Arboreal

Generated by Doxygen 1.8.14

Contents

Chapter 1

Arboreal User Guide

Table of Contents

Installing Arboreal

Arboreal is currently not integrated with the kernel and as such runs similarly to a virtual file system albeit with a more experimental structure. Future work will be focused on direct integration with the kernal in order to provide more traditional usability. In the meantime playing around with and testing the file system can be achived through a few easy steps:

1. **Download the project**
   2. **Changed directory to the folder within the project hierarchy named `Source`**
   3. **Type `make`**
   4. **You will now need to first run the daemon process. This process intercepts al communication attempts with the File System and will execute functions accordingly. There are a number of command line arguments that can be passed to the daemon:**
   * `-d` **This flag is used to tell the daemon to enable debugging**
   * `-v` **This flag is used to tell the daemon to return file information (such as that returned by a call to find) with as much information as possible. Omitting this flag will cause the daemon to return a reduced version of file information**
   * **You may enable either of these options or both (input order does not matter, that is `-d -v` will work the same as `-v -d`)**
2. **Finally, Simply Type `./daemon` followed by your chosen flag or flags (be sure to include a space in between). For example, if I wanted to run the daemon with verbose file information and debugging enabled the command would look like: `./daemon -v -d`**
   At this point you'll be ready to move on to the next step, starting the command line or GUI interface. Notice that the daemon does not output anything to the screen as it is running. **This is OK!** Its whole purpose is to be a background process that aids communication with the file system. **If you decided to enable debugging, the output will be located in a file called `Arboreal.log`.**

A final note:
By typing make, the "disk" will be formatted for you with the default values and partition names/ counts. It is possible to change these to better suit your needs however it is a little bit more involved.

1. **Open and edit a file called `diskInfo.d` (It is located in the `Source` folder)**
2. **Using the following syntax, edit the file as you see fit:**
   - **Line 1 needs to always be:** `Diskfile name, number of blocks on disk, size of each block in bytes, number of partitions` **(Omit the commas in favor of spaces)**
   - **Lines 2 - X need to always be:** `Partition name, number of blocks in the partition, maximum filename size` **(Again omit commas in favor of spaces)**
     * **There are some restrictions on allowed values for the `diskInfo` file to see these please checkout the Arboreal Technical Documentation**
3. **Next you will have to open `daemon.cpp` (it is located under `Source/Filesystem/`) and edit this line of code:**
   ```
   *d = new Disk(#1, #,2 const_cast<char *>("diskfile_name"));
   ```
   - **Change** `#1` **to whatever value you picked for** `number of blocks` **in** `diskInfo.d`**.**
     - So if I decided that I wanted 4000 blocks I would type `4000` in for `#1` (**The number here and in** `diskInfo.d` **MUST MATCH**).
   - **Change** `#2` **to whatever value you picked for** `block size in bytes` **in** `diskInfo.d`
     - So if I decided that I wanted blocks to be 4096 bytes large I would type `4096` in for `#2` (**Once again I stress that the number here and in** `diskInfo.d` **MUST MATCH**)
   - **Finally, change** `diskfile_name` **to the name of the disk file you chose in** `diskInfo.d`
4. **You are now almost ready, the final step is to type** `make clean` **followed by** `make` **in the shell and run the through the same steps as above for starting the daemon**
5. **You are now good to go!**

Starting The Command Line

Before beginning anything below, make sure that a daemon process (and ONLY ONE daemon process) is running, if the command line cannot connect to the daemon process it will quit on startup with an appropriate error message
The command line utility has multiple optional arguments but it does contain a single mandatory argument. This is the **Partition Name** that the command line will be working on. If no partition name is given the command line will fail on startup with an appropriate error message. Additionally it is important to note that **the partition that is given to the command line must already exist**. If it does not, an appropriate error will be thrown. Finally, **it does not matter if the partition is already in use by another command line and it does not matter how many command lines are currently active**, in both cases you will still be able to work wth the file system (provided that the partition you gave exists). After providing the partitoon name you are free to run the command line. However, should you wish to, there are some optional command line arguments:

- `-d` **This argument will enable debugging for both the command line and the liaison process**
  `* -s` **This argument will alert the command line that input will be coming from a file rather than a user**
- `-s -d` **This will enable debugging for the command line AND alert it to the fact that input will be coming from a file rather than a user**

For example, if i wished to pipe input from a file to the command line and enable debugging, I would run the command line process like so: `./commandline PartitionName -s -d < some_random_file.ext` (*Note that* `./commandline -d -s < some_random_file.ext` *will not work, that is, make sure the debug flag comes last!*)

But if I wanted to just enable debugging and read from user input, I would run the command line process like so:
`./commandline PartitionName -d`
At this point you should see the arboreal header and `Arboreal >>` indicating that the command line is ready to accept input. **To send input to the command line simply type the command you wish to execute** (see *Valid Commands And Their Syntax* section for commands or type `help` or `h`) **and press enter.**

Note
If you chose to enable debugging for the command line, all debug output will be written to a file named `Arboreal.log`. Do not worry if this file does not yet exist, it will be created for you on startup.

Valid Commands And Their Syntax

Help Commands

Arboreal >> help Arboreal >> h **These two commands will bring up a helper subprocess which will display a list of the command archetypes and show the user the specific commands (and their syntax) that are housed under each archetype.** The helper subprocess continues running until the user decides to quit it.

```
Arboreal >> -h --command_archetype

e.g.
Arboreal >> -h --find
```

This version of the help command will show the usage for a single command archetype. (Unlike the `help` or `h` commands it will not start a "helper" subprocess but will simply display the usage for the particular archetype and await the next file system command)

Quit Commands

Arboreal >> quit Arboreal >> q Arboreal >> Q **All of these will attempt to terminate the current command line process.** This command does not affect other concurrently running command lines it will only quit the currently active command line process. The user must confirm the quit before the command will actually be executed. this is to prevent accidental quits. The quit commands are built with proper cleanup in mind and should not leave any junk behind.

Find Commands

Arboreal >> find -t [tagname1,tagname2,...] Arboreal >> find -t {tagname1,tagname2,...} Arboreal >> find -t [tagname1,{tagname2,tagname7,...},tagname10,...] Arboreal >> find -t {tagname1,tagnam3,[tagname5,tagname6,...],...} **This command searches for files by tag.** It is quite powerful and allows you to search for any combination of tags. Commands that use `{}` are called `sets` and will tell the file system to **search for ALL files which are tagged with ALL of the specified tags.** You can think of this as a bunch of `&&` operations, that is, you want a file tagged with :
`{ this tag, and this tag, and this tag, ... etc}`
Commands that use `[]` are called `lists` and will tell the system to **search for ANY file which is tagged with ANY of the tags specified.** You can think of this as a bunch of || operations, that is, you want a file tagged with:
`[this tag, or this tag, or this tag, ... etc]`
What's great is that you can actually nest any of these within one another! Although nesting a bunch of `sets` or `lists` won't be any diffferent from simply using one big list or set (i.e. `[t1,[t2,t3,t4]]` is the exact same as `[t1,t2,t3,t4]` this goes for `sets` as well). However, tings get interesting when you pass a command such as:

`find -t [tag1,tag2,{tag45,tag78,[tag9,tag10],tag5},tag100]`
This particular command will search for any file with:

```
tag1
tag2
tag100
tag1 && tag45 && tag78 && tag9 && tag5 && tag100
tag1 && tag45 && tag78 && tag10 && tag5 && tag100
tag2 && tag45 && tag78 && tag9 && tag5 && tag100
tag2 && tag45 && tag78 && tag10 && tag5 && tag100
```

(*Of course you accomplish similar things even with a command that is a `list` nested within a `set` rather than this example which is a `set` nested within a `list`*)

As you can see, nesting these operations creates some really powerful search options!

Important! DO NOT put spaces in between the `list` or `set` items!!

- `Arboreal >> find -f [file1,file2,...]`

  This is a file for specific code notes. things to do, consider, etc, that doesn't need to clutter up the main readme file.

  Doing TRY-CATCH

tageSearch() returns a vector of structs with (string "filename", int fidentifier) [fidentifier can be FIONODE blknum or unique file identifier that is mapped to a FIONDE blknum]

Hand off storage of file tagSearch() return vector to Danny to be stored in a "current" buffer or smoe such/

There should probably be an attributes object to make our lives easier. and thats what real filesystems do.

Attributes object should be stored in FINODE or another indirect block who's reference is stored in the FINODE. Which one is used should be decided dynamically, if FIONDE is full get empty data block, store address in FIONDE (migrate data)[optional] to new block, add new data to new block, otherwise add data directly to FIONDE. TAGS ARE ATTRIBUTES

I think we may need two open functions. One that takes the unique file id,(block number) and one that takes the vector of tags and the file name . similar to a path. **YES**

I removed validName() because we should check for valid input before passing it to our filesystem. as much as possible anyway.

I think we'll be able to get rid of alot of the helper functions actually. because map will be able to do all that for us. the **big helper functions will be reading in a map and writing out a map**. which i think we can just basically write out all the key, value pairs, because a map can do that easily with its iterator. **for reading in, we'll just read in all the key value pairs and add them to the map one by one.**

Name Length HARD CAPS at size specified in partition info during formatting NEED TREE INODE READING A MAP FROM DISK TO MAIN MEMORY **----------**

so we'll have to have a reserved spot at the end of a block for a block number to the next block of continuing data.

We should write everything out in plaintext and have a converter that can change it to byte stuff that we can implement later. also we should have a flag that will zero out blocks (FOR SPEED), mainly for debugging. but can also repourpose to an encrypt flag later.

//LATER: we should try not to write out the whole tag tree everytime. instead we should only write out the parts that changed if we can. I know this is a tough solution, if a tag is deleted in the middle of the tree and we really have no way of knowing where stuff will be in the tree... but it might be possible to keep some sort of secondary data structure, like a vector with all the info because it doens't matter what order we reconstruct the map in memory, just that all the data is there. this is also somehting we can implement later.

INtermeidary Data structure will store, (in addition to Memory pointer, block pointer) a tuple (int blknum, int pos_in_blknum) of the key_value pair so we can use it later for delete operations.

A NOTE about speed:

right now, in order to do tag search, we have to read in the finode of each file in the smallest tag tree becuase I am not storing the number of tags associated with a file in the tag tree inodes. This can be changed later, but for now I just want to get it done. If, when we are testing speeds this is something that will surely improve speed.

