessed
-------------	-------------------------------

Returns

a pointer to the module. NULL if module not found

5.29.4.3 `const time_t & Master::GetStartTime () const`

Get start time of [Master](#).

Returns

start time

5.29.4.4 `const char * Master::GetVersion () [static]`

Get the version of the nfrd.

Returns

version string

5.29.4.5 `bool Master::IsOnline () const`

Test the master is online or not.

Returns

true if online

5.29.4.6 `Module * Master::LoadModule (const std::string & name) [private]`

Load a specified module according to config.

If a module exists, it will be reloaded.

Parameters

<i>name</i>	name of module to be loaded
-------------	-----------------------------

Exceptions

<i>ModuleException</i>	if fail to load or module not found
------------------------	-------------------------------------

Returns

a pointer to newly loaded module

5.29.4.7 `void Master::LoadModules () [private]`

Load modules according to config.

5.29.4.8 `void Master::Main (const std::string & configFile)`

Program entrance, loading everything according to the configuration file.

Parameters

<i>configFile</i>	filename of the config file
-------------------	-----------------------------

Exceptions

<i>ModuleException</i>	if unable to proceed config module or log module
------------------------	--

5.29.4.9 void Master::SetModule (const std::string & *name*, bool *start*)

Set a module to start or stop.

Parameters

<i>name</i>	module name
<i>start</i>	start the module if true, else stop the module

Exceptions

<i>ModuleException</i>	if module not found
------------------------	---------------------

5.29.4.10 void Master::SetModule (module::Module * *module*, bool *start*)

Set a module to start or stop.

Parameters

<i>module</i>	module to be operated
<i>start</i>	start the module if true, else stop the module

5.29.4.11 void Master::Terminate ()

Send terminate signal to the [Master](#).

5.29.4.12 void Master::UnloadModule (const std::string & *name*) [private]

Unload a specified module.

Parameters

<i>name</i>	name of module to be unloaded
-------------	-------------------------------

Exceptions

<i>ModuleException</i>	if fail to load or module not loaded
------------------------	--------------------------------------

5.29.4.13 void Master::UnloadModules () [private]

Unload modules.

5.29.5 Member Data Documentation

5.29.5.1 boost::condition nfrd::Master::condition [private]

Condition variable that controls the main thread flow.

5.29.5.2 `config::ConfigManager nfrd::Master::config` [private]

Configuration Module (required)

5.29.5.3 `log::LogManager nfrd::Master::log` [private]

Logging Module (required)

5.29.5.4 `std::map<std::string, module::Module*> nfrd::Master::module` [private]

Service Modules.

5.29.5.5 `boost::mutex nfrd::Master::mutex` [private]

Mutex in [Main\(\)](#)

5.29.5.6 `bool nfrd::Master::online` [private]

Indicate [Master](#) is online or not.

5.29.5.7 `time_t nfrd::Master::start_time` [private]

Start time.

5.29.5.8 `std::queue<Task> nfrd::Master::task_queue` [private]

Task queue.

Every element in the task queue is a pair of `Controller*` and `bool`. The `bool` value indicate the switch of the Controller whether to start(true) or stop(false) the module.

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

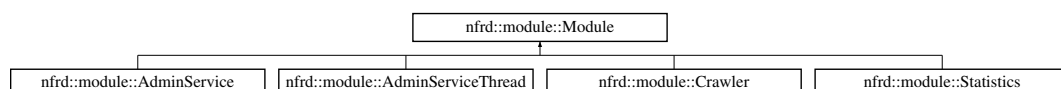
- `include/nfrd/Master.h`
- `src/Master.cpp`

5.30 nfrd::module::Module Class Reference

A generalised module interface class, providing all the interfaces of a module that start or stop.

```
#include <Module.h>
```

Inheritance diagram for `nfrd::module::Module`:



Public Types

- enum `Status` { `RUNNING` = 1, `STOPPED` = 0, `STARTING` = 2, `STOPPING` = 3 }
- Running status of the module.*

Public Member Functions

- [Module](#) (const std::string &name, const [config::ConfigManager](#) &config, const [log::LogManager](#) &log)
Initialising Constructor for [Module](#).
- [Module](#) (const std::string &name, const [Module](#) &module)
Initialising Constructor for [Module](#).
- virtual [~Module](#) ()
Delete all dynamic memory, if any.
- virtual void [operator\(\)](#) ()=0
Start the service/module in current thread.
- virtual void [Start](#) ()
Start the service/module in a new thread.
- virtual void [Stop](#) ()
Stop the service/module, joining the thread.
- const std::string & [GetName](#) () const
Get the name of the module.
- boost::thread & [GetThread](#) ()
Get the thread for the module (used in multi-threaded condition)
- [Status](#) [GetStatus](#) () const
Get the running status.
- const char * [GetStatusString](#) () const
Get the running status in string form.

Protected Attributes

- const std::string name
[Module](#) name.
- boost::thread thread
Running thread.
- [Status](#) status
Running status.
- const [config::ConfigManager](#) & config
Configuration access.
- const [log::LogManager](#) & log
Access to logging system.

5.30.1 Detailed Description

A generalised module interface class, providing all the interfaces of a module that start or stop. Since it is an abstract class, all pure virtual functions have to be implemented by the subclasses.

5.30.2 Member Enumeration Documentation

5.30.2.1 enum nfrd::module::Module::Status

Running status of the module.

Enumerator:

- RUNNING*** The module is running.
- STOPPED*** The module is stoped.
- STARTING*** The module is starting.
- STOPPING*** The module is stopping.

5.30.3 Constructor & Destructor Documentation

5.30.3.1 Module::Module (const std::string & *name*, const config::ConfigManager & *config*, const log::LogManager & *log*)

Initialising Constructor for [Module](#).

Initialise status to false (not running).

Parameters

<i>name</i>	module name
<i>config</i>	config manager
<i>log</i>	logger

5.30.3.2 Module::Module (const std::string & *name*, const Module & *module*)

Initialising Constructor for [Module](#).

Initialise status to false (not running). Using the same config and log as *module*

Parameters

<i>name</i>	module name
<i>module</i>	other module

5.30.3.3 Module::~~Module () [virtual]

Delete all dynamic memory, if any.

5.30.4 Member Function Documentation

5.30.4.1 const std::string & Module::GetName () const

Get the name of the module.

Returns

module name

5.30.4.2 Module::Status Module::GetStatus () const

Get the running status.

Returns

running status

5.30.4.3 const char * Module::GetStatusString () const

Get the running status in string form.

Returns

running status in string form

5.30.4.4 `boost::thread & Module::GetThread ()`

Get the thread for the module (used in multi-threaded condition)

Returns

running thread

5.30.4.5 `virtual void nfrd::module::Module::operator()() [pure virtual]`

Start the service/module in current thread.

Implemented in [nfrd::module::Crawler](#), [nfrd::module::AdminServiceThread](#), [nfrd::module::AdminService](#), and [nfrd::module::Statistics](#).

5.30.4.6 `void Module::Start () [virtual]`

Start the service/module in a new thread.

5.30.4.7 `void Module::Stop () [virtual]`

Stop the service/module, joining the thread.

Reimplemented in [nfrd::module::Crawler](#), [nfrd::module::AdminService](#), and [nfrd::module::Statistics](#).

5.30.5 Member Data Documentation

5.30.5.1 `const config::ConfigManager& nfrd::module::Module::config [protected]`

Configuration access.

5.30.5.2 `const log::LogManager& nfrd::module::Module::log [protected]`

Access to logging system.

5.30.5.3 `const std::string nfrd::module::Module::name [protected]`

[Module](#) name.

5.30.5.4 `Status nfrd::module::Module::status [protected]`

Running status.

5.30.5.5 `boost::thread nfrd::module::Module::thread [protected]`

Running thread.

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

- [include/nfrd/Module.h](#)
- [src/Module.cpp](#)

5.31 nfrd::module::ModuleException Class Reference

General exception for module.

```
#include <Module.h>
```

Public Member Functions

- [ModuleException](#) (const std::string &message)
Default Constructor for [ModuleException](#), recording the error message.
- virtual [~ModuleException](#) () throw ()
Delete dynamic memories, if any.
- virtual const char * [what](#) () const throw ()
Return error message.

Private Attributes

- std::string [msg](#)
Error message.

5.31.1 Detailed Description

General exception for module.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 [ModuleException::ModuleException](#) (const std::string & *message*) [explicit]

Default Constructor for [ModuleException](#), recording the error message.

Parameters

<i>message</i>	Error message
----------------	---------------

5.31.2.2 [ModuleException::~~ModuleException](#) () throw () [virtual]

Delete dynamic memories, if any.

5.31.3 Member Function Documentation

5.31.3.1 const char * [ModuleException::what](#) () const throw () [virtual]

Return error message.

Returns

Error message

5.31.4 Member Data Documentation

5.31.4.1 `std::string nfrd::module::ModuleException::msg` [private]

Error message.

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

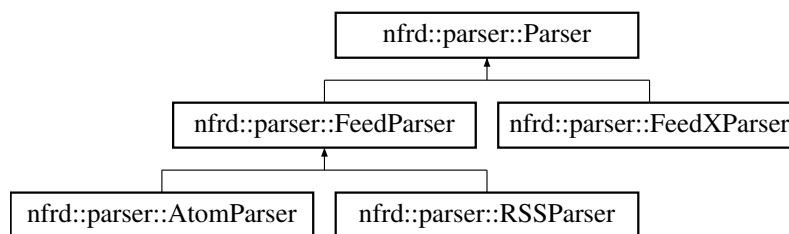
- [include/nfrd/Module.h](#)
- [src/Module.cpp](#)

5.32 `nfrd::parser::Parser` Class Reference

A generalised parser interface class, providing all the interfaces of a class that reads resource from an URL and parse it into a list of [Item](#).

```
#include <Parser.h>
```

Inheritance diagram for `nfrd::parser::Parser`:



Public Member Functions

- virtual `~Parser()`
Delete all dynamic memory, if any.
- virtual void `ReadURL(const std::string &url)=0`
Read resource from an URL and parse into a list of [Item](#).
- virtual const `misc::DateTime & GetLastBuildDate()` const
Get the last build date of the feed resource.
- virtual const `std::list< Item * > & GetItemList()` const
Get the item list of parsed feed.

5.32.1 Detailed Description

A generalised parser interface class, providing all the interfaces of a class that reads resource from an URL and parse it into a list of [Item](#).

Since it is an abstract class, all pure virtual functions have to be implemented by the subclasses.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 `Parser::~Parser()` [virtual]

Delete all dynamic memory, if any.

5.32.3 Member Function Documentation

5.32.3.1 `const std::list< Item * > & Parser::GetItemList () const` `[virtual]`

Get the item list of parsed feed.

Returns

item list

Exceptions

<i>HasNoValue</i>	if the feed has no item list
-----------------------------------	------------------------------

Reimplemented in [nfrd::parser::FeedParser](#), and [nfrd::parser::FeedXParser](#).

5.32.3.2 `const DateTime & Parser::GetLastBuildDate () const` `[virtual]`

Get the last build date of the feed resource.

Usually, this data is provided in the feed resource, telling when the feed resource is generated. Some subclasses may use pseudo-LastBuildDate that the date is the post date of the latest item.

Returns

last build date of the feed resource

Exceptions

<i>HasNoValue</i>	if the item has no last build date
-----------------------------------	------------------------------------

Reimplemented in [nfrd::parser::AtomParser](#), [nfrd::parser::RSSParser](#), and [nfrd::parser::FeedXParser](#).

5.32.3.3 `virtual void nfrd::parser::Parser::ReadURL (const std::string & url)` `[pure virtual]`

Read resource from an URL and parse into a list of [Item](#).

Parameters

<i>url</i>	URL address of the feed resource
------------	----------------------------------

Exceptions

<i>InvalidSource</i>	if the url or the feed resource is invalid
--------------------------------------	--

Implemented in [nfrd::parser::FeedParser](#), and [nfrd::parser::FeedXParser](#).

