

Gorgon Game Engine

Generated by Doxygen 1.8.6

Fri Mar 21 2014 10:25:17

Contents

1	Namespace Index	1
1.1	Namespace List	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	Namespace Documentation	7
4.1	Gorgon::Filesystem Namespace Reference	7
4.1.1	Detailed Description	8
4.1.2	Function Documentation	8
4.1.2.1	ApplicationDirectory	8
4.1.2.2	Canonize	8
4.1.2.3	ChangeDirectory	9
4.1.2.4	Copy	9
4.1.2.5	Copy	9
4.1.2.6	CreateDirectory	10
4.1.2.7	CurrentDirectory	10
4.1.2.8	Delete	10
4.1.2.9	EntryPoints	10
4.1.2.10	GetDirectory	10
4.1.2.11	Initialize	11
4.1.2.12	IsDirectory	11
4.1.2.13	IsExists	11
4.1.2.14	IsFile	11
4.1.2.15	IsHidden	11
4.1.2.16	IsWritable	12
4.1.2.17	Load	12
4.1.2.18	LocateResource	12
4.1.2.19	Move	13

4.1.2.20	Relative	13
4.1.2.21	Save	13
4.1.2.22	Size	13
4.1.2.23	StartupDirectory	14
4.1.2.24	swap	14
4.2	Gorgon::Threading Namespace Reference	14
4.2.1	Detailed Description	14
4.2.2	Function Documentation	14
4.2.2.1	RunAsync	14
4.2.2.2	RunInParallel	14
5	Class Documentation	17
5.1	Gorgon::Filesystem::EntryPoint Class Reference	17
5.1.1	Detailed Description	17
5.1.2	Constructor & Destructor Documentation	17
5.1.2.1	EntryPoint	17
5.1.3	Member Data Documentation	18
5.1.3.1	Name	18
5.1.3.2	Path	18
5.1.3.3	Readable	18
5.1.3.4	Writable	18
5.2	Gorgon::Filesystem::Iterator Class Reference	18
5.2.1	Detailed Description	19
5.2.2	Constructor & Destructor Documentation	19
5.2.2.1	Iterator	19
5.2.2.2	Iterator	19
5.2.2.3	Iterator	19
5.2.2.4	Iterator	20
5.2.2.5	~Iterator	20
5.2.3	Member Function Documentation	20
5.2.3.1	Current	20
5.2.3.2	Destroy	20
5.2.3.3	Get	20
5.2.3.4	IsValid	20
5.2.3.5	Next	20
5.2.3.6	operator std::string	20
5.2.3.7	operator!=	21
5.2.3.8	operator*	21
5.2.3.9	operator++	21
5.2.3.10	operator++	21

5.2.3.11	operator+=	21
5.2.3.12	operator->	21
5.2.3.13	operator=	21
5.2.3.14	operator==	21
5.2.3.15	Swap	21
5.3	Gorgon::Filesystem::PathNotFoundError Class Reference	22
5.3.1	Detailed Description	22
5.3.2	Constructor & Destructor Documentation	22
5.3.2.1	PathNotFoundError	22
5.3.2.2	PathNotFoundError	22
Index		23

Chapter 1

Namespace Index

1.1 Namespace List

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

Gorgon::Filesystem	
Contains filesystem functions	7
Gorgon::Threading	
Contains multi-threading functions and objects For thread and mutex see std::thread, std::mutex	14

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Gorgon::Filesystem::EntryPoint	17
std::exception	
std::runtime_error	
Gorgon::Filesystem::PathNotFoundError	22
iterator	
Gorgon::Filesystem::Iterator	18

Chapter 3

Class Index

3.1 Class List

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

Gorgon::Filesystem::EntryPoint	
This class represents filesystem entry points (roots, drives)	17
Gorgon::Filesystem::Iterator	
This iterator allows iteration of directories	18
Gorgon::Filesystem::PathNotFoundError	
This object is thrown from functions that return information rather than status	22

Chapter 4

Namespace Documentation

4.1 Gorgon::Filesystem Namespace Reference

Contains filesystem functions.

Classes

- class [Iterator](#)
This iterator allows iteration of directories.
- class [PathNotFoundError](#)
This object is thrown from functions that return information rather than status.
- class [EntryPoint](#)
This class represents filesystem entry points (roots, drives).