**Estimated read in time for everything on startup:

O(n^2*log(n))*

FileInode structure

filename - filenameSize Finode struct = sizeof(finode struct) local tag storage = rest of the space possible tag cont. block = sizeof(blknumType)

Restrictions:

1. **filename size restricted to no more than 1/2 block size**
2. **block size should be a power of 2**
3. **Hard cap on the number of tags that can be associated with a file. = (((blocksize - filenamesize - 136) / sizeof(BlkNumType)) + (blocksize / sizeof(BlkNumType)). 103 tags for blocksize of 512. and 64b filename**
4. **max block size = 16k**

TODO:

1. **Incorporate storing number of tags associated with file in Tag tree on disk, not yet**
2. **add renameTag function**
3. **don't allow duplicate tags to be sent to the filesystem when sending a tagset of any kind**

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

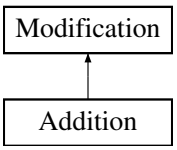
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Class Documentation

4.1 Addition Class Reference

Inheritance diagram for Addition:



Public Member Functions

- **Addition** (TreeObject ∗obj, TreeObject ∗parent)
- void **write_out** (PartitionManager ∗pm)

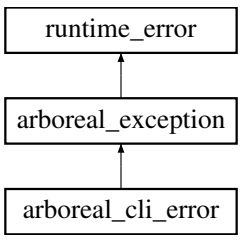
Additional Inherited Members

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

- Filesystem/DaemonDependancies/Trees/Trees.h
- Filesystem/DaemonDependancies/Trees/Trees.cpp

4.2 arboreal_cli_error Class Reference

Inheritance diagram for arboreal_cli_error:



Public Member Functions

- **arboreal_cli_error** (const string &where, const string &what, const int ecode=99)
- **arboreal_cli_error** (const char ∗what, const char ∗where, const int ecode=99)
- **arboreal_cli_error** (const char ∗what, const string &where, const int ecode=99)
- **arboreal_cli_error** (const string &what, const char ∗where, const int ecode=99)

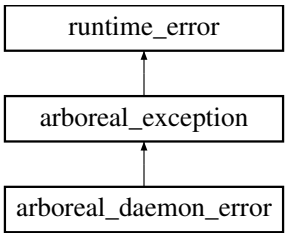
Additional Inherited Members

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

- SharedHeaders/Arboreal_Exceptions.h
- SharedCPPFiles/Arboreal_Exceptions.cpp

4.3 arboreal_daemon_error Class Reference

Inheritance diagram for arboreal_daemon_error:



Public Member Functions

- **arboreal_daemon_error** (const string &where, const string &what, const int ecode=99)
- **arboreal_daemon_error** (const char ∗what, const char ∗where, const int ecode=99)
- **arboreal_daemon_error** (const char ∗what, const string &where, const int ecode=99)
- **arboreal_daemon_error** (const string &what, const char ∗where, const int ecode=99)

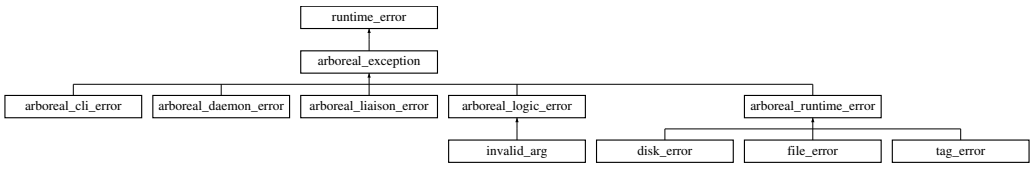
Additional Inherited Members

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

- SharedHeaders/Arboreal_Exceptions.h
- SharedCPPFiles/Arboreal_Exceptions.cpp

4.4 arboreal_exception Class Reference

Inheritance diagram for arboreal_exception:



Public Member Functions

- **arboreal_exception** (const char ∗what, const char ∗where, const int ecode=99)
- **arboreal_exception** (const char ∗what, const string &where, const int ecode=99)
- **arboreal_exception** (const string &what, const string &where, const int ecode=99)
- **arboreal_exception** (const string &what, const char ∗where, const int ecode=99)
- virtual const char ∗ **where** () const
- virtual const int **ecode** () const

Protected Attributes

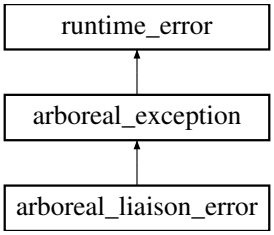
- string **_where**
- int **_ecode**

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

- SharedHeaders/Arboreal_Exceptions.h
- SharedCPPFiles/Arboreal_Exceptions.cpp

4.5 arboreal_liaison_error Class Reference

Inheritance diagram for arboreal_liaison_error:



Public Member Functions

- **arboreal_liaison_error** (const string &where, const string &what, const int ecode=99)
- **arboreal_liaison_error** (const char ∗what, const char ∗where, const int ecode=99)
- **arboreal_liaison_error** (const char ∗what, const string &where, const int ecode=99)
- **arboreal_liaison_error** (const string &what, const char ∗where, const int ecode=99)

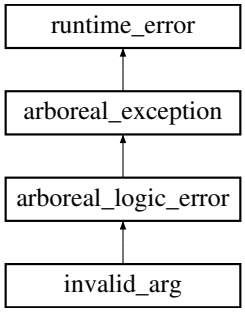
Additional Inherited Members

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

- SharedHeaders/Arboreal_Exceptions.h
- SharedCPPFiles/Arboreal_Exceptions.cpp

4.6 arboreal_logic_error Class Reference

Inheritance diagram for arboreal_logic_error:



Public Member Functions

- **arboreal_logic_error** (const char ∗what, const char ∗where, const int ecode=99)
- **arboreal_logic_error** (const char ∗what, const string &where, const int ecode=99)
- **arboreal_logic_error** (const string &what, const string &where, const int ecode=99)
- **arboreal_logic_error** (const string &what, const char ∗where, const int ecode=99)

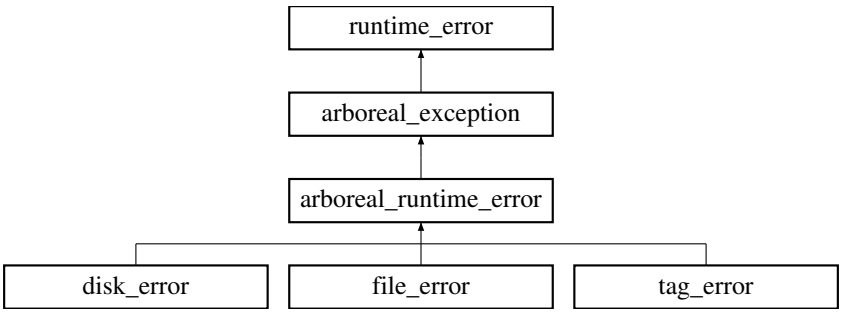
Additional Inherited Members

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

- SharedHeaders/Arboreal_Exceptions.h
- SharedCPPFiles/Arboreal_Exceptions.cpp

4.7 arboreal_runtime_error Class Reference

Inheritance diagram for arboreal_runtime_error:



Public Member Functions

- **arboreal_runtime_error** (const char ∗what, const char ∗where, const int ecode=99)
- **arboreal_runtime_error** (const char ∗what, const string &where, const int ecode=99)
- **arboreal_runtime_error** (const string &what, const string &where, const int ecode=99)
- **arboreal_runtime_error** (const string &what, const char ∗where, const int ecode=99)

Protected Attributes

- string **_where**
- int **_ecode**

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

- SharedHeaders/Arboreal_Exceptions.h
- SharedCPPFiles/Arboreal_Exceptions.cpp

4.8 Attributes Class Reference

Public Member Functions

- **Attributes** (BlkNumType blknum, PartitionManager *pm)

Modifier Functions

- void write_out ()
- void read_in ()
- void del ()
- void set_creation_time ()
- void set_owner (int owner)
- void set_permissions (char *perms)
- void set_access ()
- void set_edit ()
- void update_size (size_t size)

Accessor Functions

- time_t get_creation_time ()
- int get_owner ()
- char * get_permissions ()
- time_t get_access ()
- time_t get_edit ()
- size_t get_size ()
- FileAttributes get_file_attributes ()

4.8.1 Member Function Documentation

4.8.1.1 del()

```
void Attributes::del ( )
```

Removes the Attributes presence on disk

4.8.1.2 get_access()

```
time_t Attributes::get_access ( )
```

Returns

the UNIX time the file was last accessed

4.8.1.3 get_creation_time()

```
time_t Attributes::get_creation_time ( )
```

Returns

the UNIX time the file was created

4.8.1.4 get_edit()

```
time_t Attributes::get_edit ( )
```

Returns

the UNIX time the file was last edited

4.8.1.5 get_file_attributes()

```
FileAttributes Attributes::get_file_attributes ( )
```

Returns

the entire FileAttributes struct

4.8.1.6 get_owner()

```
int Attributes::get_owner ( )
```

Returns

the UID of the owner of the file

4.8.1.7 get_permissions()

```
char * Attributes::get_permissions ( )
```

Returns

the permisssions

See also

FileInfo::get_permissions(char*)

4.8.1.8 get_size()

```
size_t Attributes::get_size ( )
```

Returns

the size of the file in bytes

4.8.1.9 read_in()

```
void Attributes::read_in ( )
```

Reads in the Attributes from disk

4.8.1.10 set_access()

```
void Attributes::set_access ( )
```

Marks down the time as accessed time as UNIX timestamp

4.8.1.11 set_creation_time()

```
void Attributes::set_creation_time ( )
```

Marks down the creation time of the associated FileInfo as UNIX timestamp

4.8.1.12 set_edit()

```
void Attributes::set_edit ( )
```

Marks down the time as modified time as UNIX timestamp

4.8.1.13 set_owner()

```
void Attributes::set_owner (
            int owner )
```

Marks the owner as their UID

4.8.1.14 set_permissions()

```
void Attributes::set_permissions (
            char * perms )
```

sets the permisssions of the file

See also

FileInfo::set_permissions(char*)

4.8.1.15 update_size()

```
void Attributes::update_size (
            size_t size )
```

sets the size to the specified size

4.8.1.16 write_out()

```
void Attributes::write_out ( )
```

Writes out the Attributes to disk

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

- Filesystem/DaemonDependancies/Trees/Trees.h
- Filesystem/DaemonDependancies/Trees/Trees.cpp

4.9 CLI Class Reference

Public Member Functions

- CLI (char **partition)
- CLI (char **partition, bool debug)
- CLI (char **partition, char *isScript)
- CLI (char **partition, char *isScript, bool debug)
- ~CLI ()
- void start ()
- void run (std::string input)
- void run ()
- char * build (const int id, const std::string input)
- void send_cmnd (const char *command)
- void await_response ()

  Block while waiting for response from filesystem.