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

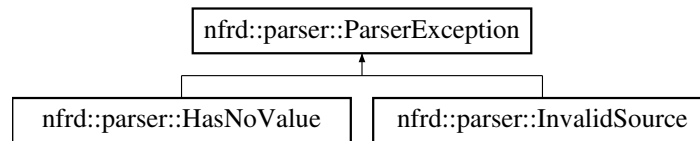
- include/nfrd/[Parser.h](#)
- src/[Parser.cpp](#)

5.33 nfrd::parser::ParserException Class Reference

General exception for parser.

```
#include <Parser.h>
```

Inheritance diagram for nfrd::parser::ParserException:



Public Member Functions

- [ParserException](#) (const std::string &message)
Default Constructor for [ParserException](#), recording the error message.
- virtual [~ParserException](#) () throw ()
Delete dynamic memories, if any.
- virtual const char * [what](#) () const throw ()
Return error message.

Private Attributes

- std::string [msg](#)
Error message.

5.33.1 Detailed Description

General exception for parser.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 [ParserException::ParserException](#) (const std::string & *message*) [explicit]

Default Constructor for [ParserException](#), recording the error message.

Parameters

<i>message</i>	Error message
----------------	---------------

5.33.2.2 [ParserException::~~ParserException](#) () throw () [virtual]

Delete dynamic memories, if any.

5.33.3 Member Function Documentation

5.33.3.1 const char * [ParserException::what](#) () const throw () [virtual]

Return error message.

Returns

Error message

5.33.4 Member Data Documentation

5.33.4.1 std::string nfrd::parser::ParserException::msg [private]

Error message.

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

- include/nfrd/Parser.h
- src/Parser.cpp

5.34 nfrd::module::QueueItem Class Reference

```
#include <QueueItem.h>
```

Public Member Functions

- [QueueItem](#) (nfrd::QueueItem &item)
Initialise the queue item as a copy of the given queue item.
- [QueueItem](#) (unsigned int, short [feedType](#), float [contentUpdateAverage](#), int numUsers)
Initialise the queue item with the required values to calculate the item's priority.
- void [IncrementPriority](#) (int maxpop)
Increment the priority of this queue item, done every time the heap is sorted.
- void [CalculateConstant](#) (int numUsersInSystem)
Calculate the constant part of the priority, based on the values that do not change very often.
- int [GetFeedID](#) () const
Get the feed id of the item.
- void [SetContentUpdateAverage](#) (float [contentUpdateAverage](#))
Set a new value for content update average.
- void [SetFeedType](#) (short [feedType](#))
Set the type of feed represented.
- void [SetUsersAffected](#) (int [usersAffected](#))
Set the number of users affected by the feed.
- void [ResetPriority](#) ()

Static Public Member Functions

- static bool [feed_comparer](#) ([QueueItem](#) *itemA, [QueueItem](#) *itemB)
Compare two feeds based on thier priority.

Private Attributes

- unsigned int [id](#)
ID of the represented feed.
- float [priority](#)
Calculated priority value for sorting.
- float [priorityConstant](#)
Constant priority value, used in the priority calculation.

- float `timeSpentInQueue`
The relative amount of time spent in the queue (iterations)
- float `contentUpdateAverage`
The average time between updates for the feed.
- short `feedType`
The type of feed represented.
- int `usersAffected`
The number of users affected by the feed.

5.34.1 Constructor & Destructor Documentation

5.34.1.1 `QueueItem::QueueItem (nfdb::QueueItem & item)`

Initialise the queue item as a copy of the given queue item.

5.34.1.2 `QueueItem::QueueItem (unsigned int feedId, short feedType, float contentUpdateAverage, int numUsers)`

Initialise the queue item with the required values to calculate the item's priority.

5.34.2 Member Function Documentation

5.34.2.1 `void QueueItem::CalculateConstant (int numUsersInSystem)`

Calculate the constant part of the priority, based on the values that do not change very often.

5.34.2.2 `static bool nfrd::module::QueueItem::feed_comparer (QueueItem * itemA, QueueItem * itemB) [inline], [static]`

Compare two feeds based on thier priority.

5.34.2.3 `int QueueItem::GetFeedID () const`

Get the feed id of the item.

5.34.2.4 `void QueueItem::IncrementPriority (int maxpop)`

Increment the priority of this queue item, done every time the heap is sorted.

Parameters

<code>maxpop</code>	Factor in the time it has been since last sort
---------------------	--

5.34.2.5 `void QueueItem::ResetPriority ()`

5.34.2.6 void QueueItem::SetContentUpdateAverage (float *contentUpdateAverage*)

Set a new value for content update average.

5.34.2.7 void QueueItem::SetFeedType (short *feedType*)

Set the type of feed represented.

5.34.2.8 void QueueItem::SetUsersAffected (int *usersAffected*)

Set the number of users affected by the feed.

5.34.3 Member Data Documentation

5.34.3.1 float nfrd::module::QueueItem::contentUpdateAverage [private]

The average time between updates for the feed.

5.34.3.2 short nfrd::module::QueueItem::feedType [private]

The type of feed represented.

5.34.3.3 unsigned int nfrd::module::QueueItem::id [private]

ID of the represented feed.

5.34.3.4 float nfrd::module::QueueItem::priority [private]

Calculated priority value for sorting.

5.34.3.5 float nfrd::module::QueueItem::priorityConstant [private]

Constant priority value, used in the priority calculation.

5.34.3.6 float nfrd::module::QueueItem::timeSpentInQueue [private]

The relative amount of time spent in the queue (iterations)

5.34.3.7 int nfrd::module::QueueItem::usersAffected [private]

The number of users affected by the feed.

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

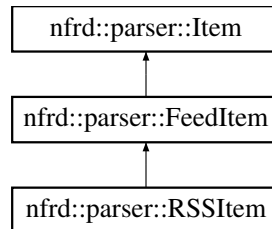
- [include/nfrd/QueueItem.h](#)
- [src/QueueItem.cpp](#)

5.35 nfrd::parser::RSSItem Class Reference

A class to store details of an item obtained by the [RSSParser](#).

```
#include <RSSParser.h>
```

Inheritance diagram for nfrd::parser::RSSItem:



Public Member Functions

- [RSSItem](#) ()
Initialising Constructor for [RSSItem](#).
- [~RSSItem](#) ()
Delete all dynamic memory, if any.
- const std::string & [GetTitle](#) () const
Get the title of the item.
- const std::string & [GetURL](#) () const
Get the URL where full edition of the item is.
- const std::string & [GetContent](#) () const
Get the content of the item.
- const [misc::DateTime](#) & [GetPostDate](#) () const
Get the post date of the item.
- const std::string & [GetAuthor](#) () const
Get the author of the item.
- void [SetTitle](#) (const char *source)
Set the title of the item.
- void [SetURL](#) (const char *source)
Set the url of the item.
- void [SetContent](#) (const char *source)
Set the content of the item.
- void [SetContent](#) (const std::string &source)
Set the content of the item.
- void [SetPostDate](#) (const [misc::DateTime](#) *source)
Set the post date of the item.
- void [SetAuthor](#) (const char *source)
Set the author of the item.
- bool [HasTitle](#) () const
Test the item has title or not.
- bool [HasURL](#) () const
Test the item has URL or not.
- bool [HasContent](#) () const
Test the item has content or not.
- bool [HasPostDate](#) () const
Test the item has post date or not.
- bool [HasAuthor](#) () const
Test the item has author or not.

Private Attributes

- `std::string * title`
Title of the item.
- `std::string * url`
URL of the item.
- `std::string * content`
Content of the item.
- `misc::DateTime * postDate`
Post date of the item.
- `std::string * author`
Author of the item.

5.35.1 Detailed Description

A class to store details of an item obtained by the [RSSParser](#).

See also

[FeedItem](#)

5.35.2 Constructor & Destructor Documentation

5.35.2.1 `RSSItem::RSSItem ()`

Initialising Constructor for [RSSItem](#).

Initialise everything to zero/null.

5.35.2.2 `RSSItem::~~RSSItem ()`

Delete all dynamic memory, if any.

5.35.3 Member Function Documentation

5.35.3.1 `const string & RSSItem::GetAuthor () const` [virtual]

Get the author of the item.

Returns

author

Exceptions

HasNoValue	if the item has no author or if this function is not overridden
----------------------------	---

Reimplemented from [nfrd::parser::Item](#).

5.35.3.2 `const string & RSSItem::GetContent () const` [virtual]

Get the content of the item.

Returns

content

Exceptions

<i>HasNoValue</i>	if the item has no content or if this function is not overridden
-----------------------------------	--

Reimplemented from [nfrd::parser::Item](#).

5.35.3.3 `const DateTime & RSSItem::GetPostDate () const` `[virtual]`

Get the post date of the item.

Returns

post date

Exceptions

<i>HasNoValue</i>	if the item has no post date or if this function is not overridden
-----------------------------------	--

Reimplemented from [nfrd::parser::Item](#).

5.35.3.4 `const string & RSSItem::GetTitle () const` `[virtual]`

Get the title of the item.

Returns

title

Exceptions

<i>HasNoValue</i>	if the item has no title or if this function is not overridden
-----------------------------------	--

Reimplemented from [nfrd::parser::Item](#).

5.35.3.5 `const string & RSSItem::GetURL () const` `[virtual]`

Get the URL where full edition of the item is.

Returns

URL

Exceptions

<i>HasNoValue</i>	if the item has no URL or if this function is not overridden
-----------------------------------	--

Reimplemented from [nfrd::parser::Item](#).

5.35.3.6 `bool RSSItem::HasAuthor () const` `[virtual]`

Test the item has author or not.

Returns

true If the item has author
false If the item has no title or this function is not overridden

Reimplemented from [nfrd::parser::Item](#).

5.35.3.7 bool RSSItem::HasContent () const [virtual]

Test the item has content or not.

Returns

true If the item has content
false If the item has no content or this function is not overridden

Reimplemented from [nfrd::parser::Item](#).

5.35.3.8 bool RSSItem::HasPostDate () const [virtual]

Test the item has post date or not.

Returns

true If the item has post date
false If the item has no post date or this function is not overridden

Reimplemented from [nfrd::parser::Item](#).

5.35.3.9 bool RSSItem::HasTitle () const [virtual]

Test the item has title or not.

Returns

true If the item has title
false If the item has no title or this function is not overridden

Reimplemented from [nfrd::parser::Item](#).

5.35.3.10 bool RSSItem::HasURL () const [virtual]

Test the item has URL or not.

Returns

true If the item has URL
false If the item has no URL or this function is not overridden

Reimplemented from [nfrd::parser::Item](#).

5.35.3.11 void RSSItem::SetAuthor (const char * source)

Set the author of the item.

Parameters

<i>source</i>	author of the item
---------------	--------------------

5.35.3.12 void RSSItem::SetContent (const char * *source*)

Set the content of the item.

Parameters

<i>source</i>	content of the item
---------------	---------------------

5.35.3.13 void nfrd::parser::RSSItem::SetContent (const std::string & *source*) [virtual]

Set the content of the item.

Parameters

<i>source</i>	content of the item
---------------	---------------------

See also

[FeedItem](#)

Implements [nfrd::parser::FeedItem](#).

5.35.3.14 void RSSItem::SetPostDate (const misc::DateTime * *source*)

Set the post date of the item.

Parameters

<i>source</i>	post date of the item
---------------	-----------------------

5.35.3.15 void RSSItem::SetTitle (const char * *source*)

Set the title of the item.

Parameters

<i>source</i>	title of the item
---------------	-------------------

5.35.3.16 void RSSItem::SetURL (const char * *source*)

Set the url of the item.

Parameters

<i>source</i>	url of the item
---------------	-----------------

5.35.4 Member Data Documentation

5.35.4.1 `std::string* nfrd::parser::RSSItem::author` [private]

Author of the item.

