News Feeder Refresh Daemon (nfrd)

Generated by Doxygen 1.8.1.1

Tue Oct 23 2012 16:48:36

Contents

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

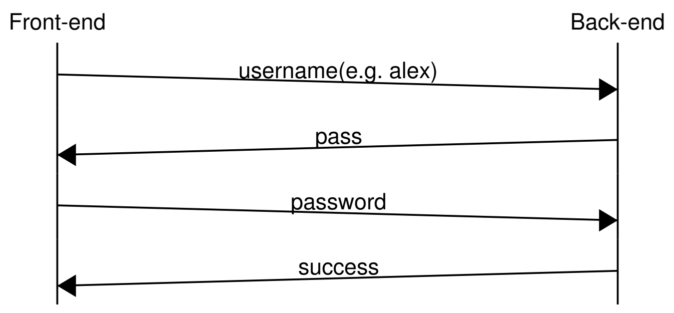
The back-end listens to both unix socket and tcp socket as defined by the config file. The user could turn off unix socket listening or tcp socket listening by modifying the config file.

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.

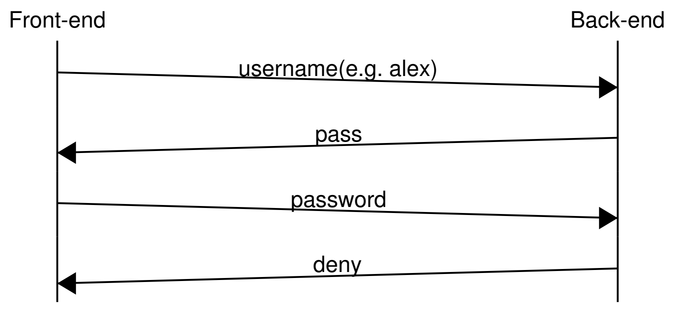
2.2 Login

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

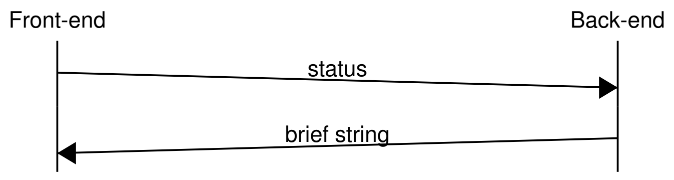


2.3 status - Check the status

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.

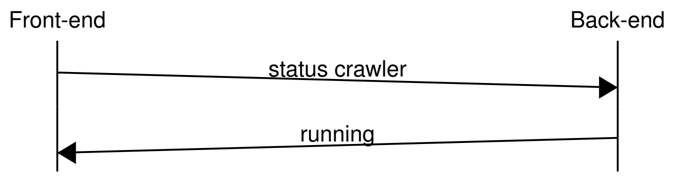


In the later version, statistical information will be available.

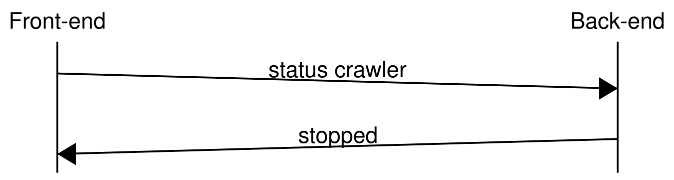
2.3.1 crawler - crawler status

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

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



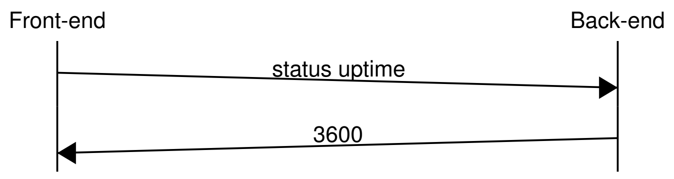
If the crawler is currently stopped, it will return "stopped":



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

2.3.2 uptime - Back-end uptime

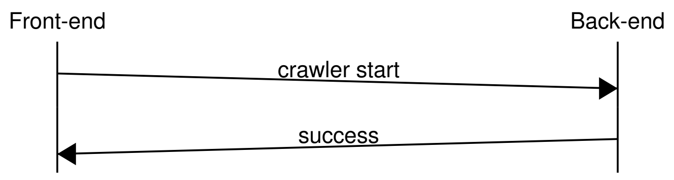
This sub-command "uptime" will return back how many seconds that the back-end has run. For example: (has run for 1 hour)