4.9.1 Constructor & Destructor Documentation

4.9.1.1 CLI() [1/4]

```
CLI::CLI (
            char ** partition )
```

Parameters

partition	A pointer to a charachter array containing the partition name that this particular command line interface will operate in

Constructor for use in Mode 1 of the Command Line Interface Reads from explicit user input Does NOT print debug data to log

4.9.1.2 CLI() [2/4]

```
CLI::CLI (
            char ** partition,
            bool debug )
```

Parameters

partition	A pointer to a charachter array containing the partition name that this particular command line interface will operate in
debug	Wether or not debug messages should be turned on for this interface

Constructor for use in Mode 2 of the Command Line Interface Reads from explicit user input Does PRINTS DEBUG data to log

4.9.1.3 CLI() [3/4]

```
CLI::CLI (
            char ** partition,
            char * isScript )
```

Parameters

partition	A pointer to a charachter array containing the partition name that this particular command line interface will operate in
isScript	Flag telling whether or not the input for this interface will be coming from a file (The flag value is '-s')

Constructor for use in Mode 3 of the Command Line Interface Reads from file Does NOT print debug data to log

4.9.1.4 CLI() [4/4]

```
CLI::CLI (
            char ** partition,
            char * isScript,
            bool debug )
```

Parameters

partition	A pointer to a charachter array containing the partition name that this particular command line interface will operate in
debug	Wether or not debug messages should be turned on for this interface
isScript	Flag telling whether or not the input for this interface will be coming from a file (The flag value is '-s')

Constructor for use in Mode 3 of the Command Line Interface Reads from file Does PRINTS DEBUG data to log

4.9.1.5 ∼CLI()

```
CLI::∼CLI ( )
```

Default Destructor

4.9.2 Member Function Documentation

4.9.2.1 await_response()

```
void CLI::await_response ( )
```

Block while waiting for response from filesystem.

Receive data from the liaison process The data is X number of charachters The data can be anything from a list of files returned by the 'find' operation To an error message. This function blocks until it receives data.

Most filesystem commands operate on a 1:1 ratio, that is, sending one command will generate one response. However, some commands (most notably 'find' & 'read') may have a ratio of 1:Many (For example 'find -t [tag1]' may return any number of files but it is only a single command). In situations such as these it is necssary to tell the Command Line to wait until the filesystem has sent all data. Thus, if the Command Line receives "WAIT" it will know to continue to block on a call to receive until the Liaison has gathered all of the nescessary data. However this is still not enough and it is also nescessary to tell the Command Line how much data it must read, for this reason, the first piece of data that the Liaison will send, will be the number of bytes the Command Line needs to read. After this value is received the actual data is sent.

4.9.2.2 build()

```
char * CLI::build (
            const int id,
            const std::string input )
```

Converts a std::string to a C-Style String, embeds the command id into the C-String, and pads it to length = Max↩ BufferSize

Parameters

id	File System Command ID
input	File System Command

Returns

A C-Style String of length = MaxBufferSize containing the command ID in the first X Bytes where X is the size of an integer type followed by the command itself followed by as many nullbytes as nescesarry in order to have a length = MaxBufferSize

Format user input for use by Liaison process:

1) Prepend a byte representation of the command ID to the array 2) Copy the user input into the the array (skip the first X indecies were X is the size of an integer (we don't want to overwrite the command ID))

Parameters

id	Comand ID
input	User input string

Returns

A pointer to a charachter array

4.9.2.3 run() [1/2]

```
void CLI::run (
            std::string input )
```

This function operates the same as run() but takes its input from a filestream rather than a user. Reads in the input data (A File System Command) and sends it down to the file system.
Some commands that do not need to interact with the File System code are handled in this function. For example, displaying the 'help' messages is executed from this function since the File System does not have or need and 'help' command. This function will block until it receives a response from the File System (provided that the command inputted is intended to go to the File System) this function will continue reading from the input file until an error occurs or 'end' is read in.

Parameters

input	A std::string value representing a File System command. This value is generally handed to the function by reading an input file. But may also be sent to it from another process such as a UI

4.9.2.4 run() [2/2]

```
void CLI::run ( )
```

Reads in the input data (A File System Command) and sends it down to the file system.
Some commands that do not need to interact with the File System code are handled in this function. For example, displaying the 'help' messages is executed from this function since the File System does not have or need and 'help' command. This function will block until it receives a response from the File System (provided that the command inputted is intended to go to the File System) this function will continue reading from user input until an error occurs or the user quits the application.

Reads input from user and sends it to the Liaison Process. Waits for corresponding data from the File System.

4.9.2.5 send_cmnd()

```
void CLI::send_cmnd (
            const char * cmnd )
```

Sends a command converted to a C-Style String to the Liaison Process for parsing and execution.

Parameters

command	A C-Style String of length = MaxBufferSize containing the command ID in the first X Bytes where X is the size of an integer type followed by the command itself followed by as many nullbytes as nescesarry in order to have a length = MaxBufferSize

Send user input (A filesystem command) to the Liaison Process

Parameters

cmnd	The input to send

4.9.2.6 start()

```
void CLI::start ( )
```

Performs initial set-up activities such as initiating connections and sending handshakes. Upon the completion of a successful handshake, run() is called and the interface is ready to use. If the handshake was not successful, the interface notifies the user and quits.

Run initial Command Line Interface setup operations:

1) Generate Shared Memory Segment For Process Synchronization 2) Fork And Run A Liaison Process 3) Create Sockets For Connection To Liaison 4) Send Handshake Command To File System 5) Run The Command Line

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

- CommandLineInterface/CLHeaders/Cli.h
- CommandLineInterface/Cli.cpp

4.10 DebugMessages Class Reference

Public Member Functions

- DebugMessages ()
- DebugMessages (std::string logfile_name)
- ~DebugMessages ()
- void ON (void)
- void OFF (void)
- template<typename T >
  void display (const T data, bool force=false)
- template<typename T >
  void log (const T data, bool force=false)
- template<typename T >
  void debug (const T data, bool force=false)
- void **lock** ()
- void **unlock** ()

4.10.1 Constructor & Destructor Documentation

4.10.1.1 DebugMessages() [1/2]

```
DebugMessages::DebugMessages ( )  [inline]
```

Create a new DebugMessage object using default logfile name: 'Arboreal.log' Automatically creates the log if it does not exist and if it does exist it will overwrite all the data in the log with the empty string. Sets the debug flag _DEBUG to FALSE on startup.

4.10.1.2 DebugMessages() [2/2]

```
DebugMessages::DebugMessages (
            std::string logfile_name )  [inline]
```

Create a new DebugMessage object using a user defined logfile name. Automatically creates the log if it does not exist and if it does exist it will overwrite all the data in the log with the empty string. Sets the debug flag _DEBUG to FALSE on startup.

4.10.1.3 ~DebugMessages()

```
DebugMessages::~DebugMessages ( )  [inline]
```

Default Destructor

4.10.2 Member Function Documentation

4.10.2.1 debug()

```
template<typename T >
void DebugMessages::debug (
            const T data,
            bool force = false )  [inline]
```

Template function for writing debug information to std::cout AND std::fstream.

Parameters

data	The data to be written to std::cout and a file. If the type of data passed is not supported by std::cout or outstream operators, behavior is undefined.
force	If data needs to be written before debugging offically starts this flag should be set to TRUE. Default value is FALSE.

4.10.2.2 display()

```
template<typename T >
void DebugMessages::display (
            const T data,
            bool force = false )  [inline]
```

Template function for writing debug information to std::cout ONLY.

Parameters

data	The data to be written to std::cout. If the type of data passed is not supported by std::cout, behavior is undefined.
force	If data needs to be written before debugging offically starts this flag should be set to TRUE. Default value is FALSE.

4.10.2.3 log()

```
template<typename T >
void DebugMessages::log (
            const T data,
            bool force = false )  [inline]
```

Template function for writing debug information to std::fstream ONLY.

Parameters

data	The data to be written to a file. If the type of data passed is not supported by outstream operators, behavior is undefined.
force	If data needs to be written before debugging offically starts this flag should be set to TRUE. Default value is FALSE.

4.10.2.4 OFF()

```
void DebugMessages::OFF (
            void  )  [inline]
```

Turns Debugging OFF Sets _DEBUG to FALSE

4.10.2.5 ON()

```
void DebugMessages::ON (
            void  )  [inline]
```

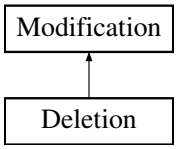
Turns Debugging ON Sets _DEBUG to TRUE

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

- SharedHeaders/DebugMessages.hpp

4.11 Deletion Class Reference

Inheritance diagram for Deletion:



Public Member Functions

- **Deletion** (TreeObject ∗obj, TreeObject ∗parent)
- void **write_out** (PartitionManager ∗pm)

Additional Inherited Members

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

- Filesystem/DaemonDependancies/Trees/Trees.h
- Filesystem/DaemonDependancies/Trees/Trees.cpp

4.12 Disk Class Reference

Public Member Functions

- Disk (BlkNumType numblocks, size_t blockSize, char ∗location)

Modifier Functions

- void writeDiskBlock (BlkNumType blknum, char ∗blkdata)

Accessor Functions

- void readDiskBlock (BlkNumType blknum, char ∗blkdata)
- size_t getBlockSize ()
- int getBlockCount ()

4.12.1 Constructor & Destructor Documentation

4.12.1.1 Disk()

```
Disk::Disk (
            BlkNumType numblocks,
            size_t blockSize,
            char * location )
```

Parameters

numblocks	the number of blocks on the Disk
blocksize	the block size for Disk blocks
location	the location of the Disk

4.12.2 Member Function Documentation

4.12.2.1 getBlockCount()

```
int Disk::getBlockCount ( )
```

Returns

the number of blocks on the entire Disk

4.12.2.2 getBlockSize()

```
size_t Disk::getBlockSize ( )
```

Returns

the blocksize of the Disk

4.12.2.3 readDiskBlock()

```
void Disk::readDiskBlock (
            BlkNumType blknum,
            char * blkdata )
```

Reads a block from the Disk.