Optional in RSS. Original tag in RSS: author or dc:creator

5.35.4.2 `std::string* nfrd::parser::RSSItem::content` [private]

Content of the item.

Optional in RSS. Original tag in RSS: description or content:encoded

5.35.4.3 `misc::DateTime* nfrd::parser::RSSItem::postDate` [private]

Post date of the item.

Optional in RSS. Original tag in RSS: pubDate

5.35.4.4 `std::string* nfrd::parser::RSSItem::title` [private]

Title of the item.

Optional in RSS. Original tag in RSS: title

5.35.4.5 `std::string* nfrd::parser::RSSItem::url` [private]

URL of the item.

Optional in RSS. Original tag in RSS: link

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

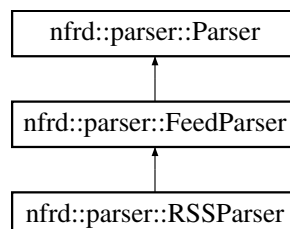
- [include/nfrd/RSSParser.h](#)
- [src/RSSParser.cpp](#)

5.36 nfrd::parser::RSSParser Class Reference

A parser to parse RSS feeds.

```
#include <RSSParser.h>
```

Inheritance diagram for nfrd::parser::RSSParser:

**Public Member Functions**

- [RSSParser\(\)](#)
Initialising Constructor for [RSSParser](#).

- virtual [~RSSParser](#) ()
Delete all dynamic memory, if any.
- void [ReadDom](#) (const rapidxml::xml_document<> &doc)
Parse feed from a dom tree (xml document) into a list of [Item](#).
- const [misc::DateTime](#) & [GetLastBuildDate](#) () const
Get the last build date of the feed resource.

Protected Attributes

- [misc::DateTime](#) * [lastBuildDate](#)
Last build date of the RSS feed Required in RSS.

Private Member Functions

- const char * [GetValue](#) (rapidxml::xml_node<> *node)
Get the text value from a node.

Private Attributes

- std::string [buffer](#)
String buffer used by [GetValue\(\)](#).

5.36.1 Detailed Description

A parser to parse RSS feeds.

Standard: <http://cyber.law.harvard.edu/rss/rss.html>

See also

[Parser](#)

5.36.2 Constructor & Destructor Documentation

5.36.2.1 RSSParser::RSSParser ()

Initialising Constructor for [RSSParser](#).

5.36.2.2 RSSParser::~RSSParser () [virtual]

Delete all dynamic memory, if any.

5.36.3 Member Function Documentation

5.36.3.1 const DateTime & RSSParser::GetLastBuildDate () const [virtual]

Get the last build date of the feed resource.

Usually, this data is provided in the feed resource, telling when the feed resource is generated. Some subclasses may use pseudo-LastBuildDate that the date is the post date of the latest item.

Returns

last build date of the feed resource

Exceptions

<i>HasNoValue</i>	if the item has no last build date
-----------------------------------	------------------------------------

Reimplemented from [nfrd::parser::Parser](#).

5.36.3.2 `const char * RSSParser::GetValue (rapidxml::xml_node<> * node) [private]`

Get the text value from a node.

If a node has multiple of CDATA nodes, the value will be appended together.

Parameters

<i>node</i>	the node for getting value
-------------	----------------------------

Returns

value from *node*.

5.36.3.3 `void RSSParser::ReadDom (const rapidxml::xml_document<> & doc) [virtual]`

Parse feed from a dom tree (xml document) into a list of [Item](#).

Parameters

<i>doc</i>	parsed xml document of the feed resource
------------	--

Exceptions

<i>InvalidSource</i>	if the dom or the feed resource is invalid
--------------------------------------	--

Implements [nfrd::parser::FeedParser](#).

5.36.4 Member Data Documentation

5.36.4.1 `std::string nfrd::parser::RSSParser::buffer [private]`

String buffer used by [GetValue\(\)](#).

5.36.4.2 `misc::DateTime* nfrd::parser::RSSParser::lastBuildDate [protected]`

Last build date of the RSS feed Required in RSS.

Original tag in RSS: lastBuildDate

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

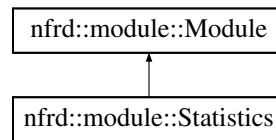
- [include/nfrd/RSSParser.h](#)
- [src/RSSParser.cpp](#)

5.37 nfrd::module::Statistics Class Reference

Periodly record and update the statistics information.

```
#include <Statistics.h>
```

Inheritance diagram for nfrd::module::Statistics:



Public Member Functions

- [Statistics](#) (const [config::ConfigManager](#) &*config*, const [log::LogManager](#) &*log*)
Initialising Constructor for [Statistics](#).
- [~Statistics](#) ()
Delete all dynamic memory, if any.
- void [operator\(\)](#) ()
Start the service/module in current thread.
- void [Stop](#) ()
Stop the service/module, joining the thread.

Private Attributes

- int [period](#)
Period of updating or recording (in seconds)

Additional Inherited Members

5.37.1 Detailed Description

Periodly record and update the statistics information.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 Statistics::Statistics (const [config::ConfigManager](#) & *config*, const [log::LogManager](#) & *log*)

Initialising Constructor for [Statistics](#).

Parameters

<i>config</i>	The config manager to use
<i>log</i>	The log manager to use

5.37.2.2 Statistics::~~Statistics ()

Delete all dynamic memory, if any.

5.37.3 Member Function Documentation

5.37.3.1 void Statistics::operator()() [virtual]

Start the service/module in current thread.

Implements [nfrd::module::Module](#).

5.37.3.2 void Statistics::Stop() [virtual]

Stop the service/module, joining the thread.

Reimplemented from [nfrd::module::Module](#).

5.37.4 Member Data Documentation

5.37.4.1 int nfrd::module::Statistics::period [private]

Period of updating or recording (in seconds)

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

- include/nfrd/[Statistics.h](#)
- src/[Statistics.cpp](#)

Chapter 6

File Documentation

6.1 docs/config.dox File Reference

6.2 docs/namespace.dox File Reference

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::config](#)
Contains classes related to configuration.
- namespace [nfrd::module](#)
Contains classes that control different modules.
- namespace [nfrd::log](#)
Contains classes related to log.
- namespace [nfrd::misc](#)
Contains classes that commonly used in other classes.
- namespace [nfrd::parser](#)
Contains classes related to all kinds of feed parsers.
- namespace [nfrd::misc::Utility](#)
Contains all utility functions.

6.3 docs/protocol.dox File Reference

6.4 docs/usage.dox File Reference

6.5 include/nfrd/AdminService.h File Reference

```
#include <string>
#include <set>
#include <boost/asio.hpp>
#include <boost/thread.hpp>
#include <boost/thread/condition.hpp>
#include "nfrd/Module.h"
#include "nfrd/AdminServiceThread.h"
```

Classes

- class [nfrd::module::AdminService](#)
Manage sockets talk to the front end and interacts with other components.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::module](#)
Contains classes that control different modules.

6.5.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

30/04/12

6.5.2 DESCRIPTION

A class to manage sockets talk to the front end and interacts with other components.

See also

[Protocol talks to Front-end](#)

6.6 include/nfrd/AdminServiceThread.h File Reference

```
#include <string>
#include <vector>
#include <boost/asio.hpp>
#include "nfrd/Module.h"
```

Classes

- class [nfrd::module::AdminServiceThread](#)
Handle sockets talk to the front end and interacts with other components.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::module](#)
Contains classes that control different modules.

6.6.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

01/05/12

6.6.2 DESCRIPTION

A class to handle front-end requests.

See also

AdminService

6.7 include/nfrd/AtomParser.h File Reference

```
#include "nfrd/FeedParser.h"
```

Classes

- class [nfrd::parser::AtomItem](#)
A class to store details of an item obtained by the [AtomParser](#).
- class [nfrd::parser::AtomParser](#)
A parser to parse Atom feeds.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::parser](#)
Contains classes related to all kinds of feed parsers.

6.7.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

22/08/12

6.7.2 DESCRIPTION

Classes related to parsing Atom feeds.

AtomItem is a part of AtomParser (as a container).

AtomParser is a class that parse Atom feeds.

See also

[Parser.h](#)

6.8 include/nfrd/AutoDB.h File Reference

Classes

- struct [nfrd::misc::AutoDBRef<_Tp1>](#)
A wrapper class to provide [AutoDB](#) with reference semantics.
- class [nfrd::misc::AutoDB<_Tp>](#)
A class to mimic `std::AutoDB` specified for `nfdb` usage: `std::AutoDB(std::vector<int>);` When out of scope, this class will deallocate the `int*` in the container automatically.*

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::misc](#)
Contains classes that commonly used in other classes.

6.8.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

14/09/12

6.8.2 DESCRIPTION

Amended `std::auto_ptr` specified for `nfdb` This file is created by modifying `<memory>` of the Standard C++ Library

6.9 include/nfrd/ConfigManager.h File Reference

```
#include <string>
#include <map>
#include <iostream>
```

Classes

- class [nfrd::config::ConfigSector](#)
A part of [ConfigManager](#) (as a container)
- class [nfrd::config::ConfigManager](#)
Manages config files (core class).
- class [nfrd::config::ConfigException](#)
General exception for config.

- class [nfrd::config::IOException](#)
Input/Output exception for config.
- class [nfrd::config::ItemNotFound](#)
Item not found.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::config](#)
Contains classes related to configuration.

6.9.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

01/05/12

6.9.2 DESCRIPTION

ConfigManager that manages config files.

ConfigSector is a part of ConfigManager (as a container).

ConfigException is for general exceptions.

IOException is for Input/Output exceptions.

ItemNotFound is for Item not found exceptions.

6.10 include/nfrd/Crawler.h File Reference

```
#include <list>
#include <queue>
#include <vector>
#include <mysql_connection.h>
#include <cppconn/resultset.h>
#include <cppconn/prepared_statement.h>
#include <boost/thread.hpp>
#include "nfrd/Module.h"
#include "nfrd/CrawlerThread.h"
#include "nfrd/QueueItem.h"
#include "nfrd/FeedPriorityQueue.h"
```

Classes

- class [nfrd::module::Crawler](#)
The main class representing the crawler module The responsibilities of this class:

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::module](#)
Contains classes that control different modules.

6.10.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au
Aaron

Date

21/08/12

6.10.2 DESCRIPTION

A class to crawl feeds from the source obtained from the database and insert the data back to the database.

Warning

This is a prototype file (Originally MySQLUpdater)

6.11 include/nfrd/CrawlerThread.h File Reference

```
#include <list>
#include <mysql_connection.h>
#include <cppconn/resultset.h>
#include <cppconn/prepared_statement.h>
#include <boost/thread.hpp>
#include "nfrd/QueueItem.h"
#include "nfrd/LogManager.h"
#include "nfrd/ConfigManager.h"
#include "nfrd/FeedPriorityQueue.h"
#include <libnfdb/FeedController.h>
#include <libnfdb/Feed.h>
```

Classes

- class [nfrd::module::CrawlerThread](#)
A worker class representing a thread.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::module](#)
Contains classes that control different modules.

6.11.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au,
Aron Hardy-Bardsley ajrhb957@uowmail.edu.au

Date

15/08/12

6.11.2 DESCRIPTION

Crawler thread polls the queue (crawler) for work and executes the parser

6.12 include/nfrd/DateTime.h File Reference

```
#include <string>
```

Classes

- class [nfrd::misc::DateTime](#)
A class to store date and time.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::misc](#)
Contains classes that commonly used in other classes.

6.12.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

09/08/12

6.12.2 DESCRIPTION

A class to store date and time

6.13 include/nfrd/FeedParser.h File Reference

```
#include <rapidxml.hpp>  
#include "nfrd/Parser.h"
```

Classes

- class [nfrd::parser::FeedItem](#)
A class to store details of an item obtained by the [FeedParser](#).
- class [nfrd::parser::FeedParser](#)
A parser to parse web feeds.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::parser](#)
Contains classes related to all kinds of feed parsers.