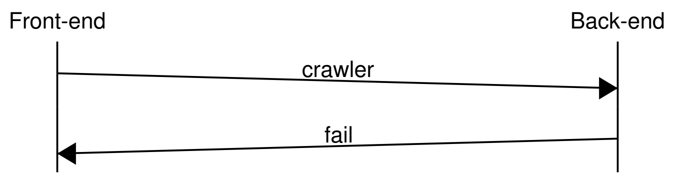
2.4 crawler - Control the crawler

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:

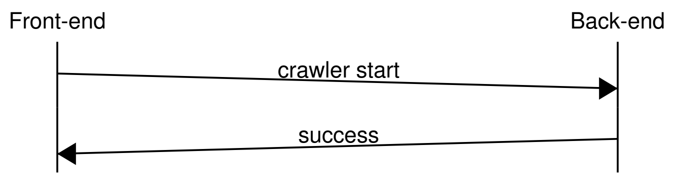


If no sub-command is attached to the command "crawler", the back-end will return a "fail" as following:

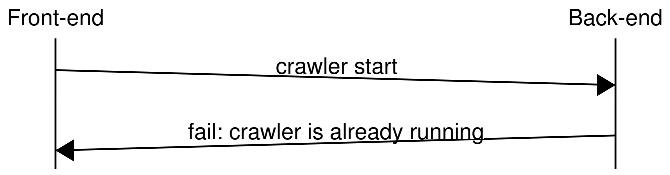


2.4.1 start - Start the crawler

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

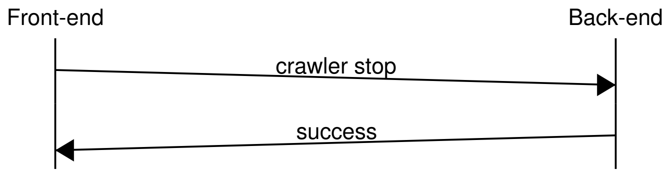
This sub-command "status" is exactly the same as command "status crawler".

See also

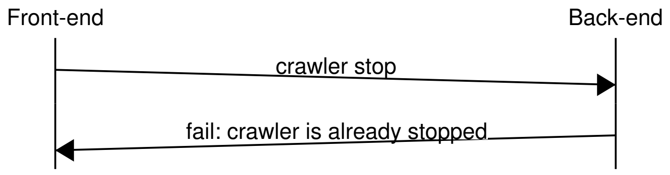
crawler - crawler status

2.4.3 stop - Stop the crawler

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

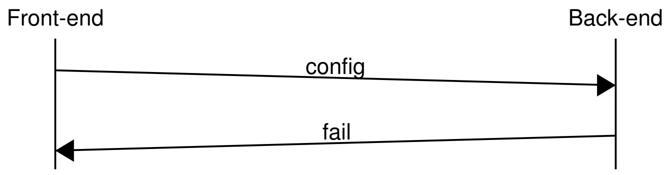


If exceptions are thrown and caught by the back-end, it will return a "fail" with error message followed by a ": ":