Parameters

blknum	the blocknumber to be read
blkdata	the buffer to put the read data. must be large enough to contain an entire block of data

See also

PartitionManger::readDiskBlock() ParitionManager::readDiskBlock()

4.12.2.4 writeDiskBlock()

```
void Disk::writeDiskBlock (
            BlkNumType blknum,
            char * blkdata )
```

Writes a block to the Disk.

Parameters

blknum	the blocknumber to be written
blkdata	the buffer to write the data from. It Will write an entire block size of data.

See also

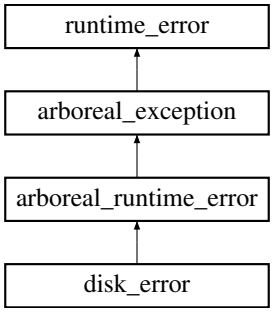
PartitionManger::writeDiskBlock() ParitionManager::writeDiskBlock()

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

- Filesystem/DaemonDependancies/Disk/Disk.h
- Filesystem/DaemonDependancies/Disk/Disk.cpp

4.13 disk_error Class Reference

Inheritance diagram for disk_error:



Public Member Functions

- **disk_error** (const char *what, const char *where, const int ecode=99)
- **disk_error** (const char *what, const string &where, const int ecode=99)
- **disk_error** (const string &what, const string &where, const int ecode=99)
- **disk_error** (const string &what, const char *where, const int ecode=99)

Additional Inherited Members

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

- SharedHeaders/Arboreal_Exceptions.h
- SharedCPPFiles/Arboreal_Exceptions.cpp

4.14 DiskManager Class Reference

Public Member Functions

- DiskManager (Disk *d)

Accessor Functions

- void readDiskBlock (string partitionName, BlkNumType blknum, char *blkdata)
- size_t getBlockSize ()
- BlkNumType getPartitionSize (string partitionName)
- DiskPartition * findPart (string partitionName)

Modifier Functions

- void writeDiskBlock (string partitionName, BlkNumType blknum, char *blkdata)

4.14.1 Constructor & Destructor Documentation

4.14.1.1 DiskManager()

```
DiskManager::DiskManager (
            Disk * d )
```

Parameters

d	Pointer to the Disk this will manage

4.14.2 Member Function Documentation

4.14.2.1 findPart()

```
DiskPartition * DiskManager::findPart (
            string partitionName )
```

Parameters

partitionName	the name of the partition

Returns

the size of a partition in blocks

4.14.2.2 getBlockSize()

```
size_t DiskManager::getBlockSize ( )
```

Returns

the blocksize of the Disk

4.14.2.3 getPartitionSize()

```
BlkNumType DiskManager::getPartitionSize (
            string partitionName )
```

Parameters

partitionName	the name of the partition

Returns

the size of a partition in blocks

4.14.2.4 readDiskBlock()

```
void DiskManager::readDiskBlock (
            string partitionName,
            BlkNumType blknum,
            char * blkdata )
```

Reads a block from the Disk.

Parameters

partitionName	the name of the partition to write the block to
blknum	the blocknumber to be read
blkdata	the buffer to put the read data. must be large enough to contain an entire block of data

See also

PartitionManger::readDiskBlock() ParitionManager::readDiskBlock()

4.14.2.5 writeDiskBlock()

```
void DiskManager::writeDiskBlock (
            string partitionName,
            BlkNumType blknum,
            char * blkdata )
```

Writes a block to the Disk.

Parameters

partitionName	the name of the partition to write the block to
blknum	the blocknumber to be written
blkdata	the buffer to write the data from. It Will write an entire block size of data.

See also

PartitionManger::writeDiskBlock() ParitionManager::writeDiskBlock()

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

- Filesystem/DaemonDependancies/DiskManager/DiskManager.h
- Filesystem/DaemonDependancies/DiskManager/DiskManager.cpp

4.15 DiskPartition Struct Reference

Public Attributes

- string **partitionName**

- BlkNumType **partitionSize**
- BlkNumType **partitionBlkStart**
- int **fileNameSize**

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

- Filesystem/DaemonDependancies/DiskManager/DiskManager.h

4.16 File Class Reference

Public Member Functions

- File (string name, const vector< string > &tags, FileAttributes attributes)

Accessor Functions

- string get_name ()
- vector< string > & get_tags ()
- FileAttributes get_attributes ()

Static Public Member Functions

- static File * read_buff (const char *serializedFile)

4.16.1 Constructor & Destructor Documentation

4.16.1.1 File()

```
File::File (
            string name,
            const vector< string > & tags,
            FileAttributes attributes )
```

Parameters

name	the name of the File
tags	the tags to be associated with the File
attributes	the File attributes

4.16.2 Member Function Documentation

4.16.2.1 get_attributes()

```
FileAttributes File::get_attributes ( )
```

Returns

the attributes associated with this File

4.16.2.2 get_name()

```
string File::get_name ( )
```

Returns

The name of the File

4.16.2.3 get_tags()

```
vector< string > & File::get_tags ( )
```

Returns

The tags associated with this File

4.16.2.4 read_buff()

```
File * File::read_buff (
            const char * serializedFile )  [static]
```

Will take a char* buffer and create a File object from it. The buffer must have been serialized in the correct format

Parameters

serializedFile	the serializedFile object

Returns

a File* to the created File

See also

FileInfo::serialize()

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

- Filesystem/DaemonDependancies/File/File.h
- Filesystem/DaemonDependancies/File/File.cpp

4.17 file_attributes Struct Reference

Public Attributes

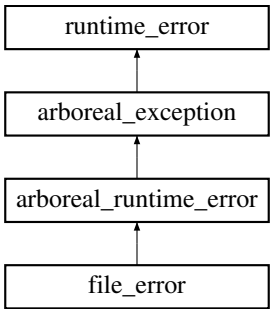
- time_t **creationTime**
- time_t **lastAccess**
- time_t **lastEdit**
- size_t **size**
- char **permissions** [12]
- int **owner**

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

- Filesystem/DaemonDependancies/Types/types.h

4.18 file_error Class Reference

Inheritance diagram for file_error:



Public Member Functions

- **file_error** (const char ∗what, const char ∗where, const int ecode=99)
- **file_error** (const char ∗what, const string &where, const int ecode=99)
- **file_error** (const string &what, const string &where, const int ecode=99)
- **file_error** (const string &what, const char ∗where, const int ecode=99)

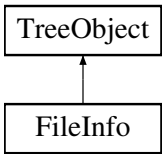
Additional Inherited Members

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

- SharedHeaders/Arboreal_Exceptions.h
- SharedCPPFiles/Arboreal_Exceptions.cpp

4.19 FileInfo Class Reference

Inheritance diagram for FileInfo:



Public Member Functions

- FileInfo (string filename, BlkNumType blknum, PartitionManager ∗pm)
- void write_out ()
- void read_in (unordered_multimap< string, FileInfo ∗> ∗allFiles, RootTree ∗rootTree)
- void erase (string name)
- void insert (string name, TreeObject ∗ptr)
- void del ()
- void delete_cont_blocks (BlkNumType blknum)
- void insert_addition (TreeObject ∗add)
- void insert_deletion (TreeObject ∗del)

Accessor Functions

- string mangle ()
  mangles the filename with its tags
- string mangle (vector< string > &tags)
  mangles the filename with the specified tags
- string mangle (unordered_set< string > &tags)
  mangles the filename with the specified tags
- Finode get_finode ()
- size_t get_file_size ()
- Attributes ∗ get_attributes ()
- FileAttributes **get_file_attributes** ()
- unordered_set< string > get_tags ()
- vector< string > **get_vec_tags** ()

Modifier Functions

- void add_direct_block (BlkNumType blknum, int index)
- void add_indirect_block (BlkNumType blknum, short level)
- void update_file_size (size_t bytes)
- void set_access ()
- void set_edit ()
- void set_permissions (char ∗perms)
  sets the permisssions for this file

Static Public Member Functions

- static string * serialize (FileInfo *file)

Additional Inherited Members

4.19.1 Constructor & Destructor Documentation

4.19.1.1 FileInfo()

```
FileInfo::FileInfo (
            string filename,
            BlkNumType blknum,
            PartitionManager * pm )
```

Parameters

filename	Name of the File
blknum	the blocknumber of the associated Finode on disk

4.19.2 Member Function Documentation

4.19.2.1 add_direct_block()

```
void FileInfo::add_direct_block (
            BlkNumType blknum,
            int index )
```

adds the specified blocknumber to the array of direct blocks in this file's Finode

Parameters

blknum	the block number of the direct block that has already been allocated
index	the index of the blknum in the array, must be less than 12 and at least 0.

Exceptions

arboreal_logic_error	index out of bounds

See also

add_indirect_block

4.19.2.2 add_indirect_block()

```
void FileInfo::add_indirect_block (
            BlkNumType blknum,
            short level )
```

adds the specified blocknumber to the Finode as the start of the specified level of indirect blocks

Parameters

blknum	the block number of the indirect block that has already been allocated
level	the level that the block number is associated with. must be 1, 2 or 3.

Exceptions

arboreal_logic_error	Invalid level

See also

add_direct_block

4.19.2.3 del()

```
void FileInfo::del ( )  [virtual]
```

Will completely remove the TreeObject's presence on disk

Implements TreeObject.

4.19.2.4 delete_cont_blocks()

```
void FileInfo::delete_cont_blocks (
            BlkNumType blknum )  [virtual]
```

Will follow the chain of continuation blocks and free all of them

Parameters

blknum	will free the blknum and use it to follow the chain of continuation blocks

Reimplemented from TreeObject.

4.19.2.5 erase()

```
void FileInfo::erase (
            string name )  [virtual]
```

Disassociate the given name from this object

Parameters

name	the name of the object to be erased.

Exceptions

arboreal_logic_error	

Reimplemented from TreeObject.