6.13.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

14/09/12

6.13.2 DESCRIPTION

Classes related to parsing web feeds with Patch and Match feature.

FeedItem is a part of FeedParser (as a container).

FeedParser is a class that parse web feeds.

See also

[Parser.h](#)

6.14 include/nfrd/FeedPriorityQueue.h File Reference

```
#include <list>
#include <queue>
#include <vector>
#include <boost/thread.hpp>
#include "nfrd/LogManager.h"
#include "nfrd/ConfigManager.h"
#include "nfrd/QueueItem.h"
```

Classes

- class [nfrd::module::FeedPriorityQueue](#)
Implements a queueing/threading model.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::module](#)
Contains classes that control different modules.

6.14.1 Detailed Description

Author

Aron Hardy-Bardsley ajrhb957@uowmail.edu.au

Date

15/08/12

6.14.2 DESCRIPTION

The FeedPriorityQueue class is the implementation of the queueing model.

It handles feed priorities and delegates work (which is then

consumed by some other entity for processing

6.15 include/nfrd/FeedXParser.h File Reference

```
#include <set>
#include <string>
#include "htmlcxx/html/ParserDom.h"
#include "nfrd/Parser.h"
#include "nfrd/FeedParser.h"
```

Classes

- class [nfrd::parser::FeedXParser](#)
A parser to parse web feeds with Patch and Match feature.
- struct [nfrd::parser::FeedXParser::IteratorPair](#)
A iterator pair structure used in [Construct\(\)](#) to store iterators for parent nodes.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::parser](#)
Contains classes related to all kinds of feed parsers.

6.15.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

14/09/12

6.15.2 DESCRIPTION

Classes related to parsing web feeds with Patch and Match feature.

See also

[Parser.h](#)

[FeedParser.h](#)

6.16 include/nfrd/Image.h File Reference

```
#include <string>
#include <vector>
#include <memory>
#include <gd.h>
```

Classes

- class [nfrd::misc::Image](#)
A class to store image and process image.
- class [nfrd::misc::Image::File](#)
A class to represent an image file (as an owner)
- class [nfrd::misc::ImageException](#)
General exception for [Image](#).

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::misc](#)
Contains classes that commonly used in other classes.

6.16.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

24/08/12

6.16.2 DESCRIPTION

A class to store image and process image

6.17 include/nfrd/LogManager.h File Reference

```
#include <string>
#include <fstream>
#include <iostream>
#include <boost/thread.hpp>
```

Classes

- class [nfrd::log::LogManager](#)
Manage logs.
- class [nfrd::log::LogException](#)
General exception for log.
- class [nfrd::log::IOException](#)
Input/Output exception for config.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::log](#)
Contains classes related to log.

6.17.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

11/04/12

6.17.2 DESCRIPTION

A class to manage logs.

6.18 include/nfrd/Master.h File Reference

```
#include <string>
#include <queue>
#include <utility>
#include <ctime>
#include <map>
#include <boost/thread.hpp>
#include <boost/thread/condition.hpp>
#include "nfrd/ConfigManager.h"
#include "nfrd/LogManager.h"
#include "nfrd/Module.h"
```

Classes

- class [nfrd::Master](#)
Manage, contain and access all components of nfrd.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.

6.18.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

01/05/12

6.18.2 DESCRIPTION

A class to manage, contain and access all components of nfrd.

6.19 include/nfrd/Module.h File Reference

```
#include <string>
#include <boost/thread.hpp>
#include "nfrd/ConfigManager.h"
#include "nfrd/LogManager.h"
```

Classes

- class [nfrd::module::Module](#)
A generalised module interface class, providing all the interfaces of a module that start or stop.
- class [nfrd::module::ModuleException](#)
General exception for module.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::module](#)
Contains classes that control different modules.

6.19.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

30/04/12

6.19.2 DESCRIPTION

Abstract/Interface Controller class and related stuff.

ControllerException is for general exceptions.

6.20 include/nfrd/Parser.h File Reference

```
#include <string>
#include <list>
#include <utility>
#include "nfrd/DateTime.h"
#include "nfrd/Image.h"
```

Classes

- class [nfrd::parser::Item](#)
A class to store details of an item obtained by the [Parser](#).
- class [nfrd::parser::Parser](#)
A generalised parser interface class, providing all the interfaces of a class that reads resource from an URL and parse it into a list of [Item](#).
- class [nfrd::parser::ParserException](#)
General exception for parser.
- class [nfrd::parser::HasNoValue](#)
Has no value exception.
- class [nfrd::parser::InvalidSource](#)
Invalid source exception.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::parser](#)
Contains classes related to all kinds of feed parsers.

6.20.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

24/08/12

6.20.2 DESCRIPTION

Abstract/Interface Parser class and related stuff.

Item is a part of Parser (as a container).

Parser is a generalised parser reading source form an URL.

ParserException is for general exceptions.

HasNoValue is thrown when the attribute has no value.

InvalidSource is thrown when the resource cannot be parsed by the parser.

6.21 include/nfrd/QueueItem.h File Reference

```
#include <libnfrdb/QueueItem.h>
```

Classes

- class [nfrd::module::QueueItem](#)

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::module](#)
Contains classes that control different modules.

6.21.1 Detailed Description

Author

Aron Hardy-Bardsley ajrhb957@uow.edu.au

Date

15/08/12

6.21.2 DESCRIPTION

Represents a compact version of a feed for use with the queueing model
(implemented by crawler)

6.22 include/nfrd/RSSParser.h File Reference

```
#include "nfrd/FeedParser.h"
```

Classes

- class [nfrd::parser::RSSItem](#)
A class to store details of an item obtained by the [RSSParser](#).
- class [nfrd::parser::RSSParser](#)
A parser to parse RSS feeds.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::parser](#)
Contains classes related to all kinds of feed parsers.

6.22.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

22/08/12

6.22.2 DESCRIPTION

Classes related to parsing RSS feeds.

RSSItem is a part of RSSParser (as a container).

RSSParser is a class that parse RSS feeds.

See also

[Parser.h](#)

6.23 include/nfrd/Statistics.h File Reference

```
#include "nfrd/Module.h"
```

Classes

- class [nfrd::module::Statistics](#)
Periodly record and update the statistics information.

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::module](#)
Contains classes that control different modules.

6.23.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

18/10/12

6.23.2 DESCRIPTION

A class to periodically record and update the statistics information.

6.24 include/nfrd/Utility.h File Reference

```
#include <vector>
#include <string>
#include <istream>
#include <memory>
#include <stdexcept>
```

Namespaces

- namespace [nfrd](#)
Contains all classes unique to News Feeder Refresh Daemon.
- namespace [nfrd::misc](#)
Contains classes that commonly used in other classes.
- namespace [nfrd::misc::Utility](#)
Contains all utility functions.

Functions

- bool [nfrd::misc::Utility::Read](#) (const char *url, std::vector< char > &container)
Read the content of the url and write to the container.
- bool [nfrd::misc::Utility::Read](#) (const std::string &url, std::vector< char > &container)
This function overloads and calls bool [nfrd::misc::Utility::Read](#)(const char url, std::vector<char> & container).*
- std::auto_ptr< std::vector< char > > [nfrd::misc::Utility::Read](#) (const std::string &url)
This function overloads and calls bool [nfrd::misc::Utility::Read](#)(const char url, std::vector<char> & container).*
- int [nfrd::misc::Utility::ToInt](#) (const std::string &str)
String to integer.
- bool [nfrd::misc::Utility::ToBool](#) (const std::string &str)

String to boolean.

- void `nfrd::misc::Utility::GetArguments` (std::istream &in, std::vector< std::string > &args)

Get arguments from a line of a stream.

- std::string `nfrd::misc::Utility::Trim` (const std::string &str)

Trim the trailing and ending whitespaces and return a new string.

6.24.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

24/08/12

6.24.2 DESCRIPTION

A namespace contains all utility functions

6.25 src/AdminService.cpp File Reference

```
#include "nfrd/AdminService.h"
#include "nfrd/Utility.h"
```

6.25.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

30/04/12

6.25.2 DESCRIPTION

Implementation of AdminService and associated stuffs

6.26 src/AdminServiceThread.cpp File Reference

```
#include <memory>
#include <ctime>
#include "nfrd/AdminServiceThread.h"
#include "nfrd/AdminService.h"
#include "nfrd/Master.h"
#include "nfrd/Utility.h"
```

6.26.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

20/09/12

6.26.2 DESCRIPTION

Implementation of AdminServiceThread

6.27 src/AtomParser.cpp File Reference

```
#include <cstring>
#include "nfrd/AtomParser.h"
#include "nfrd/Utility.h"
```

6.27.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

09/08/12

6.27.2 DESCRIPTION

Implementation of Atom Parser

6.28 src/ConfigManager.cpp File Reference

```
#include <cctype>
#include <iostream>
#include <fstream>
#include <cstring>
#include <errno.h>
#include "nfrd/ConfigManager.h"
```

Variables

- const string [WHITESPACES](#) = "\t\f\v\n\r"
Define whitespace character string.

6.28.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

01/05/12

6.28.2 DESCRIPTION

Implementation of ConfigManager and associated stuffs

6.28.3 Variable Documentation

6.28.3.1 `const string WHITESPACES = "\t\f\v\n\r"`

Define whitespace character string.

This variable is to indicate whitespace characters when parsing config file.

See also

[config::ConfigSector::Read\(\)](#)

6.29 src/Crawler.cpp File Reference

```
#include <string>
#include <cppconn/exception.h>
#include <cppconn/driver.h>
#include "nfrd/Crawler.h"
#include "nfrd/RSSParser.h"
#include "nfrd/FeedPriorityQueue.h"
#include <libnfd/FeedController.h>
#include <libnfd/Feed.h>
```

6.29.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au
Aron Hardy-Bardsley ajrhb957@uowmail.edu.au

Date

06/09/12

6.29.2 DESCRIPTION

Implementation of Crawler

6.30 src/CrawlerThread.cpp File Reference

```
#include <string>
#include <memory>
#include <cppconn/exception.h>
#include <cppconn/driver.h>
#include "nfrd/CrawlerThread.h"
#include "nfrd/FeedXParser.h"
#include "nfrd/LogManager.h"
#include "nfrd/ConfigManager.h"
#include "nfrd/FeedPriorityQueue.h"
#include "nfrd/AutoDB.h"
#include <libnfdb/FeedController.h>
#include <libnfdb/ItemController.h>
#include <libnfdb/ImageController.h>
#include <libnfdb/NotificationController.h>
#include <libnfdb/Feed.h>
```

6.30.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au,
Aron Hardy-Bardsley ajrhb957@uowmail.edu.au

Date

20/09/12

6.30.2 DESCRIPTION

Implementation of Crawler Thread

6.31 src/DateTime.cpp File Reference

```
#include <sstream>
#include <iomanip>
#include <cctype>
#include <cstdlib>
#include <stdexcept>
#include "nfrd/DateTime.h"
```

6.31.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

09/08/12

6.31.2 DESCRIPTION

Implementation of DateTime class

6.32 src/FeedParser.cpp File Reference

```
#include <cstring>
#include "nfrd/FeedParser.h"
#include "nfrd/Utility.h"
```

6.32.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

20/09/12

6.32.2 DESCRIPTION

Implementation of FeedItem and FeedParser

6.33 src/FeedPriorityQueue.cpp File Reference

```
#include "nfrd/FeedPriorityQueue.h"
#include <vector>
#include <libnfrdb/UserController.h>
```

6.33.1 Detailed Description

Author

Aron Hardy-Bardsley ajrhhb957@uowmail.edu.au

Date

17/08/12

6.33.2 DESCRIPTION

Implementation of the FeedPriorityQueue class.