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

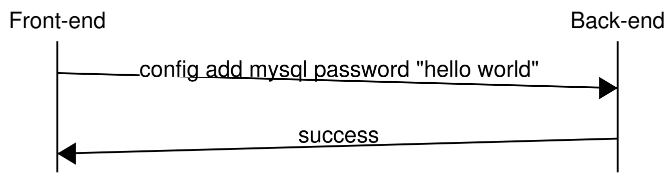
config::ConfigManager

2.5.1 add - Add/Change an attribute

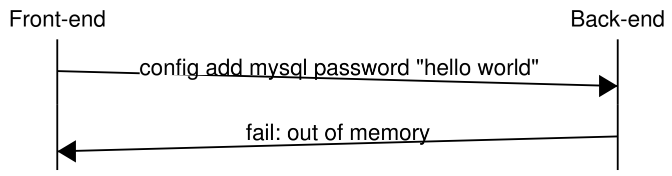
This sub-command "add" tells the back-end to add or change an attribute. The grammar for this command is:

```
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 ": ":



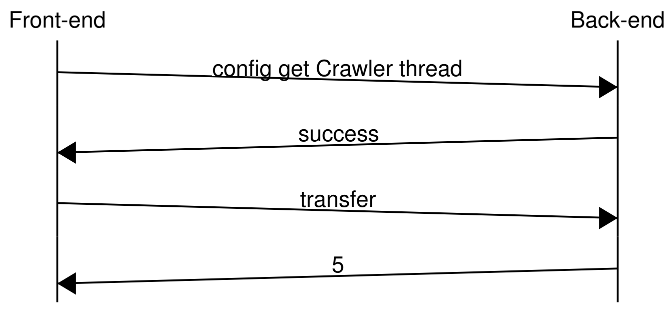
If the required sector or attribute does not exist, it will automatically add a new one. If the attribute is already there, the back-end will change it.

2.5.2 get - Return an attribute

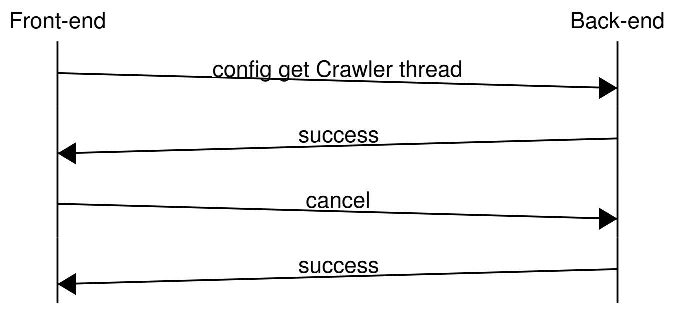
This sub-command "get" tells the back-end to get the value of an attribute. The grammar for this command is:

```
config get sector attribute
```

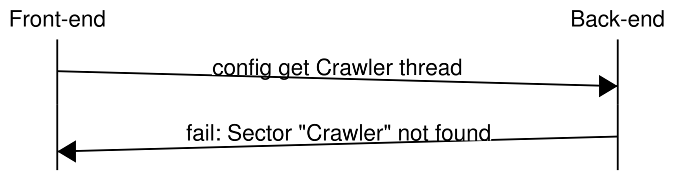
The back-end will search for this attribute as requested. if request is successful, the back-end will return "success". Otherwise, it will return "fail", followed by an error message. Once success, the front-end can get the value back by sending "transmit".



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:



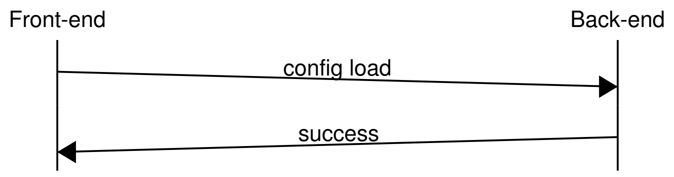
2.5.3 load - Load configuration file

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

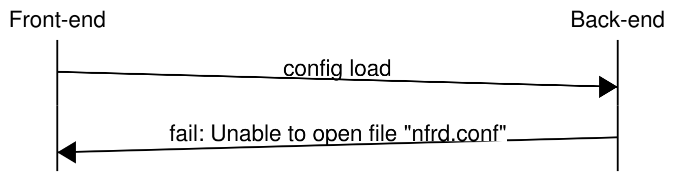
```
config load [file]
```

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()

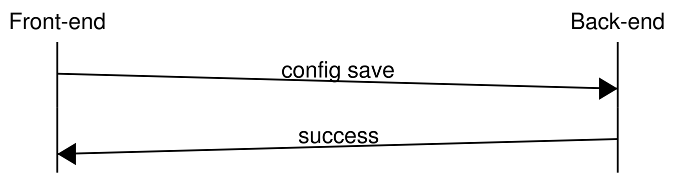
2.5.4 save - Save configuration file

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

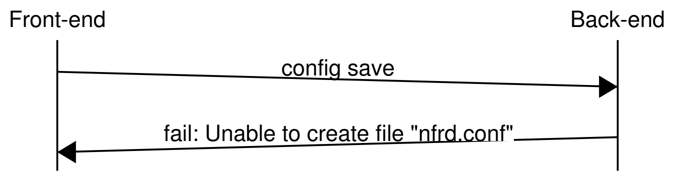
```
config save [file]
```

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:

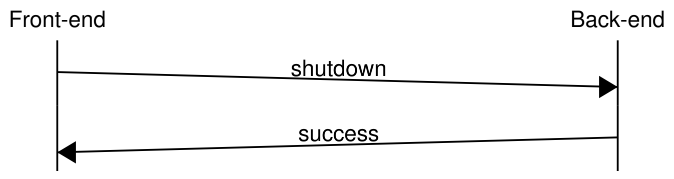


See also

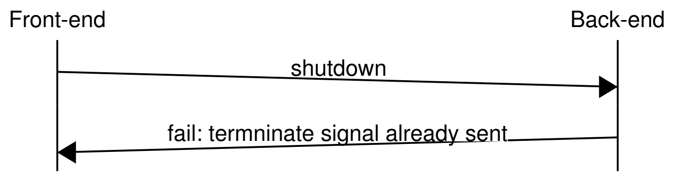
config::ConfigManager::Write()

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:

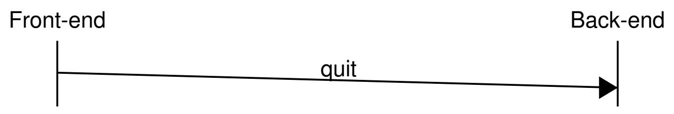


Warning

This command should be used for maintenance only.

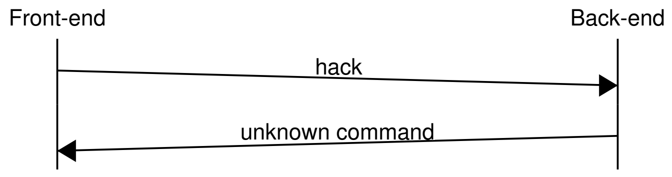
2.7 quit - Close connection

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

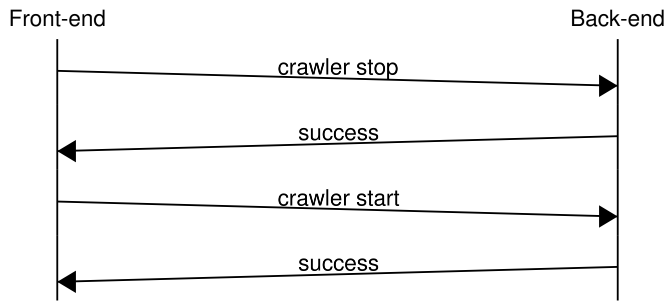
If a command is not supported by the back-end, a message of "unknown command" will be returned.



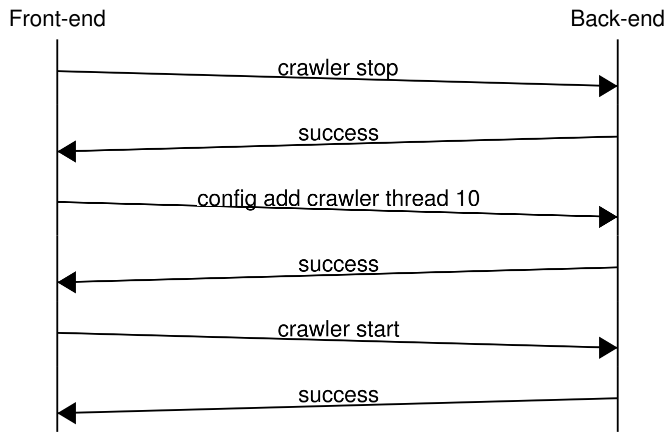
2.9 Examples

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 deallocte 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 interpretor on Unix in operations of quotes (" and ') and backslash Note: *args* will be truncated

Parameters

in	whether the arguments comes from
args	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

url	URL of the file
container	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

url	URL of the file
container	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

url	URL of the file

Returns

the file data

Exceptions

length_error	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

str	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

str	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

str	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

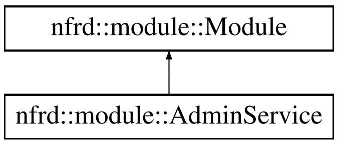
Class Documentation

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

master	Master of nfrd
config	config manager
log	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

username	username from front-end
password	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

thread	finished AdminServiceThread
dynamic	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

thread	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:

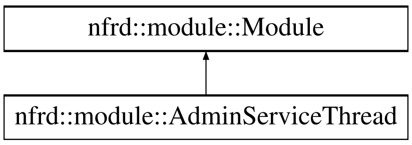
- include/nfrd/AdminService.h
- src/AdminService.cpp

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 baisc 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

parent	parent of this thread
master	Master of nfrd to be controled
timeout	timeout on socket operation
dynamic	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

args	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

args	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

args	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

args	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

name	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 parrent.

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:

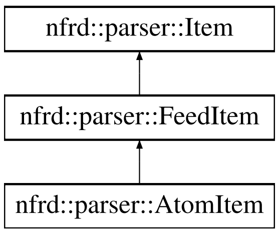
- include/nfrd/AdminServiceThread.h
- src/AdminServiceThread.cpp

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

HasNoValue	if the item has no author or if this function is not overrided

Reimplemented from nfrd::parser::Item.

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

Get the content of the item.

Returns

content

Exceptions

HasNoValue	if the item has no content or if this function is not overrided

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

HasNoValue	if the item has no post date or if this function is not overrided

Reimplemented from nfrd::parser::Item.

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

Get the title of the item.

Returns

title

Exceptions

HasNoValue	if the item has no title or if this function is not overrided

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

HasNoValue	if the item has no URL or if this function is not overrided

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 overrided

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 overrided

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 overrided

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 overrided

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 overrided

Reimplemented from nfrd::parser::Item.

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

Set the author of the item.

Parameters

source	author of the item

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

Set the content of the item.

Parameters

source	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