4.19.2.6 get_attributes()

```
Attributes * FileInfo::get_attributes ( )
```

Returns

the Attributes accociated with this file

4.19.2.7 get_file_size()

```
size_t FileInfo::get_file_size ( )
```

Returns

the size of this file in bytes

4.19.2.8 get_finode()

```
Finode FileInfo::get_finode ( )
```

Returns

the Finode associated with this file

4.19.2.9 get_tags()

```
unordered_set< string > FileInfo::get_tags ( )
```

Returns

The tags associated with this file

4.19.2.10 insert()

```
void FileInfo::insert (
            string name,
            TreeObject * obj )  [virtual]
```

Associate a TreeObject with this object

Parameters

name	name of the object, mangled if inserting a FileInfo
obj	the object to be inserted

Exceptions

tag_error	

See also

FileInfo::insert()

Reimplemented from TreeObject.

4.19.2.11 insert_addition()

```
void FileInfo::insert_addition (
            TreeObject * add )  [virtual]
```

Do not call on FileInfo

Reimplemented from TreeObject.

4.19.2.12 insert_deletion()

```
void FileInfo::insert_deletion (
            TreeObject * del )  [virtual]
```

Do not call on FileInfo

Reimplemented from TreeObject.

4.19.2.13 mangle() [1/3]

```
string FileInfo::mangle ( )
```

mangles the filename with its tags

The name is mangled as follows: Each tag is placed in alphabetical order and appended to the filename using '_' as the seperator.

Returns

the mangled name of this file.

See also

mangle(vector<string>&) mangle(unordered_set<string>& tags)

4.19.2.14 mangle() [2/3]

```
string FileInfo::mangle (
            vector< string > & tags )
```

mangles the filename with the specified tags

The name is mangled as follows: Each tag is placed in alphabetical order and appended to the filename using '_' as the seperator.

Returns

the mangled name of this file.

Parameters

tags	the tags you wish to mangle the filename with

See also

mangle() mangle(unordered_set<string>& tags)

4.19.2.15 mangle() [3/3]

```
string FileInfo::mangle (
            unordered_set< string > & tags )
```

mangles the filename with the specified tags

) The name is mangled as follows: Each tag is placed in alphabetical order and appended to the filename using '_' as the seperator.

Returns

the mangled name of this file.

Parameters

tags	the tags you wish to mangle the filename with

See also

mangle() mangle(unordered_set<string>& tags

4.19.2.16 read_in()

```
void FileInfo::read_in (
            unordered_multimap< string, FileInfo *> * allFiles,
            RootTree * rootTree )  [virtual]
```

Will read in all object data from disk

Parameters

allFiles	a pointer to the map of all files
rootTree	a pointer to the root tree

Implements TreeObject.

4.19.2.17 serialize()

```
string * FileInfo::serialize (
            FileInfo * file )  [static]
```

Will serialize a FileInfo object such that it can be read in as a File object

Parameters

file	the FileInfo object to be serialized

Returns

The serialized object in string form

See also

File::read_buff()

4.19.2.18 set_access()

```
void FileInfo::set_access ( )
```

marks the file as accessed at the current UNIX time

4.19.2.19 set_edit()

```
void FileInfo::set_edit ( )
```

marks the file as edited at the current UNIX time

4.19.2.20 set_permissions()

```
void FileInfo::set_permissions (
            char * perms )
```

sets the permisssions for this file

The permisssions format is as follows. a 1 for true 0 false Byte 0, 1, 2 : reserved, for now Byte 3 - 5 : read write and execute permisssions for the user Byte 6 - 8 : read write and execute permisssions for the group Byte 9 - 11 : read write and execute permisssions for the world Currently there is no differentiation between user group and world

Parameters

perms	the permisssions in the correct format

4.19.2.21 update_file_size()

```
void FileInfo::update_file_size (
            size_t bytes )
```

Sets the file size to the specified bytes. Only the filesystem should call.

Parameters

bytes	the new file size

4.19.2.22 write_out()

```
void FileInfo::write_out ( )  [virtual]
```

Intended to write out the object to disk

Implements TreeObject.

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

- Filesystem/DaemonDependancies/Trees/Trees.h
- Filesystem/DaemonDependancies/Trees/Trees.cpp

4.20 FileOpen Class Reference

Public Member Functions

- **FileOpen** (FileInfo *file, char mode, PartitionManager *pm)
- FileInfo * **get_file** ()
- size_t **get_seek** ()
- char **get_mode** ()
- void **increment_seek** (size_t bytes, bool write=false)
- void **decrement_seek** (size_t bytes)
- Index **byte_to_index** (short offset)
- Index **increment_index** ()
- void **set_EOF** ()
- void **reset_seek** ()
- bool **get_EOF** ()
- void **go_past_last_byte** ()
- void **refresh** ()

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

- Filesystem/DaemonDependancies/FileSystem/FileSystem.h
- Filesystem/DaemonDependancies/FileSystem/FileSystem.cpp

4.21 FileSystem Class Reference

Public Member Functions

- FileSystem (DiskManager *dm, string partitionName)
- void write_changes ()
- FileInfo * path_to_file (vector< string > &path)
- int get_file_name_size ()
- size_t **num_of_files** ()
- size_t **num_of_tags** ()

Tag Operations

- vector< FileInfo * > * tag_search (unordered_set< string > &tags)
- void create_tag (string tagName)
- void delete_tag (string tagName)
- void merge_tags (string tag1, string tag2)
- void tag_file (FileInfo *file, unordered_set< string > tagsToAdd)
- void tag_file (vector< string > &filePath, unordered_set< string > tags)
- void untag_file (FileInfo *file, unordered_set< string > tagsToRemove, bool deleting=false)
- void untag_file (vector< string > &filePath, unordered_set< string > tags)
- void rename_tag (string originalTagName, string newTagName)

File Operations

- vector< FileInfo * > * file_search (string name)
- FileInfo * create_file (string filename, unordered_set< string > &tags)
- int open_file (vector< string > &filePath, char mode)
- void close_file (unsigned int fileDesc)
- size_t read_file (unsigned int fileDesc, char *data, size_t len)
- size_t write_file (unsigned int fileDesc, const char *data, size_t len)
- size_t append_file (unsigned int fileDesc, const char *data, size_t len)
- void seek_file_absolute (unsigned int fileDesc, size_t offset)
- void seek_file_relative (unsigned int fileDesc, long int offset)
- void rename_file (vector< string > &originalFilePath, string newFileName)
- Attributes * get_attributes (vector< string > &filePath)
- void set_permissions (vector< string > &filePath, char *perms)
- void delete_file (FileInfo *file)
- void delete_file (vector< string > &filePath)

Debug Functions

- void print_root ()
- void print_tags ()
- void print_files ()

4.21.1 Constructor & Destructor Documentation

4.21.1.1 FileSystem()

```
FileSystem::FileSystem (
            DiskManager * dm,
            string partitionName )
```

Parameters

dm	the Disk manager for the disk that this Filesystem will be accessing
partitionName	the name of the partition that this FileSystem will be associated with

4.21.2 Member Function Documentation

4.21.2.1 append_file()

```
size_t FileSystem::append_file (
            unsigned int fileDesc,
            const char * data,
            size_t len )
```

Will Append a number of bytes to an open file. The file must have been opened with write permissions.

Parameters

fileDesc	the file descriptor returned from open_file
data	a buffer to be read from to write to the file. must be at least of len size
len	the number of bytes to write from data.

4.21.2.2 close_file()

```
void FileSystem::close_file (
            unsigned int fileDesc )
```

Will close a file. the File must have been opened.

Parameters

fileDesc	the file descriptor returned from open_file

4.21.2.3 create_file()

```
FileInfo * FileSystem::create_file (
            string filename,
            unordered_set< string > & tags )
```

Will create a new file with the specified name and tags. The new file must not already exist.

Parameters

filename	the name of the new file
tags	the tag set to tag the file with. If empty, will be tagged with default.

Returns

a FileInfo to the created file, in case the calling code needs it

4.21.2.4 create_tag()

```
void FileSystem::create_tag (
            string tagName )
```

Will create a new tag if that tag name does not already exist

Parameters

tagName	the name of the Tag to create

4.21.2.5 delete_file() [1/2]

```
void FileSystem::delete_file (
            FileInfo * file )
```

Delete a particular file. The file must exist.

Parameters

file	the FileInfo object to be deleted.

See also

delete_file(vector<string>&)

4.21.2.6 delete_file() [2/2]

```
void FileSystem::delete_file (
            vector< string > & filePath )
```

Delete a particular file. The file must exist.

Parameters

filePath	the full path to the file to you wish to delete

See also

delete_file(FileInfo*)

4.21.2.7 delete_tag()

```
void FileSystem::delete_tag (
            string tagName )
```

Will delete the specified tag only if it has no files associated with it(it is empty) and it does in fact exist.

Parameters

tagName	the name of the tag to be deleted

4.21.2.8 file_search()

```
vector< FileInfo * > * FileSystem::file_search (
            string name )
```

Will search for a specified file name. Searches the entire FileSystem

Parameters

name	the name of the file to search for.

Returns

a pointer to a vector of FileInfo objects that have the specified name. This should be freed by the calling code

4.21.2.9 get_attributes()

```
Attributes * FileSystem::get_attributes (
            vector< string > & filePath )
```

Will search for a file and return its Attributes

Parameters

filePath	the full path to the file to you wish to get the Attributes of

Returns

the Attributes object associated with a particular file.

4.21.2.10 get_file_name_size()

```
int FileSystem::get_file_name_size ( )
```

Returns

the Maximum file name size for the partition associated with this FileSystem object

4.21.2.11 merge_tags()

```
void FileSystem::merge_tags (
            string tag1,
            string tag2 )
```

TODO: description and Function

Parameters

tag1	
tag	2

4.21.2.12 open_file()

```
int FileSystem::open_file (
            vector< string > & filePath,
            char mode )
```

Will open a file. The file must exist. There is a cap on the Maximum number of open files. You can open the same file as many times as you want.

Parameters

filePath	the full path including the file name as the last entry
mode	the mode to open the file with. r, w, or x. x is read and write ability.