6.34 src/FeedXParser.cpp File Reference

```
#include <cstring>
#include <utility>
#include <stack>
#include <boost/algorithm/string/predicate.hpp>
#include <boost/algorithm/string/case_conv.hpp>
#include "nfrd/FeedXParser.h"
#include "nfrd/RSSParser.h"
#include "nfrd/AtomParser.h"
#include "nfrd/Utility.h"
```

Variables

- static const char * [ALLOWED_TAGS](#) []
The tags that will not be changed.
- static const char * [TRIMMED_TAGS](#) []
The tag that only text part is kept.
- static const string::size_type [SIZE_MODIFIER](#) = 3
The right paragraph should be SIZE_MODIFIER times larger than the abstract article in the rss item when processing patch and match algorithm.

6.34.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

20/09/12

6.34.2 DESCRIPTION

Implementation of RSS X Parser

6.34.3 Variable Documentation

6.34.3.1 const char* [ALLOWED_TAGS](#)[] [static]

Initial value:

```
{
    "b",    "i",    "p",    "a",    "strong",    "u",    "br",
    "strike", "s",    "q",    "ul",    "li",    "pre"
}
```

The tags that will not be changed.

See also

`void FeedXParser::BuildStaticMember()`

6.34.3.2 `const string::size_type SIZE_MODIFIER = 3` `[static]`

The right paragraph should be SIZE_MODIFIER times larger than the abstract article in the rss item when processing patch and match algorithm.

See also

```
void FeedXParser::PatchAndMatch(RSSItem& item)
```

6.34.3.3 `const char* TRIMMED_TAGS[]` `[static]`

Initial value:

```
{
    "html", "body", "div", "h1", "h2", "h3", "h4", "h5", "h6",
    "small", "font", "center", "table", "tr", "td",
    "img", "span", "",
}
```

The tage that only text part is kept.

See also

```
void FeedXParser::BuildStaticMember()
```

6.35 src/Image.cpp File Reference

```
#include <cstring>
#include "nfrd/Image.h"
#include "nfrd/Utility.h"
```

6.35.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

24/08/12

6.35.2 DESCRIPTION

Implementation of Image class

6.36 src/LogManager.cpp File Reference

```
#include <cstring>
#include <errno.h>
#include <ctime>
#include "nfrd/LogManager.h"
```

6.36.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

30/04/12

6.36.2 DESCRIPTION

Implementation of LogManager and associated stuffs

6.37 src/Master.cpp File Reference

```
#include "nfrd/Master.h"  
#include "nfrd/Utility.h"  
#include "nfrd/AdminService.h"  
#include "nfrd/Crawler.h"  
#include "nfrd/Statistics.h"
```

6.37.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

01/05/12

6.37.2 DESCRIPTION

Implementation of Master and associated stuffs

6.38 src/Module.cpp File Reference

```
#include "nfrd/Module.h"
```

6.38.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

30/04/12

6.38.2 DESCRIPTION

Part implementation of Abstract/Interface Module class and related stuff

6.39 src/nfrd.cpp File Reference

```
#include <iostream>
#include <signal.h>
#include <unistd.h>
#include <errno.h>
#include "nfrd/Master.h"
```

Functions

- void [Terminate](#) (int signo)
A signal handler to safely terminate this program.
- void [PrintUsage](#) (const char prog[])
Print the usage of nfrd.
- int [main](#) (int argc, char *argv[])
Program entrance that initialises [nfrd::Master](#) and run it.

Variables

- static [nfrd::Master](#) * [tracker](#) = 0
Global tracking on nfrd master instance.

6.39.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

30/04/12

6.39.2 DESCRIPTION

Program entrance of News Feeder Refresh Daemon(nfrd). The main function will calls interface of [nfrd::Master](#)

6.39.3 Function Documentation

6.39.3.1 int main (int argc, char * argv[])

Program entrance that initialises [nfrd::Master](#) and run it.

Parameters

<i>argc</i>	count number of arguments taken from command line
<i>argv</i>	value of arguments taken from command line

Returns

0 if program terminated without error

6.39.3.2 void PrintUsage (const char *prog*[])

Print the usage of nfrd.

Parameters

<i>prog</i>	the executable name of the program
-------------	------------------------------------

6.39.3.3 void Terminate (int *signo*)

A signal handler to safely terminate this program.

Parameters

<i>signo</i>	signal number
--------------	---------------

6.39.4 Variable Documentation**6.39.4.1 nfrd::Master* tracker = 0 [static]**

Global tracking on nfrd master instance.

6.40 src/Parser.cpp File Reference

```
#include "nfrd/Parser.h"
```

6.40.1 Detailed Description**Author**

Shiwei Zhang sz653@uow.edu.au

Date

24/08/12

6.40.2 DESCRIPTION

Part implementation of Abstract/Interface Parser class and related stuff

6.41 src/QueueItem.cpp File Reference

```
#include "nfrd/QueueItem.h"
```

6.41.1 Detailed Description

Author

Aron Hardy-Bardsley ajrhb957@uowmail.edu.au

Date

16/08/12

6.41.2 DESCRIPTION

A light weight object for representing a feed in the queue

6.42 src/RSSParser.cpp File Reference

```
#include <cstring>
#include "nfrd/RSSParser.h"
#include "nfrd/Utility.h"
```

6.42.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

22/08/12

6.42.2 DESCRIPTION

Implementation of RSS Parser

6.43 src/Statistics.cpp File Reference

```
#include <libnfdb/StatController.h>
#include "nfrd/Statistics.h"
#include "nfrd/Utility.h"
```

6.43.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

18/10/12

6.43.2 DESCRIPTION

Implementation of Statistics

6.44 src/Utility.cpp File Reference

```
#include <curl/curl.h>
#include <cstdlib>
#include "nfrd/Utility.h"
```

Functions

- static `size_t _write_data` (`void *ptr`, `size_t size`, `size_t nmemb`, `void *stream`)
Callback function used by bool `nfrd::misc::Utility::Read`(`const char url`, `std::vector<char>& container`).*

Variables

- static const string `WHITESPACES` = "`\t\f\v\n\r`"
Define whitespace character string.

6.44.1 Detailed Description

Author

Shiwei Zhang sz653@uow.edu.au

Date

24/08/12

6.44.2 DESCRIPTION

Implementation of all utility functions.

6.44.3 Function Documentation

6.44.3.1 `static size_t _write_data (void * ptr, size_t size, size_t nmemb, void * stream)` `[static]`

Callback function used by bool `nfrd::misc::Utility::Read`(`const char* url`, `std::vector<char>& container`).

This is a callback function for cURL easy interface API. The functionality is to write bytes obtain from the url to the container. This function is static, meaning that this function is only accessible in this file.

Parameters

<i>ptr</i>	data pointer
<i>size</i>	size of a memory block
<i>nmemb</i>	number of memory block
<i>stream</i>	container/stream the function writes data to

Returns

number of data successfully writed to the container/stream

See also

bool [nfrd::misc::Utility::Read](#)(const char* url, std::vector<char>& container)

6.44.4 Variable Documentation

6.44.4.1 `const string WHITESPACES = "\t\f\v\n\r"` `[static]`

Define whitespace character string.

This variable is to indicate whitespace characters when parsing config file.

See also

std::string [nfrd::misc::Utility::Trim](#)(const std::string& str)

Index

- ~AdminService
 - nfrd::module::AdminService, [24](#)
- ~AdminServiceThread
 - nfrd::module::AdminServiceThread, [27](#)
- ~AtomItem
 - nfrd::parser::AtomItem, [31](#)
- ~AtomParser
 - nfrd::parser::AtomParser, [36](#)
- ~AutoDB
 - nfrd::misc::AutoDB, [39](#)
- ~ConfigException
 - nfrd::config::ConfigException, [43](#)
- ~ConfigManager
 - nfrd::config::ConfigManager, [45](#)
- ~ConfigSector
 - nfrd::config::ConfigSector, [48](#)
- ~Crawler
 - nfrd::module::Crawler, [51](#)
- ~CrawlerThread
 - nfrd::module::CrawlerThread, [54](#)
- ~DateTime
 - nfrd::misc::DateTime, [57](#)
- ~FeedItem
 - nfrd::parser::FeedItem, [66](#)
- ~FeedParser
 - nfrd::parser::FeedParser, [69](#)
- ~FeedPriorityQueue
 - nfrd::module::FeedPriorityQueue, [73](#)
- ~FeedXParser
 - nfrd::parser::FeedXParser, [78](#)
- ~File
 - nfrd::misc::Image::File, [82](#)
- ~HasNoValue
 - nfrd::parser::HasNoValue, [85](#)
- ~IOException
 - nfrd::config::IOException, [92](#)
 - nfrd::log::IOException, [93](#)
- ~Image
 - nfrd::misc::Image, [87](#)
- ~ImageException
 - nfrd::misc::ImageException, [90](#)
- ~InvalidSource
 - nfrd::parser::InvalidSource, [92](#)
- ~Item
 - nfrd::parser::Item, [95](#)
- ~ItemNotFound
 - nfrd::config::ItemNotFound, [99](#)
- ~LogException
 - nfrd::log::LogException, [101](#)
- ~LogManager
 - nfrd::log::LogManager, [103](#)
- ~Master
 - nfrd::Master, [106](#)
- ~Module
 - nfrd::module::Module, [111](#)
- ~ModuleException
 - nfrd::module::ModuleException, [113](#)
- ~Parser
 - nfrd::parser::Parser, [114](#)
- ~ParserException
 - nfrd::parser::ParserException, [116](#)
- ~RSSItem
 - nfrd::parser::RSSItem, [121](#)
- ~RSSParser
 - nfrd::parser::RSSParser, [126](#)
- ~Statistics
 - nfrd::module::Statistics, [128](#)
- _M_ptr
 - nfrd::misc::AutoDB, [41](#)
 - nfrd::misc::AutoDBRef, [42](#)
- _write_data
 - Utility.cpp, [158](#)
- ALLOWED_TAGS
 - FeedXParser.cpp, [152](#)
- acceptor
 - nfrd::module::AdminService, [25](#)
- AddImage
 - nfrd::parser::FeedItem, [66](#)
- AdminService
 - nfrd::module::AdminService, [24](#)
- AdminServiceThread
 - nfrd::module::AdminServiceThread, [27](#)
- AdvancedMatchDivide
 - nfrd::parser::FeedXParser, [78](#)
- AdvancedMatchLine
 - nfrd::parser::FeedXParser, [78](#)
- allowed_tags
 - nfrd::parser::FeedXParser, [81](#)
- AtomItem
 - nfrd::parser::AtomItem, [31](#)
- AtomParser
 - nfrd::parser::AtomParser, [36](#)
- Authenticate
 - nfrd::module::AdminService, [24](#)
- author
 - nfrd::parser::AtomItem, [35](#)
 - nfrd::parser::RSSItem, [124](#)
- AutoDB