source	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

source	post date of the item

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

Set the title of the item.

Parameters

source	title of the item

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

Set the url of the item.

Parameters

source	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:

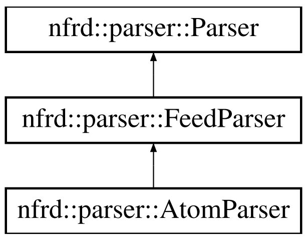
- include/nfrd/AtomParser.h
- src/AtomParser.cpp

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

HasNoValue	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

doc	parsed xml document of the feed resource

Exceptions

InvalidSource	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 deallocte 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 deallocte 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

p	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

a	Another AutoDB of the same type.

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

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

a	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 ownsership.

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

a	Another AutoDB of the same type.

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

a	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 ownsership. 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

p	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:

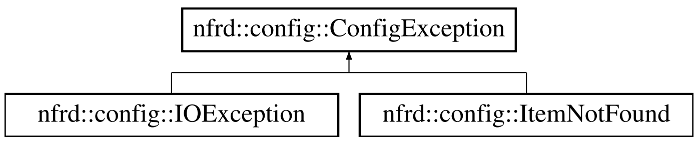
- include/nfrd/AutoDB.h

5.7 nfrd::config::ConfigException Class Reference

General exception for config.

`#include <ConfigManager.h>`

Inheritance diagram for nfrd::config::ConfigException:


nfrd::config::ConfigException
nfrd::config::IOException
nfrd::config::ItemNotFound

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

message	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 &sector)

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

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

filename	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

sector	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

sector	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

filename	The name of the file to read

Exceptions

IOException	If file does not exist or file corrputed.

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

Externally set the file name of the config file.

Parameters

filename	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

filename	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

arg	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

arg	Argument name

Returns

Value of fetched argument

Exceptions

ItemNotFound	If the argument does not exist

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

Read in ConfigSector.

Parameters

in	Read ConfigSector from this stream

Exceptions

IOException	If argument is not correctly formated

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

Write out ConfigSector.

Parameters

out	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:

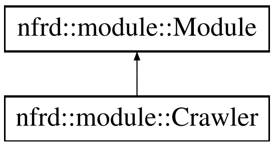
- include/nfrd/ConfigManager.h
- src/ConfigManager.cpp

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 priortiy 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

config	The config manager to use
log	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 priortiy 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

config	The config manager to use
log	The log manager to use
crawler	The crawler to use (prority 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

item	Item to update
feed	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 ()

    Dafault Constructor for Datatime.
- 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 datatime 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 ( )

Dafault 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

rhs	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 datatime 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

source	original date time string

Exceptions

std::domain_error	if

Parameters

source	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

rhs	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

rhs	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

rhs	DateTime 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

rhs	DateTime 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

source	original date time string

Exceptions

std::domain_error	if

Parameters

source	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

source	original date time string

Exceptions

std::domain_error	if

Parameters

source	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 coverneted to UTC+0000.

Parameters

source	original date time string

Exceptions

std::domain_error	if

Parameters

source	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

y	year
n	month
d	day
h	hour
m	minute
s	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

d	day
m	month
y	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

d	day part

Exceptions

std::domain_error	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

h	hour part

Exceptions