Returns

a file descriptor that can later be used to reference the opened file

4.21.2.13 path_to_file()

```
FileInfo * FileSystem::path_to_file (
            vector< string > & path )
```

Will find a FileInfo object if it exists, given the full path

Parameters

path	The full path to the file. The filename must be the last entry in the vector. an file name with no path is considered to be in the default path

Returns

the found FileInfo object

4.21.2.14 print_files()

```
void FileSystem::print_files ( )
```

Print out all files and their blocknumbers

4.21.2.15 print_root()

```
void FileSystem::print_root ( )
```

Print out the root Tree

4.21.2.16 print_tags()

```
void FileSystem::print_tags ( )
```

Print out all the tag trees and their contents

4.21.2.17 read_file()

```
size_t FileSystem::read_file (
            unsigned int fileDesc,
            char * data,
            size_t len )
```

Will read a number of bytes from an open file. The file must have been opened with read permissions. If you read past the end of the file, EOF will be tripped and you cannot continue reading. will return all the data up to that point

Parameters

fileDesc	the file descriptor returned from open_file
data	a buffer to store the read data must be at least len size
len	the number of bytes to read.

4.21.2.18 rename_file()

```
void FileSystem::rename_file (
            vector< string > & originalFilePath,
            string newFileName )
```

Will rename a file. The new file must not already exist in the emulated directory

Parameters

originalFilePath	the full path to the file to be renamed
newFileName	the name that the file will be renamed to.

4.21.2.19 rename_tag()

```
void FileSystem::rename_tag (
            string originalTagName,
            string newTagName )
```

Will rename the tag. The tag must exist. The new tag name must already exist. This is a slow operation.

Parameters

originalTagName	the name of the tag to be renamed
newTagName	the new tag name for that tag

4.21.2.20 seek_file_absolute()

```
void FileSystem::seek_file_absolute (
            unsigned int fileDesc,
            size_t offset )
```

Seek to an absolute position in the file. Will trip EOF if the offset is larger than the file size. The posistion in the file is indexed at 1.

Parameters

fileDesc	the file descriptor returned from open_file
offset	the absolute position in the file to seek to.

4.21.2.21 seek_file_relative()

```
void FileSystem::seek_file_relative (
            unsigned int fileDesc,
            long int offset )
```

Seek to a relative position in the file. Will trip EOF if you try to seek too far in a direction. The posistion in the file is indexed at 1.

Parameters

fileDesc	the file descriptor returned from open_file
offset	the relative position in the file to seek to. may be a negative number.

4.21.2.22 set_permissions()

```
void FileSystem::set_permissions (
            vector< string > & filePath,
            char * perms )
```

Set the permissions for a file. The format is defined in FileInfo.

Parameters

filePath	the full path to the file to you wish to get the Attributes of
perms	the permissions following the correct format to set to this file

See also

FileInfo::set_permissions()

4.21.2.23 tag_file() [1/2]

```
void FileSystem::tag_file (
            FileInfo * file,
            unordered_set< string > tagsToAdd )
```

Will tag a file with the specified tags. If some or all of the tags do not exist, a warning is printed and the operation continues. The file must exist. The file that would be created by adding tags must not already exist.

Parameters

file	the FileInfo* that will be tagged with the specified tags
tagsToAdd	the tags that will be added to the file's tag set

See also

tag_file(vector<string>&, unordered_set<string>)

4.21.2.24 tag_file() [2/2]

```
void FileSystem::tag_file (
            vector< string > & filePath,
            unordered_set< string > tags )
```

An alternate way to tag a file using a file path instead.Will tag a file with the specified tags. If some or all of the tags do not exist, a warning is printed and the operation continues. The file must exist. The file that would be created by adding tags must not already exist.

Parameters

filePath	the FileInfo* that will be tagged with the specified tags
tagsToAdd	the tags that will be added to the file's tag set

See also

tag_file(FileInfo*, unordered_set<string>)

4.21.2.25 tag_search()

```
vector< FileInfo * > * FileSystem::tag_search (
            unordered_set< string > & tags )
```

Search for files by tags. The tag search is an "and" operation, meaning the files returned will have at least all the specified tags.

Parameters

tags	the tags that the files will be tagged with in the return vector

Returns

a pointer to a vector of the FileInfo objects which then can be serialized. The returned vector should be freed by the calling code

4.21.2.26 untag_file() [1/2]

```
void FileSystem::untag_file (
            FileInfo * file,
            unordered_set< string > tagsToRemove,
            bool deleting = false )
```

Will remove tags associated with the specified file. The tags must exist. The file must exist. The file that would be created by removing tags must not already exist.

Parameters

file	the FileInfo* that will be untagged with the specified tags
tagsToRemove	the tags that will be removed from the file's tag set
deleting	this is a tag only used by the FileSystem itself for deleting a file

See also

tag_file(FileInfo*, unordered_set<string>)

4.21.2.27 untag_file() [2/2]

```
void FileSystem::untag_file (
            vector< string > & filePath,
            unordered_set< string > tags )
```

Will remove tags associated with the specified file. The tags must exist. The file must exist. The file that would be created by removing tags must not already exist.

Parameters

file	the FileInfo* that will be untagged with the specified tags
tagsToRemove	the tags that will be removed from the file's tag set
deleting	this is a tag only used by the FileSystem itself for deleting a file

See also

tag_file(FileInfo*, unordered_set<string>)

4.21.2.28 write_changes()

```
void FileSystem::write_changes ( )
```

Since the FileSystem is journaling. The changes to tag trees and the Root tree are only written out when this is called. File Operations are not journaled.

4.21.2.29 write_file()

```
size_t FileSystem::write_file (
            unsigned int fileDesc,
            const char * data,
            size_t len )
```

Will write a number of bytes to an open file. The file must have been opened with write permissions. You can write past the EOF with no problems.

Parameters

fileDesc	the file descriptor returned from open_file
data	a buffer to be read from to write to the file. must be at least of len size
len	the number of bytes to write from data.

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

- Filesystem/DaemonDependancies/FileSystem/FileSystem.h
- Filesystem/DaemonDependancies/FileSystem/FileSystem.cpp

4.22 finode Struct Reference

Public Attributes

- BlkNumType **attributes**
- BlkNumType **directBlocks** [12]
- BlkNumType **level1Indirect**
- BlkNumType **level2Indirect**
- BlkNumType **level3Indirect**

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

- Filesystem/DaemonDependancies/Types/types.h

4.23 index Struct Reference

Public Attributes

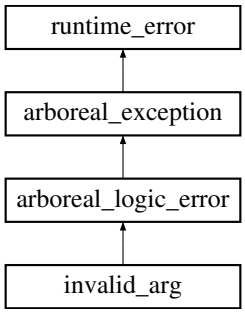
- BlkNumType **blknum**
- size_t **offset**

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

- Filesystem/DaemonDependancies/Types/types.h

4.24 invalid_arg Class Reference

Inheritance diagram for invalid_arg:



Public Member Functions

- **invalid_arg** (const char ∗what, const char ∗where, const int ecode=99)
- **invalid_arg** (const char ∗what, const string &where, const int ecode=99)
- **invalid_arg** (const string &what, const string &where, const int ecode=99)
- **invalid_arg** (const string &what, const char ∗where, const int ecode=99)

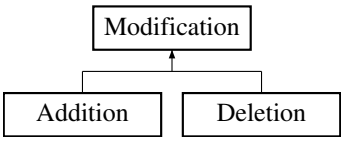
Additional Inherited Members

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

- SharedHeaders/Arboreal_Exceptions.h
- SharedCPPFiles/Arboreal_Exceptions.cpp

4.25 Modification Class Reference

Inheritance diagram for Modification:



Public Member Functions

- virtual void **write_out** (PartitionManager ∗pm)=0

Protected Member Functions

- **Modification** (TreeObject ∗obj, TreeObject ∗parent)

Protected Attributes

- TreeObject * **_mod**
- TreeObject * **_parent**

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

- Filesystem/DaemonDependancies/Trees/Trees.h
- Filesystem/DaemonDependancies/Trees/Trees.cpp

4.26 ParseError Class Reference

Public Member Functions

- ParseError (const char *where, const char *what)
- std::string where ()
- std::string what ()

4.26.1 Constructor & Destructor Documentation

4.26.1.1 ParseError()

```
ParseError::ParseError (
            const char * where,
            const char * what )  [inline]
```

Parameters

where	Where the parse error took place
what	What the parse error consisted of

4.26.2 Member Function Documentation

4.26.2.1 what()

```
std::string ParseError::what ( )  [inline]
```

Returns

A std::string detailing what the parse error consisted of

4.26.2.2 where()

```
std::string ParseError::where ( )  [inline]
```

Returns

A std::string detailing where the parse error occured

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

- SharedHeaders/Parser.h

4.27 Parser Class Reference

Public Member Functions

- Parser (char *buffer, char *cwd, int max_name_size)
- Parser (std::string string, std::string cwd, int max_name_size)
- Parser (const char *string_lit, const char *cwd, int max_name_size)
- Parser ()
- ~Parser ()
- void reset (std::string string, std::string cwd="")

  Changes the member _string of the parser class to whatever is passed.
- void reset (char *buffer, char *cwd=NULL)

  Changes the member _string of the parser class to whatever is passed.
- void reset (const char *string_lit, const char *cwd="")

  Changes the member _string of the parser class to whatever is passed.
- void set_max_name_size (int size)

  Sets the maximum allowed file and tagname size that the Parser will use.
- void set_cwd (std::string cwd)

  Sets the Current Working Directory that the Parser will use.
- std::vector< std::string > parse (int type)

  Parse a string based on a certain rule.
- std::vector< std::string > get_cwd_tags ()

  Returns a vector representation of the current working directory.