- nfrd::misc::AutoDB, [39](#)
- AutoDBRef
 - nfrd::misc::AutoDBRef, [42](#)
- begin
 - nfrd::config::ConfigSector, [48](#)
- buffer
 - nfrd::parser::RSSParser, [127](#)
- CalculateConstant
 - nfrd::module::QueueItem, [118](#)
- CleanupQueue
 - nfrd::module::FeedPriorityQueue, [73](#)
- ClearImage
 - nfrd::parser::FeedItem, [66](#)
- condition
 - nfrd::Master, [108](#)
 - nfrd::module::AdminService, [25](#)
- config
 - nfrd::Master, [108](#)
 - nfrd::module::CrawlerThread, [55](#)
 - nfrd::module::FeedPriorityQueue, [74](#)
 - nfrd::module::Module, [112](#)
- config_main
 - nfrd::module::AdminServiceThread, [27](#)
- ConfigException
 - nfrd::config::ConfigException, [43](#)
- configFile
 - nfrd::config::ConfigManager, [46](#)
- ConfigManager
 - nfrd::config::ConfigManager, [44](#)
- ConfigManager.cpp
 - WHITESPACES, [149](#)
- ConfigSector
 - nfrd::config::ConfigSector, [48](#)
- const_iterator
 - nfrd::config::ConfigSector, [48](#)
- Construct
 - nfrd::parser::FeedXParser, [78](#)
- content
 - nfrd::parser::AtomItem, [35](#)
 - nfrd::parser::RSSItem, [125](#)
- contentUpdateAverage
 - nfrd::module::QueueItem, [119](#)
- Crawl
 - nfrd::module::CrawlerThread, [54](#)
- Crawler
 - nfrd::module::Crawler, [51](#)
- crawler_main
 - nfrd::module::AdminServiceThread, [28](#)
- CrawlerThread
 - nfrd::module::CrawlerThread, [54](#)
- crawlers
 - nfrd::module::Crawler, [52](#)
- data
 - nfrd::misc::Image::File, [84](#)
- DateTime
 - nfrd::misc::DateTime, [57](#)
- day
 - nfrd::misc::DateTime, [64](#)
- Disable
 - nfrd::log::LogManager, [103](#)
- docs/config.dox, [131](#)
- docs/namespace.dox, [131](#)
- docs/protocol.dox, [131](#)
- docs/usage.dox, [131](#)
- dynamic
 - nfrd::module::AdminServiceThread, [29](#)
- ERROR
 - nfrd::log::LogManager, [102](#)
- element_type
 - nfrd::misc::AutoDB, [38](#)
- Enable
 - nfrd::log::LogManager, [103](#)
- enabled
 - nfrd::log::LogManager, [104](#)
- end
 - nfrd::config::ConfigSector, [48](#)
 - nfrd::parser::FeedXParser::IteratorPair, [100](#)
- Exit
 - nfrd::module::AdminService, [24](#)
- ExportJpeg
 - nfrd::misc::Image, [87](#)
- ExportPng
 - nfrd::misc::Image, [87](#)
- ExportToMySQL
 - nfrd::misc::DateTime, [57](#)
- ExtractGeoLocation
 - nfrd::parser::FeedXParser, [79](#)
- ExtractImages
 - nfrd::parser::FeedXParser, [79](#)
- feed_comparer
 - nfrd::module::QueueItem, [118](#)
- FeedItem
 - nfrd::parser::FeedItem, [66](#)
- FeedParser
 - nfrd::parser::FeedParser, [69](#)
- FeedPriorityQueue
 - nfrd::module::FeedPriorityQueue, [72](#)
- feedType
 - nfrd::module::QueueItem, [119](#)
- FeedXParser
 - nfrd::parser::FeedXParser, [78](#)
- FeedXParser.cpp
 - ALLOWED_TAGS, [152](#)
 - SIZE_MODIFIER, [152](#)
 - TRIMMED_TAGS, [153](#)
- File
 - nfrd::misc::Image::File, [82](#)
- FitSize
 - nfrd::misc::Image, [87](#)
- fout
 - nfrd::log::LogManager, [104](#)
- full_content
 - nfrd::parser::FeedItem, [68](#)

GD
 nfrd::misc::Image, 86
 GD2
 nfrd::misc::Image, 86
 GIF
 nfrd::misc::Image, 86
 geo_location
 nfrd::parser::FeedItem, 68
 get
 nfrd::misc::AutoDB, 39
 GetAllItems
 nfrd::module::FeedPriorityQueue, 73
 GetArguments
 nfrd::misc::Utility, 18
 GetAuthor
 nfrd::parser::AtomItem, 31
 nfrd::parser::Item, 95
 nfrd::parser::RSSItem, 121
 GetConfig
 nfrd::Master, 106
 GetContent
 nfrd::parser::AtomItem, 32
 nfrd::parser::Item, 95
 nfrd::parser::RSSItem, 121
 GetData
 nfrd::misc::Image::File, 83
 GetDay
 nfrd::misc::DateTime, 58
 GetFeedID
 nfrd::module::QueueItem, 118
 GetGeoLocation
 nfrd::parser::FeedItem, 66
 nfrd::parser::Item, 95
 GetHeight
 nfrd::misc::Image, 88
 GetHour
 nfrd::misc::DateTime, 58
 GetId
 nfrd::module::CrawlerThread, 54
 GetImageList
 nfrd::parser::FeedItem, 66
 nfrd::parser::Item, 96
 GetItemList
 nfrd::parser::FeedParser, 69
 nfrd::parser::FeedXParser, 79
 nfrd::parser::Parser, 115
 GetLastBuildDate
 nfrd::parser::AtomParser, 36
 nfrd::parser::FeedXParser, 79
 nfrd::parser::Parser, 115
 nfrd::parser::RSSParser, 126
 GetLongestText
 nfrd::parser::FeedXParser, 80
 GetMinute
 nfrd::misc::DateTime, 58
 GetModule
 nfrd::Master, 106
 GetMonth
 nfrd::misc::DateTime, 58
 GetName
 nfrd::module::Module, 111
 GetPostDate
 nfrd::parser::AtomItem, 32
 nfrd::parser::Item, 96
 nfrd::parser::RSSItem, 122
 GetSecond
 nfrd::misc::DateTime, 58
 GetSize
 nfrd::misc::Image::File, 83
 GetStartTime
 nfrd::Master, 106
 GetStatus
 nfrd::module::Module, 111
 GetStatusString
 nfrd::module::Module, 111
 GetStream
 nfrd::module::AdminServiceThread, 28
 GetThread
 nfrd::module::FeedPriorityQueue, 73
 nfrd::module::Module, 111
 GetTitle
 nfrd::parser::AtomItem, 32
 nfrd::parser::Item, 96
 nfrd::parser::RSSItem, 122
 GetType
 nfrd::misc::Image::File, 83
 GetURL
 nfrd::parser::AtomItem, 32
 nfrd::parser::Item, 97
 nfrd::parser::RSSItem, 122
 GetValue
 nfrd::parser::RSSParser, 127
 GetVersion
 nfrd::Master, 107
 GetWidth
 nfrd::misc::Image, 88
 GetYear
 nfrd::misc::DateTime, 58
 Handle
 nfrd::module::AdminServiceThread, 28
 HasAuthor
 nfrd::parser::AtomItem, 33
 nfrd::parser::Item, 97
 nfrd::parser::RSSItem, 122
 HasContent
 nfrd::parser::AtomItem, 33
 nfrd::parser::Item, 97
 nfrd::parser::RSSItem, 123
 HasFullContent
 nfrd::parser::FeedItem, 67
 HasGeoLocation
 nfrd::parser::FeedItem, 67
 nfrd::parser::Item, 97
 HasImageList
 nfrd::parser::FeedItem, 67
 nfrd::parser::Item, 97

- HasNoValue
 - nfrd::parser::HasNoValue, 85
- HasPostDate
 - nfrd::parser::AtomItem, 33
 - nfrd::parser::Item, 97
 - nfrd::parser::RSSItem, 123
- HasTitle
 - nfrd::parser::AtomItem, 33
 - nfrd::parser::Item, 98
 - nfrd::parser::RSSItem, 123
- HasURL
 - nfrd::parser::AtomItem, 33
 - nfrd::parser::Item, 98
 - nfrd::parser::RSSItem, 123
- healthMutex
 - nfrd::module::Crawler, 52
- healthThreadCondition
 - nfrd::module::Crawler, 52
- heapMutex
 - nfrd::module::FeedPriorityQueue, 74
- hour
 - nfrd::misc::DateTime, 64
- IOException
 - nfrd::config::IOException, 92
 - nfrd::log::IOException, 93
- id
 - nfrd::module::CrawlerThread, 55
 - nfrd::module::QueueItem, 119
- im
 - nfrd::misc::Image, 90
- Image
 - nfrd::misc::Image, 86
 - nfrd::parser::Item, 95
- image_list
 - nfrd::parser::FeedItem, 68
- ImageException
 - nfrd::misc::ImageException, 90
- ImportFromMySQL
 - nfrd::misc::DateTime, 58
- include/nfrd/AdminService.h, 131
- include/nfrd/AdminServiceThread.h, 132
- include/nfrd/AtomParser.h, 133
- include/nfrd/AutoDB.h, 134
- include/nfrd/ConfigManager.h, 134
- include/nfrd/Crawler.h, 135
- include/nfrd/CrawlerThread.h, 136
- include/nfrd/DateTime.h, 137
- include/nfrd/FeedParser.h, 137
- include/nfrd/FeedPriorityQueue.h, 138
- include/nfrd/FeedXParser.h, 139
- include/nfrd/Image.h, 140
- include/nfrd/LogManager.h, 141
- include/nfrd/Master.h, 142
- include/nfrd/Module.h, 142
- include/nfrd/Parser.h, 143
- include/nfrd/QueueItem.h, 144
- include/nfrd/RSSParser.h, 145
- include/nfrd/Statistics.h, 145
- include/nfrd/Utility.h, 146
- incomingHasChanged
 - nfrd::module::FeedPriorityQueue, 74
- incomingMutex
 - nfrd::module::FeedPriorityQueue, 74
- incomingQueue
 - nfrd::module::FeedPriorityQueue, 75
- IncrementPriority
 - nfrd::module::QueueItem, 118
- Initialise
 - nfrd::module::CrawlerThread, 54
- InitialiseQueue
 - nfrd::module::Crawler, 51
- InvalidSource
 - nfrd::parser::InvalidSource, 91
- io_mutex
 - nfrd::log::LogManager, 104
- is_allowed_tag
 - nfrd::parser::FeedXParser::IteratorPair, 100
- isAlive
 - nfrd::module::Crawler, 52
 - nfrd::module::CrawlerThread, 55
 - nfrd::module::FeedPriorityQueue, 75
- isEnabled
 - nfrd::log::LogManager, 103
- IsOnline
 - nfrd::Master, 107
- it
 - nfrd::parser::FeedXParser::IteratorPair, 100
- item
 - nfrd::parser::FeedParser, 70
- ItemNotFound
 - nfrd::config::ItemNotFound, 99
- itemQueue
 - nfrd::module::FeedPriorityQueue, 75
- itemsPopped
 - nfrd::module::FeedPriorityQueue, 75
- IterateOnce
 - nfrd::module::FeedPriorityQueue, 73
- iterator
 - nfrd::config::ConfigSector, 48
- IteratorPair
 - nfrd::parser::FeedXParser::IteratorPair, 100
- JPEG
 - nfrd::misc::Image, 86
- JustifyDate
 - nfrd::misc::DateTime, 59
- JustifyTime
 - nfrd::misc::DateTime, 59
- lastBuildDate
 - nfrd::parser::AtomParser, 37
 - nfrd::parser::RSSParser, 127
- Load
 - nfrd::misc::Image, 88, 89
- LoadModule
 - nfrd::Master, 107
- LoadModules