std::domain_error	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

min	minute part

Exceptions

std::domain_error	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

m	month part

Exceptions

std::domain_error	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

sec	second part

Exceptions

std::domain_error	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

h	hour
m	minute
s	second

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

Set the year part of the DateTime.

The year could be nagetive which means BC.

Parameters

y	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:

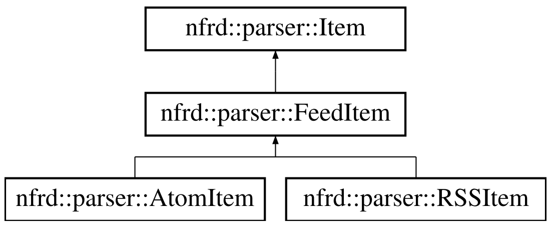
- include/nfrd/DateTime.h
- src/DateTime.cpp

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

source	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 overrided

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 overrided

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 overrided

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 overrided

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

source	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

source	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

hasFullContent	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

source	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:

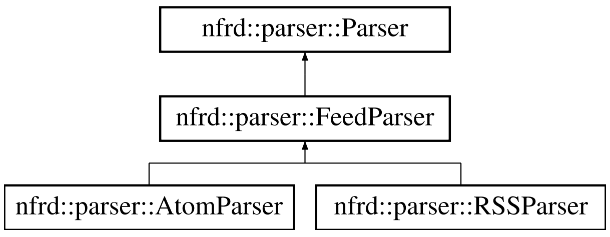
- include/nfrd/FeedParser.h
- src/FeedParser.cpp

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

doc	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 resouce from an URL and parse into a list of Item.

Parameters

url	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:

- Proritising 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

config	The config manager to use
log	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

item	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

item	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 thead 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 effecient 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:

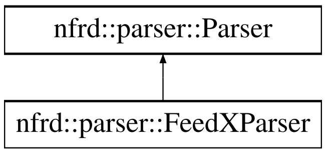
- include/nfrd/FeedPriorityQueue.h
- src/FeedPriorityQueue.cpp

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 resouce 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 FeedXParser::FeedXParser ( )

Initialising Constructor for FeedParser.

5.16.2.2 FeedXParser::∼FeedXParser ( )

Delete all dynamic memory, if any.

5.16.3 Member Function Documentation

5.16.3.1 tree< htmlcxx::HTML::Node >::iterator FeedXParser::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

dom	source dom tree
match_string	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 FeedXParser::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

dom	source dom tree
match_string	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

iterator_pair	a iterator pair of the root node
content	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

item	where to add geographical location
iterator_pair	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

item	where to add images
iterator_pair	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

HasNoValue	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

HasNoValue	if the item has no last build date

Reimplemented from nfrd::parser::Parser.

5.16.3.8 std::string FeedXParser::GetLongestText (const std::string & *source*) [static], [private]

Get the longest text from a html source code.

Parameters

source	html source code

Returns

the longest text in *source*

5.16.3.9 void FeedXParser::PatchAndMatch (FeedItem ∗ *item*) [static], [private]

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

Parameters

item	an item to be patched.

5.16.3.10 void FeedXParser::ReadURL (const std::string & *url*) [virtual]

Read resouce from an URL and parse into a list of Item.

Parameters

url	URL address of the feed resource

Exceptions

InvalidSource	if the url or the feed resource is invalid

Implements nfrd::parser::Parser.

5.16.3.11 void FeedXParser::RefineFeed (FeedItem ∗ *item*) [static], [private]

Refine a feed by stripping tags and extracting images.

Parameters

item	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

iterator	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

source	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 ownner)

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

data	starting memory address of the file
size	size of the memory block
type	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

file	source file

Exceptions

ImageException	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

file	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:

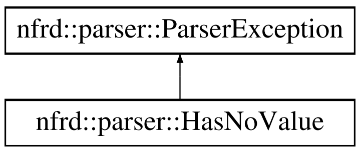
- include/nfrd/Image.h
- src/Image.cpp

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

msg	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 ownner)

Public Types

- enum Type {
  UNKNOWN, GD, GD2, GIF,
  JPEG, PNG, WBMP }

  Types of Image.

Public Member Functions

- Image ()

  Dafault 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 ( )

Dafault Constructor for Image.

5.19.3.2 Image::Image (const Image & *rhs*)

Copy Constructor for Image.