Static Public Member Functions

- static std::vector< std::string > split_on_delim (std::string string, char delim)

  Splits a string at each instance of a particualar char (the delimeter)

4.27.1 Constructor & Destructor Documentation

4.27.1.1 Parser() [1/4]

```
Parser::Parser (
            char * buffer,
            char * cwd,
            int max_name_size )
```

Parameters

buffer	A C-Style String representation of the string to be parsed
cwd	A C-Style String representation of the current working directory; (This value is typically provided by the Liaison process). The directory string is used to parse commands which act within directories only thus providing commands such as 'tag' a "path" to the file(s) which will be tagged without the user having to explicitly enter those file's entire paths themselves.
max_name_size	The maximum length that a file or tagname is allowed to have; (This value is typically provided by the Liaison process)

4.27.1.2 Parser() [2/4]

```
Parser::Parser (
            std::string string,
            std::string cwd,
            int max_name_size )
```

Parameters

buffer	A std::string representation of the string to be parsed
cwd	A std::string representation of the current working directory; (This value is typically provided by the Liaison process). The directory string is used to parse commands which act within directories only thus, providing commands such as 'tag' a "path" to the file(s) which will be tagged without the user having to explicitly enter those file's entire paths themselves.
max_name_size	The maximum length that a file or tagname is allowed to have; (This value is typically provided by the Liaison process)

4.27.1.3 Parser() [3/4]

```
Parser::Parser (
            const char * string_lit,
            const char * cwd,
            int max_name_size )
```

Parameters

buffer	A String Literal representation of the string to be parsed
cwd	A String Literal representation of the current working directory; (This value is typically provided by the Liaison process). The directory string is used to parse commands which act within directories only thus, providing commands such as 'tag' a "path" to the file(s) which will be tagged without the user having to explicitly enter those file's entire paths themselves.
max_name_size	The maximum length that a file or tagname is allowed to have; (This value is typically provided by the Liaison process)

4.27.1.4 Parser() [4/4]

```
Parser::Parser ( )
```

Default Constructor to be used in case initialization of values needs to be done elsewhere

4.27.1.5 ~Parser()

```
Parser::~Parser ( )
```

Default Destructor; Does nothing

4.27.2 Member Function Documentation

4.27.2.1 get_cwd_tags()

```
std::vector< std::string > Parser::get_cwd_tags ( )
```

Returns a vector representation of the current working directory.

That is, it will decompose '/string1/string2' into a vector containing [string1, string2]. This is useful when the calling code requires the current working directory as a vector of strings rather than as a standard string representation.

Returns

A std::vector of std::string comprised of the non-'/' parts of the Parser member value _cwd

4.27.2.2 parse()

```
std::vector< std::string > Parser::parse (
            int type )
```

Parse a string based on a certain rule.

The rule generally corresponds to how a CLI command should be decomposed.
For example the CLI command for finding files takes a list of files, hower the filesystem itself does not support batch commands, therefore, the Parser will decompose the command into its constituent parts (i.e. a single file).
This particular behavior is access by passing '8' as the "type" of decomposition that needs to take place (Note that this corresponds to the command's ID).
However the Parser can be extended to support any rule whatsoever, so long as it is added to the Parser's parse() function switch statement.

Parameters

type	The integer identification of the parse rule that will be executed

Returns

A std::vector of std::string comprised of the result after the chosen parse rule is executed.

4.27.2.3 reset() [1/3]

```
void Parser::reset (
            std::string string,
            std::string cwd = "" )
```

Changes the member _string of the parser class to whatever is passed.

The Parser class conducts all operations on its member _string rather than requiring that a string value be passed to its parse() method. This was done in order to make use of the class as streamlined as possible.

Parameters

string	A std::string representation of the string to be parsed
cwd	A std::string representation of the current working directory; Note that this argument is optional and allows the user to both reset the string the Parser will work with as well as the directory string the Parser will use. The directory string is used to parse commands which act within directories only thus providing commands such as 'tag' a "path" to the file(s) which will be tagged without the user having to explicitly enter those file's entire paths themselves.

4.27.2.4 reset() [2/3]

```
void Parser::reset (
            char * buffer,
            char * cwd = NULL )
```

Changes the member _string of the parser class to whatever is passed.

The Parser class conducts all operations on its member _string rather than requiring that a string value be passed to its parse() method. This was done in order to make use of the class as streamlined as possible.

Parameters

string	A C-Style String representation of the string to be parsed
cwd	A C-Style String representation of the current working directory; Note that this argument is optional and allows the user to both reset the string the Parser will work with as well as the directory string the Parser will use. The directory string is used to parse commands which act within directories only thus providing commands such as 'tag' a "path" to the file(s) which will be tagged without the user having to explicitly enter those file's entire paths themselves.

Returns

Void

4.27.2.5 reset() [3/3]

```
void Parser::reset (
            const char * string_lit,
            const char * cwd = "" )
```

Changes the member _string of the parser class to whatever is passed.

The Parser class conducts all operations on its member _string rather than requiring that a string value be passed to its parse() method. This was done in order to make use of the class as streamlined as possible.

Parameters

string	A String Literal representation of the string to be parsed
cwd	A String Literal representation of the current working directory; Note that this argument is optional and allows the user to both reset the string the Parser will work with as well as the directory string the Parser will use. The directory string is used to parse commands which act within directories only thus providing commands such as 'tag' a "path" to the file(s) which will be tagged without the user having to explicitly enter those file's entire paths themselves.

Returns

Void

4.27.2.6 set_cwd()

```
void Parser::set_cwd (
            std::string cwd )
```

Sets the Current Working Directory that the Parser will use.

The directory string is used to parse commands which act within directories only thus providing commands such as 'tag' a "path" to the file(s) which will be tagged without the user having to explicitly enter those file's entire paths themselves. This function does not have counterparts which tahe C-Style Strings or String Literals. This is because, in all situations, if the current working directory must be set using this method, it is highly likely that the calling code has a std::string representation of the current working directory rather than a representation in one of the other formats. If such functionality (C-Style Strings and others) is desired, extensibility is easy enough. Regardless the Parser's _cwd member will always be a std::string.

Parameters

cwd	A std::string representation of the current working directory

Returns

Void

4.27.2.7 set_max_name_size()

```
void Parser::set_max_name_size (
            int size )
```

Sets the maximum allowed file and tagname size that the Parser will use.

If this size is exceeded an error is thrown and the Parser will stop its current activities. This value is dictated by the CLI and is generally provided to the Parser by the Liaison Process.

Parameters

size	The maximum file/tag name length

4.27.2.8 split_on_delim()

```
std::vector< std::string > Parser::split_on_delim (
            std::string string,
            char delim )  [static]
```

Splits a string at each instance of a particualar char (the delimeter)

The delimeters are NOT included anywhere in the resulting vector. This function is static and is mainly used outside the Parser in order to split values that the parser returned. This can happen because the complexity of certain commands does not allow the parser to fully decompose the string and instead it can only reorganize the command into a form which can be easily split later. It is important to note that this function does not differentiate between the number of delimeter charcters the string contains. That is, it will read the whole string and split it at any point where the delimeter is seen whether it is seen in 1 or 100 places.

Parameters

string	A std::string representation of whatever string needs to be split
delim	A char value representing where the string should be split

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

- SharedHeaders/Parser.h
- SharedCPPFiles/Parser.cpp

4.28 PartitionManager Class Reference

Public Member Functions

- PartitionManager (DiskManager *dm, string partitionName)

Accessor Functions

- void readDiskBlock (BlkNumType blknum, char *blkdata)
- size_t getBlockSize ()
- string getPartitionName ()
- int get_file_name_size ()

Modifier Functions

- void writeDiskBlock (BlkNumType blknum, char *blkdata)
- BlkNumType getFreeDiskBlock ()
- void returnDiskBlock (BlkNumType blknum)

4.28.1 Constructor & Destructor Documentation

4.28.1.1 PartitionManager()

```
PartitionManager::PartitionManager (
            DiskManager * dm,
            string partitionName )
```

Parameters

dm	the DiskManager associated with this object
partitionName	the name of the partition that this will be managing

4.28.2 Member Function Documentation

4.28.2.1 get_file_name_size()

```
int PartitionManager::get_file_name_size ( )
```

Returns

The maximum file name size for this partition in bytes

4.28.2.2 getBlockSize()

```
size_t PartitionManager::getBlockSize ( )
```

Returns

the blocksize of the Disk

4.28.2.3 getFreeDiskBlock()

```
BlkNumType PartitionManager::getFreeDiskBlock ( )
```

Allocates a block on disk if there is a free one. The Disk free list is updated accordingly

Returns

the block number of the newly allocated block

4.28.2.4 getPartitionName()

```
string PartitionManager::getPartitionName ( )
```

Returns

The name of the partition this PartitionManager is associated with

4.28.2.5 readDiskBlock()

```
void PartitionManager::readDiskBlock (
            BlkNumType blknum,
            char * blkdata )
```

Reads a block from the Disk.

Parameters

blknum	the blocknumber to be read
blkdata	the buffer to put the read data. must be large enough to contain an entire block of data

4.28.2.6 returnDiskBlock()

```
void PartitionManager::returnDiskBlock (
            BlkNumType blknum )
```

returns a block to the Disk free list and zeros it out before writing.

Parameters

blknum	the blocknumber of the block to be freed

4.28.2.7 writeDiskBlock()

```
void PartitionManager::writeDiskBlock (
            BlkNumType blknum,
            char * blkdata )
```

Writes a block to the Disk.

Parameters

blknum	the blocknumber to be written
blkdata	the buffer to write the data from. It Will write an entire block size of data.

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

- Filesystem/DaemonDependancies/PartitionManager/PartitionManager.h
- Filesystem/DaemonDependancies/PartitionManager/PartitionManager.cpp

4.29 rootSuperBlock Struct Reference

Public Attributes

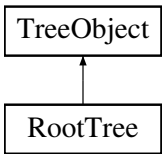
- size_t **size**
- Index **lastEntry**
- BlkNumType **startBlock**

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

- Filesystem/DaemonDependancies/Types/types.h

4.30 RootTree Class Reference

Inheritance diagram for RootTree:



Public Member Functions

- RootTree (PartitionManager *pm)
- void write_out ()
- void read_in (unordered_multimap< string, FileInfo *> *allFiles, RootTree *rootTree)
- void del ()

Additional Inherited Members

4.30.1 Constructor & Destructor Documentation

4.30.1.1 RootTree()

```
RootTree::RootTree (
            PartitionManager * pm )
```

Parameters

pm	the PartitionManager to be associated with the RootTree

4.30.2 Member Function Documentation

4.30.2.1 del()

```
void RootTree::del ( )  [virtual]
```

Will completely remove the TreeObject's presence on disk

Implements TreeObject.

4.30.2.2 read_in()

```
void RootTree::read_in (
            unordered_multimap< string, FileInfo *> * allFiles,
            RootTree * rootTree )  [virtual]
```

Will read in all object data from disk

Parameters

allFiles	a pointer to the map of all files
rootTree	a pointer to the root tree

Implements TreeObject.