- nfrd::Master, 107
- log
 - nfrd::Master, 109
 - nfrd::module::CrawlerThread, 55
 - nfrd::module::FeedPriorityQueue, 75
 - nfrd::module::Module, 112
- LogException
 - nfrd::log::LogException, 101
- LogManager
 - nfrd::log::LogManager, 103
- MAXPOP
 - nfrd::module::FeedPriorityQueue, 75
- Main
 - nfrd::Master, 107
- main
 - nfrd.cpp, 155
- MainLoop
 - nfrd::module::Crawler, 51
 - nfrd::module::FeedPriorityQueue, 73
- Master
 - nfrd::Master, 106
- master
 - nfrd::module::AdminService, 25
 - nfrd::module::AdminServiceThread, 29
- minute
 - nfrd::misc::DateTime, 64
- Module
 - nfrd::module::Module, 111
- module
 - nfrd::Master, 109
- ModuleException
 - nfrd::module::ModuleException, 113
- month
 - nfrd::misc::DateTime, 64
- msg
 - nfrd::config::ConfigException, 43
 - nfrd::log::LogException, 101
 - nfrd::misc::ImageException, 91
 - nfrd::module::ModuleException, 113
 - nfrd::parser::ParserException, 117
- mutex
 - nfrd::Master, 109
- NORMAL
 - nfrd::log::LogManager, 102
- name
 - nfrd::config::ConfigSector, 49
 - nfrd::module::Module, 112
- nfrd, 15
- nfrd.cpp
 - main, 155
 - PrintUsage, 156
 - Terminate, 156
 - tracker, 156
- nfrd::log::LogManager
 - ERROR, 102
 - NORMAL, 102
 - WARNING, 103
- nfrd::misc::Image
 - GD, 86
 - GD2, 86
 - GIF, 86
 - JPEG, 86
 - PNG, 86
 - UNKNOWN, 86
 - WBMP, 86
- nfrd::module::Module
 - RUNNING, 110
 - STARTING, 110
 - STOPPED, 110
 - STOPPING, 110
- nfrd::Master, 104
 - ~Master, 106
 - condition, 108
 - config, 108
 - GetConfig, 106
 - GetModule, 106
 - GetStartTime, 106
 - GetVersion, 107
 - IsOnline, 107
 - LoadModule, 107
 - LoadModules, 107
 - log, 109
 - Main, 107
 - Master, 106
 - module, 109
 - mutex, 109
 - online, 109
 - SetModule, 108
 - start_time, 109
 - Task, 106
 - task_queue, 109
 - Terminate, 108
 - UnloadModule, 108
 - UnloadModules, 108
- nfrd::config, 15
- nfrd::config::ConfigException, 42
 - ~ConfigException, 43
 - ConfigException, 43
 - msg, 43
 - what, 43
- nfrd::config::ConfigManager, 43
 - ~ConfigManager, 45
 - configFile, 46
 - ConfigManager, 44
 - Read, 45
 - SetFileName, 46
 - value, 46
 - Write, 46
- nfrd::config::ConfigSector, 47
 - ~ConfigSector, 48
 - begin, 48
 - ConfigSector, 48
 - const_iterator, 48
 - end, 48
 - iterator, 48

- name, 49
- Read, 49
- value, 50
- Write, 49
- nfrd::config::IOException, 92
 - ~IOException, 92
 - IOException, 92
- nfrd::config::ItemNotFound, 98
 - ~ItemNotFound, 99
 - ItemNotFound, 99
- nfrd::log, 16
- nfrd::log::IOException, 93
 - ~IOException, 93
 - IOException, 93
- nfrd::log::LogException, 100
 - ~LogException, 101
 - LogException, 101
 - msg, 101
 - what, 101
- nfrd::log::LogManager, 102
 - ~LogManager, 103
 - Disable, 103
 - Enable, 103
 - enabled, 104
 - fout, 104
 - io_mutex, 104
 - isEnabled, 103
 - LogManager, 103
 - operator(), 103, 104
 - Type, 102
- nfrd::misc, 16
- nfrd::misc::AutoDB
 - ~AutoDB, 39
 - _M_ptr, 41
 - AutoDB, 39
 - element_type, 38
 - get, 39
 - operator AutoDB< _Tp1 >, 40
 - operator AutoDBRef< _Tp1 >, 40
 - operator*, 40
 - operator->, 40
 - operator=, 40, 41
 - release, 41
 - reset, 41
- nfrd::misc::AutoDB< _Tp >, 37
- nfrd::misc::AutoDBRef
 - _M_ptr, 42
 - AutoDBRef, 42
- nfrd::misc::AutoDBRef< _Tp1 >, 41
- nfrd::misc::DateTime, 55
 - ~DateTime, 57
 - DateTime, 57
 - day, 64
 - ExportToMySQL, 57
 - GetDay, 58
 - GetHour, 58
 - GetMinute, 58
 - GetMonth, 58
 - GetSecond, 58
 - GetYear, 58
 - hour, 64
 - ImportFromMySQL, 58
 - JustifyDate, 59
 - JustifyTime, 59
 - minute, 64
 - month, 64
 - operator<, 59
 - operator>, 60
 - operator==, 59
 - ParseFromRFC3339, 60
 - ParseFromRFC822, 60
 - ParseFromString, 61
 - second, 64
 - Set, 61
 - SetDateOffset, 61
 - SetDay, 62
 - SetHour, 62
 - SetMinute, 62
 - SetMonth, 63
 - SetSecond, 63
 - SetTimeOffset, 63
 - SetYear, 63
 - year, 64
- nfrd::misc::Image, 85
 - ~Image, 87
 - ExportJpeg, 87
 - ExportPng, 87
 - FitSize, 87
 - GetHeight, 88
 - GetWidth, 88
 - im, 90
 - Image, 86
 - Load, 88, 89
 - operator=, 89
 - Type, 86
- nfrd::misc::Image::File, 81
 - ~File, 82
 - data, 84
 - File, 82
 - GetData, 83
 - GetSize, 83
 - GetType, 83
 - operator=, 83
 - size, 84
 - type, 84
- nfrd::misc::ImageException, 90
 - ~ImageException, 90
 - ImageException, 90
 - msg, 91
 - what, 90
- nfrd::misc::Utility, 17
 - GetArguments, 18
 - Read, 18
 - ToBool, 19
 - ToInt, 19
 - Trim, 19

- nfrd::module, 19
- nfrd::module::AdminService, 23
 - ~AdminService, 24
 - acceptor, 25
 - AdminService, 24
 - Authenticate, 24
 - condition, 25
 - Exit, 24
 - master, 25
 - operator(), 25
 - password, 25
 - Register, 25
 - Stop, 25
 - thread_mutex, 25
 - threads, 25
 - username, 25
- nfrd::module::AdminServiceThread, 26
 - ~AdminServiceThread, 27
 - AdminServiceThread, 27
 - config_main, 27
 - crawler_main, 28
 - dynamic, 29
 - GetStream, 28
 - Handle, 28
 - master, 29
 - operator(), 28
 - parent, 29
 - remote_address, 29
 - shutdown_main, 28
 - sin, 28
 - sout, 28
 - status_main, 29
 - status_send_module_status, 29
 - stream, 29
 - timeout, 29
- nfrd::module::Crawler, 50
 - ~Crawler, 51
 - Crawler, 51
 - crawlers, 52
 - healthMutex, 52
 - healthThreadCondition, 52
 - InitialiseQueue, 51
 - isAlive, 52
 - MainLoop, 51
 - operator(), 52
 - PersistQueue, 52
 - priorityQueue, 52
 - RunMaintenanceTasks, 52
 - StartThreads, 52
 - Stop, 52
 - threads, 52
 - WAITTIME, 53
- nfrd::module::CrawlerThread, 53
 - ~CrawlerThread, 54
 - config, 55
 - Crawl, 54
 - CrawlerThread, 54
 - GetId, 54
 - id, 55
 - Initialise, 54
 - isAlive, 55
 - log, 55
 - priorityQueue, 55
 - Request, 54
 - Stop, 54
 - UpdateItem, 54
- nfrd::module::FeedPriorityQueue, 70
 - ~FeedPriorityQueue, 73
 - CleanupQueue, 73
 - config, 74
 - FeedPriorityQueue, 72
 - GetAllItems, 73
 - GetThread, 73
 - heapMutex, 74
 - incomingHasChanged, 74
 - incomingMutex, 74
 - incomingQueue, 75
 - isAlive, 75
 - itemQueue, 75
 - itemsPopped, 75
 - IterateOnce, 73
 - log, 75
 - MAXPOP, 75
 - MainLoop, 73
 - numberOfUsersInSystem, 75
 - outgoingHasChanged, 75
 - outgoingMutex, 75
 - outgoingQueue, 75
 - outgoingQueueCondition, 76
 - PopFeed, 73
 - priorityQueueThread, 76
 - PushFeed, 74
 - queueUpdateCondition, 76
 - SetNumberOfUsersInSystem, 74
 - Start, 74
 - StartThreads, 74
 - Stop, 74
 - tempOutgoingQueue, 76
 - updatedMutex, 76
- nfrd::module::Module, 109
 - ~Module, 111
 - config, 112
 - GetName, 111
 - GetStatus, 111
 - GetStatusString, 111
 - GetThread, 111
 - log, 112
 - Module, 111
 - name, 112
 - operator(), 112
 - Start, 112
 - Status, 110
 - status, 112
 - Stop, 112
 - thread, 112
- nfrd::module::ModuleException, 113

- ~ModuleException, 113
- ModuleException, 113
- msg, 113
- what, 113
- nfrd::module::QueueItem, 117
 - CalculateConstant, 118
 - contentUpdateAverage, 119
 - feed_comparer, 118
 - feedType, 119
 - GetFeedID, 118
 - id, 119
 - IncrementPriority, 118
 - priority, 119
 - priorityConstant, 119
 - QueueItem, 118
 - ResetPriority, 118
 - SetContentUpdateAverage, 118
 - SetFeedType, 119
 - SetUsersAffected, 119
 - timeSpentInQueue, 119
 - usersAffected, 119
- nfrd::module::Statistics, 128
 - ~Statistics, 128
 - operator(), 129
 - period, 129
 - Statistics, 128
 - Stop, 129
- nfrd::parser, 20
- nfrd::parser::AtomItem, 30
 - ~AtomItem, 31
 - AtomItem, 31
 - author, 35
 - content, 35
 - GetAuthor, 31
 - GetContent, 32
 - GetPostDate, 32
 - GetTitle, 32
 - GetURL, 32
 - HasAuthor, 33
 - HasContent, 33
 - HasPostDate, 33
 - HasTitle, 33
 - HasURL, 33
 - postDate, 35
 - SetAuthor, 34
 - SetContent, 34
 - SetPostDate, 34
 - SetTitle, 34
 - SetURL, 35
 - title, 35
 - url, 35
- nfrd::parser::AtomParser, 35
 - ~AtomParser, 36
 - AtomParser, 36
 - GetLastBuildDate, 36
 - lastBuildDate, 37
 - ReadDom, 37
- nfrd::parser::FeedItem, 64
 - ~FeedItem, 66
 - AddImage, 66
 - ClearImage, 66
 - FeedItem, 66
 - full_content, 68
 - geo_location, 68
 - GetGeoLocation, 66
 - GetImageList, 66
 - HasFullContent, 67
 - HasGeoLocation, 67
 - HasImageList, 67
 - image_list, 68
 - RemoveImage, 67
 - SetContent, 67
 - SetFullContent, 67
 - SetGeoLocation, 68
- nfrd::parser::FeedParser, 68
 - ~FeedParser, 69
 - FeedParser, 69
 - GetItemList, 69
 - item, 70
 - ReadDom, 70
 - ReadURL, 70
 - url, 70
- nfrd::parser::FeedXPather, 76
 - ~FeedXPather, 78
 - AdvancedMatchDivide, 78
 - AdvancedMatchLine, 78
 - allowed_tags, 81
 - Construct, 78
 - ExtractGeoLocation, 79
 - ExtractImages, 79
 - FeedXPather, 78
 - GetItemList, 79
 - GetLastBuildDate, 79
 - GetLongestText, 80
 - parser, 81
 - PatchAndMatch, 80
 - ReadURL, 80
 - RefineFeed, 80
 - TextSize, 80
 - TrimEmptyTag, 81
 - trimmed_tags, 81
- nfrd::parser::FeedXPather::IteratorPair, 99
 - end, 100
 - is_allowed_tag, 100
 - it, 100
 - IteratorPair, 100
- nfrd::parser::HasNoValue, 84
 - ~HasNoValue, 85
 - HasNoValue, 85
- nfrd::parser::InvalidSource, 91
 - ~InvalidSource, 92
 - InvalidSource, 91
- nfrd::parser::Item, 93
 - ~Item, 95
 - GetAuthor, 95
 - GetContent, 95