Functions

- void [Initialize](#) ()
Initializes the filesystem module.
- std::string [StartupDirectory](#) ()
Returns the directory where the program is started from.
- std::string [ApplicationDirectory](#) ()
Returns the directory where the program resides.
- bool [Delete](#) (const std::string &path)
Deletes the given file or directory.
- bool [Copy](#) (const std::vector< std::string > &source, const std::string &target)
Copies list of files and/or directories from the given source to destination.
- bool [Save](#) (const std::string &filename, const std::string &data, bool append=false)
Saves a given data into the filename.
- std::string [Load](#) (const std::string &filename)
Loads the given file and returns it in a string form.
- std::string [Relative](#) (std::string path, std::string base=".")
Determine shortest relative path from the given path.
- void [swap](#) ([Iterator](#) &l, [Iterator](#) &r)
Swaps two iterators.
- bool [CreateDirectory](#) (const std::string &path)
Creates a new directory.
- bool [IsDirectory](#) (const std::string &path)

- Checks whether the given path is a directory.*
- bool [IsFile](#) (const std::string &path)
Checks whether the given path is a file.
- bool [IsExists](#) (const std::string &path)
Checks whether the given path exists.
- bool [IsWritable](#) (const std::string &path)
Checks whether the given path is writable.
- bool [IsHidden](#) (const std::string &path)
Checks whether the given path is hidden.
- std::string [Canonize](#) (const std::string &path)
Canonizes a given relative path.
- bool [ChangeDirectory](#) (const std::string &path)
Changes current working directory.
- std::string [CurrentDirectory](#) ()
Returns the current working directory.
- bool [Copy](#) (const std::string &source, const std::string &target)
Copies a file or directory from the given source to destination.
- unsigned long long [Size](#) (const std::string &filename)
Returns the size of the given file.
- bool [Move](#) (const std::string &source, const std::string &target)
Moves a given file or directory.
- std::string [LocateResource](#) (const std::string &path, const std::string &directory="", bool localonly=true)
Locates the given file or directory.
- std::string [GetDirectory](#) (std::string filepath)
Returns the directory portion of a file path.
- std::vector< [EntryPoint](#) > [EntryPoints](#) ()
This function returns all entry points in the current system.

4.1.1 Detailed Description

Contains filesystem functions. All file related functions in Gorgon uses forward slash as directory separator. This includes return values from these functions as well.

Warning

This is a rudimentary filesystem module. Its not meant to be used in serious filesystem tasks. Most functions will not provide any choices related to hard/symbolic links. Additionally, Copy, Delete functions are fully blocking without any reporting facilities.

4.1.2 Function Documentation

4.1.2.1 std::string Gorgon::Filesystem::ApplicationDirectory ()

Returns the directory where the program resides.

Can be used to locate resources. May not be same as StartupDirectory

4.1.2.2 std::string Gorgon::Filesystem::Canonize (const std::string & path)

Canonizes a given relative path.

This method always return with a path that contains at least one slash. However, a slash should be appended to string as it never leaves a slash at the end unless, the path is a root path therefore, it is save to append an extra slash.

Parameters

<i>path</i>	is the file/directory to be canonized. Should contain forward slash as directory separator.
-------------	---

Returns

the full path of the given relative path. Forward slashes are used as directory separator. If the directory is a root directory this method will return a path ending with slash, otherwise it will never return a path ending with slash.

Exceptions

<i>std::runtime_error</i>	if canonize fails. This may or may not be related to path being inexistent or inaccessible.
---------------------------	---

4.1.2.3 bool Gorgon::Filesystem::ChangeDirectory (const std::string & *path*)

Changes current working directory.

Parameters

<i>path</i>	is the directory to become current directory. Should contain forward slash as directory separator.
-------------	--

Returns

false if directory does not exist

4.1.2.4 bool Gorgon::Filesystem::Copy (const std::vector< std::string > & *source*, const std::string & *target*)

Copies list of files and/or directories from the given source to destination.

Hard link sources will be copied instead of links themselves.

Parameters

<i>source</i>	list of source file or directories
<i>target</i>	is directory to copy files or directories into

Returns

true on success

4.1.2.5 bool Gorgon::Filesystem::Copy (const std::string & *source*, const std::string & *target*)

Copies a file or directory from the given source to destination.

Hard link sources will be copied instead of links themselves.

Parameters

<i>source</i>	file or directory. Should either be a single file, or single directory
<i>target</i>	is the new filename or directory name

Returns

true on success

4.1.2.6 bool Gorgon::Filesystem::CreateDirectory (const std::string & *path*)

Creates a new directory.

This function works recursively to create any missing parent directories as well. If directory exists, this function returns true.

Parameters

<i>path</i>	is the path of the directory to be created. Should contain forward slash as directory separator.
-------------	--

Returns

true if the directory is ready to be used, false otherwise

4.1.2.7 std::string Gorgon::Filesystem::CurrentDirectory ()

Returns the current working directory.