Parameters

rhs	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

quality	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

compression	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

width	specified size
height	specified height

Returns

resized image

Exceptions

ImageException	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

file	file that contains a memory block of image

Exceptions

ImageException	if fail to load

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

Load image from specified URL.

Parameters

url	url address of the image
type	type of the image

Exceptions

ImageException	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

data	memory block of the file
type	type of the image

Exceptions

ImageException	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

data	starting memory address of the file
size	size of the memory block
type	type of the image

Exceptions

ImageException	if fail to load

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

Copy an image from another image.

Parameters

image	source image

Returns

∗this

Exceptions

ImageException	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

message	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:

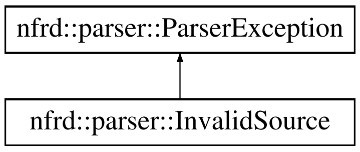
- include/nfrd/Image.h
- src/Image.cpp

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

msg	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:

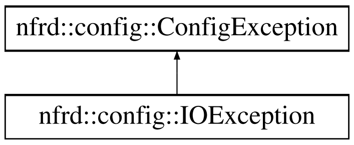
- include/nfrd/Parser.h
- src/Parser.cpp

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

msg	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:

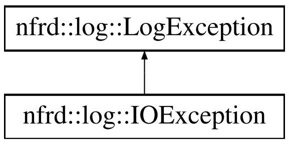
- include/nfrd/ConfigManager.h
- src/ConfigManager.cpp

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

msg	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:

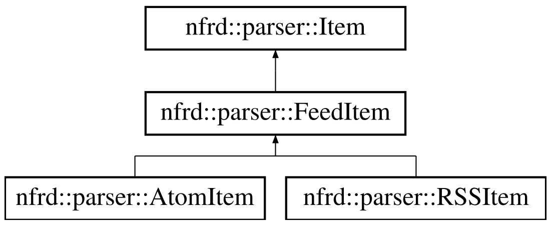
- include/nfrd/LogManager.h
- src/LogManager.cpp

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

HasNoValue	if the item has no author or if this function is not overrided

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

HasNoValue	if the item has no content or if this function is not overrided

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

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

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

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

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

HasNoValue	if the item has no post date or if this function is not overrided

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

HasNoValue	if the item has no title or if this function is not overrided

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

HasNoValue	if the item has no URL or if this function is not overrided

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 overrided

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 overrided

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 overrided

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 overrided

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 overrided

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 overrided

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 overrided

Reimplemented in nfrd::parser::AtomItem, and nfrd::parser::RSSItem.

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

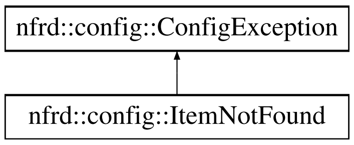
- include/nfrd/Parser.h
- src/Parser.cpp

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

msg	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

iterator	where it and end are extracted
is_allowed_tag	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

node	where it and end are extracted
is_allowed_tag	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:

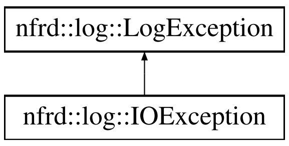
- include/nfrd/FeedXParser.h

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

message	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

filename	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

message	message to log
type	log type
title	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

message	message to log
title	title (module name) of the log
type	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

name	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

name	name of module to be loaded

Exceptions

ModuleException	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

configFile	filename of the config file

Exceptions

ModuleException	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

name	module name
start	start the module if true, else stop the module

Exceptions

ModuleException	if module not found

5.29.4.10 void Master::SetModule (module::Module ∗ *module*, bool *start*)

Set a module to start or stop.

Parameters

module	module to be operated
start	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

name	name of module to be unloaded

Exceptions

ModuleException	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:

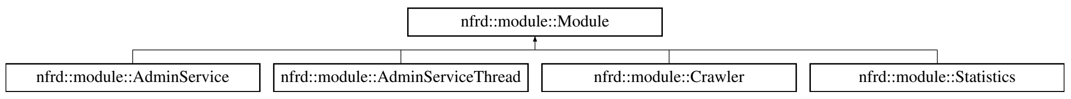
- include/nfrd/Master.h
- src/Master.cpp

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