- GetGeoLocation, 95
- GetImageList, 96
- GetPostDate, 96
- GetTitle, 96
- GetURL, 97
- HasAuthor, 97
- HasContent, 97
- HasGeoLocation, 97
- HasImageList, 97
- HasPostDate, 97
- HasTitle, 98
- HasURL, 98
- Image, 95
- nfrd::parser::Parser, 114
 - ~Parser, 114
 - GetItemList, 115
 - GetLastBuildDate, 115
 - ReadURL, 115
- nfrd::parser::ParserException, 115
 - ~ParserException, 116
 - msg, 117
 - ParserException, 116
 - what, 116
- nfrd::parser::RSSItem, 120
 - ~RSSItem, 121
 - author, 124
 - content, 125
 - GetAuthor, 121
 - GetContent, 121
 - GetPostDate, 122
 - GetTitle, 122
 - GetURL, 122
 - HasAuthor, 122
 - HasContent, 123
 - HasPostDate, 123
 - HasTitle, 123
 - HasURL, 123
 - postDate, 125
 - RSSItem, 121
 - SetAuthor, 123
 - SetContent, 124
 - SetPostDate, 124
 - SetTitle, 124
 - SetURL, 124
 - title, 125
 - url, 125
- nfrd::parser::RSSParser, 125
 - ~RSSParser, 126
 - buffer, 127
 - GetLastBuildDate, 126
 - GetValue, 127
 - lastBuildDate, 127
 - RSSParser, 126
 - ReadDom, 127
- numberOfUsersInSystem
 - nfrd::module::FeedPriorityQueue, 75
- online
 - nfrd::Master, 109
- operator AutoDB< _Tp1 >
 - nfrd::misc::AutoDB, 40
- operator AutoDBRef< _Tp1 >
 - nfrd::misc::AutoDB, 40
- operator<
 - nfrd::misc::DateTime, 59
- operator>
 - nfrd::misc::DateTime, 60
- operator*
 - nfrd::misc::AutoDB, 40
- operator()
 - nfrd::log::LogManager, 103, 104
 - nfrd::module::AdminService, 25
 - nfrd::module::AdminServiceThread, 28
 - nfrd::module::Crawler, 52
 - nfrd::module::Module, 112
 - nfrd::module::Statistics, 129
- operator->
 - nfrd::misc::AutoDB, 40
- operator=
 - nfrd::misc::AutoDB, 40, 41
 - nfrd::misc::Image, 89
 - nfrd::misc::Image::File, 83
- operator==
 - nfrd::misc::DateTime, 59
- outgoingHasChanged
 - nfrd::module::FeedPriorityQueue, 75
- outgoingMutex
 - nfrd::module::FeedPriorityQueue, 75
- outgoingQueue
 - nfrd::module::FeedPriorityQueue, 75
- outgoingQueueCondition
 - nfrd::module::FeedPriorityQueue, 76
- PNG
 - nfrd::misc::Image, 86
- parent
 - nfrd::module::AdminServiceThread, 29
- ParseFromRFC3339
 - nfrd::misc::DateTime, 60
- ParseFromRFC822
 - nfrd::misc::DateTime, 60
- ParseFromString
 - nfrd::misc::DateTime, 61
- parser
 - nfrd::parser::FeedXParser, 81
- ParserException
 - nfrd::parser::ParserException, 116
- password
 - nfrd::module::AdminService, 25
- PatchAndMatch
 - nfrd::parser::FeedXParser, 80
- period
 - nfrd::module::Statistics, 129
- PersistQueue
 - nfrd::module::Crawler, 52
- PopFeed
 - nfrd::module::FeedPriorityQueue, 73
- postDate

- nfrd::parser::AtomItem, 35
- nfrd::parser::RSSItem, 125
- PrintUsage
 - nfrd.cpp, 156
- priority
 - nfrd::module::QueueItem, 119
- priorityConstant
 - nfrd::module::QueueItem, 119
- priorityQueue
 - nfrd::module::Crawler, 52
 - nfrd::module::CrawlerThread, 55
- priorityQueueThread
 - nfrd::module::FeedPriorityQueue, 76
- PushFeed
 - nfrd::module::FeedPriorityQueue, 74
- QueueItem
 - nfrd::module::QueueItem, 118
- queueUpdateCondition
 - nfrd::module::FeedPriorityQueue, 76
- RUNNING
 - nfrd::module::Module, 110
- RSSItem
 - nfrd::parser::RSSItem, 121
- RSSParser
 - nfrd::parser::RSSParser, 126
- Read
 - nfrd::config::ConfigManager, 45
 - nfrd::config::ConfigSector, 49
 - nfrd::misc::Utility, 18
- ReadDom
 - nfrd::parser::AtomParser, 37
 - nfrd::parser::FeedParser, 70
 - nfrd::parser::RSSParser, 127
- ReadURL
 - nfrd::parser::FeedParser, 70
 - nfrd::parser::FeedXParser, 80
 - nfrd::parser::Parser, 115
- RefineFeed
 - nfrd::parser::FeedXParser, 80
- Register
 - nfrd::module::AdminService, 25
- release
 - nfrd::misc::AutoDB, 41
- remote_address
 - nfrd::module::AdminServiceThread, 29
- RemoveImage
 - nfrd::parser::FeedItem, 67
- Request
 - nfrd::module::CrawlerThread, 54
- reset
 - nfrd::misc::AutoDB, 41
- ResetPriority
 - nfrd::module::QueueItem, 118
- RunMaintenanceTasks
 - nfrd::module::Crawler, 52
- STARTING
 - nfrd::module::Module, 110
- STOPPED
 - nfrd::module::Module, 110
- STOPPING
 - nfrd::module::Module, 110
- SIZE_MODIFIER
 - FeedXParser.cpp, 152
- second
 - nfrd::misc::DateTime, 64
- Set
 - nfrd::misc::DateTime, 61
- SetAuthor
 - nfrd::parser::AtomItem, 34
 - nfrd::parser::RSSItem, 123
- SetContent
 - nfrd::parser::AtomItem, 34
 - nfrd::parser::FeedItem, 67
 - nfrd::parser::RSSItem, 124
- SetContentUpdateAverage
 - nfrd::module::QueueItem, 118
- SetDateOffset
 - nfrd::misc::DateTime, 61
- SetDay
 - nfrd::misc::DateTime, 62
- SetFeedType
 - nfrd::module::QueueItem, 119
- SetFileName
 - nfrd::config::ConfigManager, 46
- SetFullContent
 - nfrd::parser::FeedItem, 67
- SetGeoLocation
 - nfrd::parser::FeedItem, 68
- SetHour
 - nfrd::misc::DateTime, 62
- SetMinute
 - nfrd::misc::DateTime, 62
- SetModule
 - nfrd::Master, 108
- SetMonth
 - nfrd::misc::DateTime, 63
- SetNumberOfUsersInSystem
 - nfrd::module::FeedPriorityQueue, 74
- SetPostDate
 - nfrd::parser::AtomItem, 34
 - nfrd::parser::RSSItem, 124
- SetSecond
 - nfrd::misc::DateTime, 63
- SetTimeOffset
 - nfrd::misc::DateTime, 63
- SetTitle
 - nfrd::parser::AtomItem, 34
 - nfrd::parser::RSSItem, 124
- SetURL
 - nfrd::parser::AtomItem, 35
 - nfrd::parser::RSSItem, 124
- SetUsersAffected
 - nfrd::module::QueueItem, 119
- SetYear

- nfrd::misc::DateTime, 63
- shutdown_main
 - nfrd::module::AdminServiceThread, 28
- sin
 - nfrd::module::AdminServiceThread, 28
- size
 - nfrd::misc::Image::File, 84
- sout
 - nfrd::module::AdminServiceThread, 28
- src/AdminService.cpp, 147
- src/AdminServiceThread.cpp, 147
- src/AtomParser.cpp, 148
- src/ConfigManager.cpp, 148
- src/Crawler.cpp, 149
- src/CrawlerThread.cpp, 150
- src/DateTime.cpp, 150
- src/FeedParser.cpp, 151
- src/FeedPriorityQueue.cpp, 151
- src/FeedXPather.cpp, 152
- src/Image.cpp, 153
- src/LogManager.cpp, 153
- src/Master.cpp, 154
- src/Module.cpp, 154
- src/Parser.cpp, 156
- src/QueueItem.cpp, 156
- src/RSSParser.cpp, 157
- src/Statistics.cpp, 157
- src/Utility.cpp, 158
- src/nfrd.cpp, 155
- Start
 - nfrd::module::FeedPriorityQueue, 74
 - nfrd::module::Module, 112
- start_time
 - nfrd::Master, 109
- StartThreads
 - nfrd::module::Crawler, 52
 - nfrd::module::FeedPriorityQueue, 74
- Statistics
 - nfrd::module::Statistics, 128
- Status
 - nfrd::module::Module, 110
- status
 - nfrd::module::Module, 112
- status_main
 - nfrd::module::AdminServiceThread, 29
- status_send_module_status
 - nfrd::module::AdminServiceThread, 29
- Stop
 - nfrd::module::AdminService, 25
 - nfrd::module::Crawler, 52
 - nfrd::module::CrawlerThread, 54
 - nfrd::module::FeedPriorityQueue, 74
 - nfrd::module::Module, 112
 - nfrd::module::Statistics, 129
- stream
 - nfrd::module::AdminServiceThread, 29
- TRIMMED_TAGS
 - FeedXPather.cpp, 153
- Task
 - nfrd::Master, 106
- task_queue
 - nfrd::Master, 109
- tempOutgoingQueue
 - nfrd::module::FeedPriorityQueue, 76
- Terminate
 - nfrd.cpp, 156
 - nfrd::Master, 108
- TextSize
 - nfrd::parser::FeedXPather, 80
- thread
 - nfrd::module::Module, 112
- thread_mutex
 - nfrd::module::AdminService, 25
- threads
 - nfrd::module::AdminService, 25
 - nfrd::module::Crawler, 52
- timeSpentInQueue
 - nfrd::module::QueueItem, 119
- timeout
 - nfrd::module::AdminServiceThread, 29
- title
 - nfrd::parser::AtomItem, 35
 - nfrd::parser::RSSItem, 125
- ToBool
 - nfrd::misc::Utility, 19
- ToInt
 - nfrd::misc::Utility, 19
- tracker
 - nfrd.cpp, 156
- Trim
 - nfrd::misc::Utility, 19
- TrimEmptyTag
 - nfrd::parser::FeedXPather, 81
- trimmed_tags
 - nfrd::parser::FeedXPather, 81
- Type
 - nfrd::log::LogManager, 102
 - nfrd::misc::Image, 86
- type
 - nfrd::misc::Image::File, 84
- UNKNOWN
 - nfrd::misc::Image, 86
- UnloadModule
 - nfrd::Master, 108
- UnloadModules
 - nfrd::Master, 108
- UpdateItem
 - nfrd::module::CrawlerThread, 54
- updatedMutex
 - nfrd::module::FeedPriorityQueue, 76
- url
 - nfrd::parser::AtomItem, 35
 - nfrd::parser::FeedParser, 70
 - nfrd::parser::RSSItem, 125
- username
 - nfrd::module::AdminService, 25

usersAffected
 nfrd::module::QueueItem, [119](#)

Utility.cpp
 _write_data, [158](#)
 WHITESPACES, [159](#)

value
 nfrd::config::ConfigManager, [46](#)
 nfrd::config::ConfigSector, [50](#)

WARNING
 nfrd::log::LogManager, [103](#)

WBMP
 nfrd::misc::Image, [86](#)

WAITTIME
 nfrd::module::Crawler, [53](#)

WHITESPACES
 ConfigManager.cpp, [149](#)
 Utility.cpp, [159](#)

what
 nfrd::config::ConfigException, [43](#)
 nfrd::log::LogException, [101](#)
 nfrd::misc::ImageException, [90](#)
 nfrd::module::ModuleException, [113](#)
 nfrd::parser::ParserException, [116](#)

Write
 nfrd::config::ConfigManager, [46](#)
 nfrd::config::ConfigSector, [49](#)

year
 nfrd::misc::DateTime, [64](#)