Returns

the current working directory Forward slashes are used as directory separator

4.1.2.8 bool Gorgon::Filesystem::Delete (const std::string & *path*)

Deletes the given file or directory.

If the directory is not empty, this function will delete all its contents.

Parameters

<i>path</i>	is the file/directory to be deleted. Should contain forward slash as directory separator.
-------------	---

Returns

true if the given path is deleted. If the given path does not exists, this function will still return true.

4.1.2.9 std::vector< EntryPoint > Gorgon::Filesystem::EntryPoints ()

This function returns all entry points in the current system.

This function does not perform caching and should be used sparingly. It may cause Windows systems to read external devices. On Linux systems, home, root and removable devices are listed.

Returns

The list of entry points

4.1.2.10 std::string Gorgon::Filesystem::GetDirectory (std::string *filepath*)

Returns the directory portion of a file path.

If the file path does not contain any directory related information, this method returns "." to denote current directory. Consider canonizing the returned value. This function expects the input to have / as directory separator.

Parameters

<i>filepath</i>	path that contains the filename
-----------------	---------------------------------

4.1.2.11 void Gorgon::Filesystem::Initialize ()

Initializes the filesystem module.

Gorgon system requires every module to have initialization function even if they are not used. Currently used for following tasks:

- Set startup directory

4.1.2.12 bool Gorgon::Filesystem::IsDirectory (const std::string & path)

Checks whether the given path is a directory.

Parameters

<i>path</i>	is the directory to be checked. Should contain forward slash as directory separator.
-------------	--

Returns

true if the given path is present and a directory

4.1.2.13 bool Gorgon::Filesystem::IsExists (const std::string & path)

Checks whether the given path exists.

Parameters

<i>path</i>	is the file to be checked. Should contain forward slash as directory separator.
-------------	---

Returns

true if the given path is present

4.1.2.14 bool Gorgon::Filesystem::IsFile (const std::string & path)

Checks whether the given path is a file.

Parameters

<i>path</i>	is the file to be checked. Should contain forward slash as directory separator.
-------------	---

Returns

true if the given path is present and a file

4.1.2.15 bool Gorgon::Filesystem::IsHidden (const std::string & path)

Checks whether the given path is hidden.

Parameters

<i>path</i>	is the directory to be checked. Should contain forward slash as directory separator.
-------------	--

Returns

true if the path should be hidden.

4.1.2.16 bool Gorgon::Filesystem::IsWritable (const std::string & *path*)

Checks whether the given path is writable.

Parameters

<i>path</i>	is the directory or file to be checked. Should contain forward slash as directory separator.
-------------	--

Returns

true if the given file/path exists and is writable.

4.1.2.17 std::string Gorgon::Filesystem::Load (const std::string & *filename*)

Loads the given file and returns it in a string form.

Notice that there is no size restriction over the file. This function can handle binary data. Throws [PathNotFoundError](#) if the file cannot be found. Before deciding on file is not found, this function checks if there is a lzma compressed file as filename.lzma

Parameters

<i>filename</i>	is the file to be loaded
-----------------	--------------------------

Returns

the data loaded from the file

Exceptions

PathNotFoundError	if the file cannot be read or does not exists
-----------------------------------	---

4.1.2.18 std::string Gorgon::Filesystem::LocateResource (const std::string & *path*, const std::string & *directory* = " ", bool *localonly* = true)

Locates the given file or directory.

If localonly is true, this function only searches locations that are in the working directory. If it is set to false standard system locations like user home directory or application data directory is also searched. While looking for the resource, if the directory parameter is not empty, the resource is expected to be in the given directory under the local or system wide directory. Additionally, if the file is found as a lzma compressed file, it will be extracted. For instance, if directory parameter is "images", localonly is false, systemname is system and we are looking for icon.png, this function checks whether file exists in the following forms. The first one found is returned, if none exists, [PathNotFoundError](#) exception is thrown. The following list assume user path to be ~ application data path to be ~/apps images/icon.png, ../images/icon.png, icon.png, ../icon.png, images/icon.png.lzma, ../images/icon.png.lzma, icon.png.lzma, ../icon.png.lzma, ~/.system/images/icon.png, ~/.system/images/icon.png.lzma, ~/apps/.system/images/icon.png, ~/apps/.system/images/icon.png.lzma, ~/system/images/icon.png, ~/system/images/icon.png.lzma, ~/apps/system/images/icon.png, ~/apps/system/images/icon.png.lzma, ~/images/icon.png, ~/images/icon.png.lzma, ~/apps/images/icon.png, ~/apps/images/icon.png.lzma, ~/system/icon.png, ~/system/icon.png.lzma, ~/apps/system/icon.png, ~/apps/system/icon.png.lzma If compressed file is found, it will be extracted in place, and the extracted filename will be returned. Therefore, its not necessary to check if the file is compressed or not.