name	module name
config	config manager
log	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

name	module name
module	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

message	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:

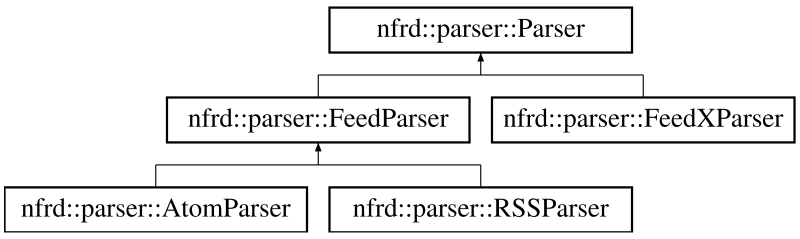
- include/nfrd/Module.h
- src/Module.cpp

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 resouce 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

HasNoValue	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

HasNoValue	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 resouce from an URL and parse into a list of Item.

Parameters

url	URL address of the feed resource

Exceptions

InvalidSource	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:

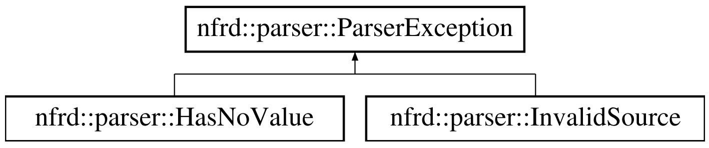
- include/nfrd/Parser.h
- src/Parser.cpp

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

message	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 (nfdb::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

maxpop	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:

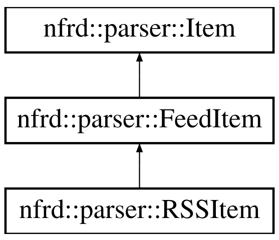
- include/nfrd/QueueItem.h
- src/QueueItem.cpp

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 overrided

Reimplemented from nfrd::parser::Item.

5.35.3.2 const string & RSSItem::GetContent ( ) const [virtual]

Get the content of the item.

Returns

content

Exceptions

HasNoValue	if the item has no content or if this function is not overrided

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

HasNoValue	if the item has no post date or if this function is not overrided

Reimplemented from nfrd::parser::Item.

5.35.3.4 const string & RSSItem::GetTitle ( ) const [virtual]

Get the title of the item.

Returns

title

Exceptions

HasNoValue	if the item has no title or if this function is not overrided

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

HasNoValue	if the item has no URL or if this function is not overrided

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 overrided

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 overrided

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 overrided

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 overrided

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 overrided

Reimplemented from nfrd::parser::Item.

5.35.3.11 void RSSItem::SetAuthor (const char * *source*)

Set the author of the item.

Parameters

source	author of the item

5.35.3.12 void RSSItem::SetContent (const char ∗ *source*)

Set the content of the item.

Parameters

source	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

source	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

source	post date of the item

5.35.3.15 void RSSItem::SetTitle (const char ∗ *source*)

Set the title of the item.

Parameters

source	title of the item

5.35.3.16 void RSSItem::SetURL (const char ∗ *source*)

Set the url of the item.

Parameters

source	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:

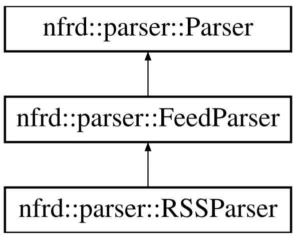
- include/nfrd/RSSParser.h
- src/RSSParser.cpp

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

HasNoValue	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

node	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

doc	parsed xml document of the feed resource

Exceptions

InvalidSource	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:

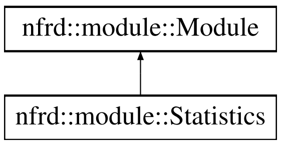
- include/nfrd/RSSParser.h
- src/RSSParser.cpp

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

config	The config manager to use
log	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 deallocte 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 Stardard 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 databse.

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 ownner)
- 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 <libnfdb/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 periodly 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 <libnfdb/FeedController.h>
#include <libnfdb/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 <libnfdb/UserController.h>
```

6.33.1 Detailed Description

Author

Aron Hardy-Bardsley `ajrhb957@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 tage that only text part is kept.
- static const string::size_type SIZE_MODIFIER = 3

  The right paragraph shoud 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 shoud 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

argc	count number of arguments taken from command line
argv	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

prog	the excutable name of the program

6.39.3.3 void Terminate (int *signo*)

A signal handler to safely terminate this program.

Parameters

signo	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 inteface API. The functionality is to write bytes obtain from the url to the container. This function is static, meaning that this function is only accessable in this file.

Parameters

ptr	data pointer
size	size of a memory block
nmemb	number of memory block
stream	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