4.30.2.3 write_out()

```
void RootTree::write_out ( )  [virtual]
```

Intended to write out the object to disk

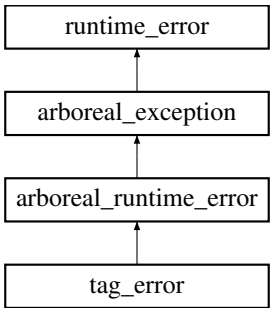
Implements TreeObject.

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

- Filesystem/DaemonDependancies/Trees/Trees.h
- Filesystem/DaemonDependancies/Trees/Trees.cpp

4.31 tag_error Class Reference

Inheritance diagram for tag_error:



Public Member Functions

- **tag_error** (const char *what, const char *where, const int ecode=99)
- **tag_error** (const char *what, const string &where, const int ecode=99)
- **tag_error** (const string &what, const string &where, const int ecode=99)
- **tag_error** (const string &what, const char *where, const int ecode=99)

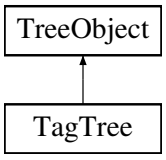
Additional Inherited Members

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

- SharedHeaders/Arboreal_Exceptions.h
- SharedCPPFiles/Arboreal_Exceptions.cpp

4.32 TagTree Class Reference

Inheritance diagram for TagTree:



Public Member Functions

- TagTree (string tagName, BlkNumType blknum, PartitionManager *pm)
- void write_out ()
- void read_in (unordered_multimap< string, FileInfo *> *allFiles, RootTree *rootTree)
- void del ()

Additional Inherited Members

4.32.1 Constructor & Destructor Documentation

4.32.1.1 TagTree()

```
TagTree::TagTree (
            string tagName,
            BlkNumType blknum,
            PartitionManager * pm )
```

Parameters

tagName	the name of this tag
blknum	the blocknumber for the superblock of this tagTree

4.32.2 Member Function Documentation

4.32.2.1 del()

```
void TagTree::del ( )  [virtual]
```

Will completely remove the TreeObject's presence on disk

Implements TreeObject.

4.32.2.2 read_in()

```
void TagTree::read_in (
            unordered_multimap< string, FileInfo *> * allFiles,
            RootTree * rootTree )  [virtual]
```

Will read in all object data from disk

Parameters

allFiles	a pointer to the map of all files
rootTree	a pointer to the root tree

Implements TreeObject.

4.32.2.3 write_out()

```
void TagTree::write_out ( )  [virtual]
```

Intended to write out the object to disk

Implements TreeObject.

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

- Filesystem/DaemonDependancies/Trees/Trees.h
- Filesystem/DaemonDependancies/Trees/Trees.cpp

4.33 tagTreeSuperBlock Struct Reference

Public Attributes

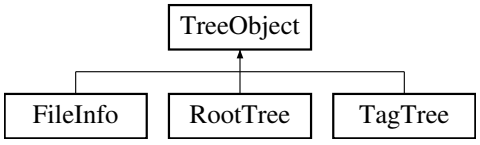
- size_t **size**
- Index **lastEntry**
- BlkNumType **startBlock**

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

- Filesystem/DaemonDependancies/Types/types.h

4.34 TreeObject Class Reference

Inheritance diagram for TreeObject:



Public Member Functions

- TreeObject (string name, BlkNumType blknum, PartitionManager *pm)

Accessor Functions

- string get_name () const
- BlkNumType get_block_number () const
- Index get_index (TreeObject *obj) const
- Index get_last_entry () const
- BlkNumType get_start_block () const
- size_t size () const
- unordered_map< string, TreeObject * >::iterator begin ()
- unordered_map< string, TreeObject * >::iterator end ()
- TreeObject * find (string name) const
- queue< Index > * get_free_spots ()

Modifier Functions

- void set_name (string name)
- void add_index (TreeObject *obj, Index index)
- void set_last_entry (Index index)
- virtual void insert (string name, TreeObject *obj)
- virtual void erase (string name)
- virtual void insert_addition (TreeObject *add)
- virtual void insert_deletion (TreeObject *del)

Disk Functions

- virtual void write_out ()=0
- virtual void read_in (unordered_multimap< string, FileInfo *> *allFiles, RootTree *rootTree)=0
- virtual void del ()=0
- void increment_allocate (Index *index)
- void increment_follow (Index *index)

Protected Member Functions

- virtual void delete_cont_blocks (BlkNumType blknum)

Protected Attributes

- queue< Modification * > _modifications

  A collection of associated Modifications.
- unordered_map< string, TreeObject * > _myTree

  A collection of contained TreeObjects.
- string _name

  name or value
- BlkNumType _blockNumber

  Blocknumber of the superblock on disk.
- unordered_map< TreeObject *, Index > _indeces

  location(s) of the superblock entry(ies) on disk
- Index _lastEntry

  Index of the last entry of data on disk.
- BlkNumType _startBlock

  blocknumber of the start of this data on disk
- PartitionManager * _myPartitionManager

  Associated PartitionManager.
- queue< Index > **_freeSpots**

4.34.1 Constructor & Destructor Documentation

4.34.1.1 TreeObject()

```
TreeObject::TreeObject (
            string name,
            BlkNumType blknum,
            PartitionManager * pm )
```

Parameters

name	name of this object
blknum	blocknumber of the superblock
pm	PartitionManager object to be associated with this object

4.34.2 Member Function Documentation

4.34.2.1 add_index()

```
void TreeObject::add_index (
            TreeObject * obj,
            Index index )
```

Add an index to _indeces for the specified TreeObject. If the index already existed. nothing happpens

Parameters

obj	the object that the Index references to
index	the Index of obj

4.34.2.2 begin()

```
unordered_map< string, TreeObject * >::iterator TreeObject::begin ( )
```

Returns

An iterator to the beginning of the TreeObjects associated with this object

4.34.2.3 del()

```
virtual void TreeObject::del ( )  [pure virtual]
```

Will completely remove the TreeObject's presence on disk

Implemented in RootTree, TagTree, and FileInfo.

4.34.2.4 delete_cont_blocks()

```
void TreeObject::delete_cont_blocks (
            BlkNumType blknum )  [protected], [virtual]
```

Will follow the chain of continuation blocks and free all of them

Parameters

blknum	will free the blknum and use it to follow the chain of continuation blocks

Reimplemented in FileInfo.

4.34.2.5 end()

```
unordered_map< string, TreeObject * >::iterator TreeObject::end ( )
```

Returns

An iterator to the end of the TreeObjects associated with this object

4.34.2.6 erase()

```
void TreeObject::erase (
            string name )  [virtual]
```

Disassociate the given name from this object

Parameters

name	the name of the object to be erased.

Exceptions

arboreal_logic_error	

Reimplemented in FileInfo.

4.34.2.7 find()

```
TreeObject * TreeObject::find (
            string name ) const
```

Search _myTree for the specified name

Parameters

name	the name of the desired object

Returns

a pointer to the object if found, 0 otherwise

4.34.2.8 get_block_number()

```
BlkNumType TreeObject::get_block_number ( ) const
```

Returns

The blocknumber of the superblock

4.34.2.9 get_free_spots()

```
queue< Index > * TreeObject::get_free_spots ( )
```

Returns

a pointer to the queue of empty spaces where new entries can be added

4.34.2.10 get_index()

```
Index TreeObject::get_index (
            TreeObject * obj ) const
```

Searches for obj and returns the Index of obj on disk, if found

Parameters

obj	object whose position is desired

Returns

The Index of obj on disk,

Exceptions

arboreal_logic_error	

4.34.2.11 get_last_entry()

```
Index TreeObject::get_last_entry ( ) const
```

Find the Index of the last entry for this object on disk

Returns

Index of the last entry on disk

4.34.2.12 get_name()

```
string TreeObject::get_name ( ) const
```

Returns

The name

4.34.2.13 get_start_block()

```
BlkNumType TreeObject::get_start_block ( ) const
```

Returns

The start block of data for this object

4.34.2.14 increment_allocate()

```
void TreeObject::increment_allocate (
            Index * index )
```

Will increment the Index passed and allocate blocks if necessary to do so

Parameters

index	the Index to be incremented

4.34.2.15 increment_follow()

```
void TreeObject::increment_follow (
            Index * index )
```

Will increment the Index passed but only follow the chain of already allocated blocks

Parameters

index	the Index to be incremented

4.34.2.16 insert()

```
void TreeObject::insert (
            string name,
            TreeObject * obj )  [virtual]
```

Associate a TreeObject with this object

Parameters

name	name of the object, mangled if inserting a FileInfo
obj	the object to be inserted

Exceptions

tag_error	

See also

FileInfo::insert()

Reimplemented in FileInfo.

4.34.2.17 insert_addition()

```
void TreeObject::insert_addition (
            TreeObject * add )  [virtual]
```

Add an Addition to the list of Modifications so that it can be written out later. Note: Do not call this on a FileInfo.

Parameters

add	the object that was previously inserted to this object which will be added to the list of Modifications

See also

FileSystem::write_out() TreeObject::insert()

Reimplemented in FileInfo.

4.34.2.18 insert_deletion()

```
void TreeObject::insert_deletion (
            TreeObject * del )  [virtual]
```

Add a Deletion to the list of Modifications so that it can be written out later. Note: Do not call this on a FileInfo.

Parameters

del	the object that was previously erased from this object which will be added to the list of Modifications

See also

FileSystem::write_out() TreeObject::erase()

Reimplemented in FileInfo.

4.34.2.19 read_in()

```
virtual void TreeObject::read_in (
            unordered_multimap< string, FileInfo *> * allFiles,
            RootTree * rootTree )  [pure virtual]
```

Will read in all object data from disk

Parameters

allFiles	a pointer to the map of all files
rootTree	a pointer to the root tree

Implemented in RootTree, TagTree, and FileInfo.

4.34.2.20 set_last_entry()

```
void TreeObject::set_last_entry (
            Index index )
```

Set the last Index for the last entry belonging to this object on disk

Parameters

index	The last Index

4.34.2.21 set_name()

```
void TreeObject::set_name (
            string name )
```

Set the name

Parameters

name	The new name

4.34.2.22 size()

```
size_t TreeObject::size ( ) const
```

Returns

The size of _myTree

4.34.2.23 write_out()

```
virtual void TreeObject::write_out ( )  [pure virtual]
```

Intended to write out the object to disk

Implemented in RootTree, TagTree, and FileInfo.

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

- Filesystem/DaemonDependancies/Trees/Trees.h
- Filesystem/DaemonDependancies/Trees/Trees.cpp

Index