Parameters

<i>path</i>	is the filename or directory to be searched. Only compressed files are handled.
<i>directory</i>	is the directory the resource expected to be in. Should be relative.
<i>localonly</i>	if set, no system or user directories will be searched

Returns

the full path of the resource

Exceptions

<i>PathNotFoundError</i>	if the file cannot be found
<i>std::runtime_error</i>	if the file cannot be read

4.1.2.19 bool Gorgon::Filesystem::Move (const std::string & *source*, const std::string & *target*)

Moves a given file or directory.

Target is the new path rather than the target directory. This function can also be used to rename a file or directory.

Parameters

<i>source</i>	file or directory to be moved or renamed
<i>target</i>	path

Returns

true on success. On Windows this function may not work across different drives. If this case is possible, use Copy then Delete.

4.1.2.20 std::string Gorgon::Filesystem::Relative (std::string *path*, std::string *base* = " . ")

Determine shortest relative path from the given path.

Using "." in second parameter will return the path relative to current directory. This function never returns a path ending with a slash unless its safe to append another slash (i.e. a root path)

Parameters

<i>path</i>	is the path to be relativized.
<i>base</i>	is the base path that will be used to find relative path

Returns

relative path. Note that in Windows it is may be impossible to find relative path.

4.1.2.21 bool Gorgon::Filesystem::Save (const std::string & *filename*, const std::string & *data*, bool *append* = false)

Saves a given data into the filename.

If the file exists, it will be appended if append parameter is set to true. This function can handle binary data.

4.1.2.22 unsigned long long Gorgon::Filesystem::Size (const std::string & *filename*)

Returns the size of the given file.

If the file is not found 0 is returned.

Parameters

<i>filename</i>	is the name of the file
-----------------	-------------------------

Returns

size of the given file

4.1.2.23 `std::string Gorgon::Filesystem::StartupDirectory ()`

Returns the directory where the program is started from.

This will always return the same value through out the execution.

4.1.2.24 `void Gorgon::Filesystem::swap (Iterator & l, Iterator & r)`

Swaps two iterators.

4.2 Gorgon::Threading Namespace Reference

Contains multi-threading functions and objects For thread and mutex see `std::thread`, `std::mutex`.

Functions

- void [RunAsync](#) (`std::function< void()> fn`)
Executes a function asynchronously.
- void [RunInParallel](#) (`std::function< void(int, int)> fn`, unsigned *threads*=0)
Runs a function specified amount of times in parallel.

4.2.1 Detailed Description

Contains multi-threading functions and objects For thread and mutex see `std::thread`, `std::mutex`.

4.2.2 Function Documentation

4.2.2.1 `void Gorgon::Threading::RunAsync (std::function< void()> fn)`

Executes a function asynchronously.

This function starts the thread immediately. There is no way to wait for the thread, stop or query its execution.

Parameters

<i>fn</i>	the function to be executed.
-----------	------------------------------

4.2.2.2 `void Gorgon::Threading::RunInParallel (std::function< void(int, int)> fn, unsigned threads = 0)`

Runs a function specified amount of times in parallel.

threads parameter controls the amount of parallel executions. This function will return when all threads it controls finishes. The following example performs an operation over the data vector using 4 threads. If the *threads* parameter is omitted, the number of threads supported by hardware is used.

```
std::vector<int> data(1000);
RunInParallel([&data](int threadid, int threads) {
    for(int i=threadid;i<data.size();i+=threads) {
        //... do something with data[i]
    }
}, 4);
```

Parameters

<i>fn</i>	is the function to be executed. First parameter of the function should be thread id, second is the number of threads. See the example.
<i>threads</i>	the number of threads to be executed.

Chapter 5

Class Documentation

5.1 Gorgon::Filesystem::EntryPoint Class Reference

This class represents filesystem entry points (roots, drives).

Public Member Functions

- [EntryPoint](#) ()
Default constructor.

Public Attributes

- `std::string` [Path](#)
The path of the entry point.
- `bool` [Readable](#)
Whether the entry point is readable. Currently all entry points are readable.
- `bool` [Writable](#)
Whether the entry point is writable.
- `std::string` [Name](#)
Name or label of the entry point.

5.1.1 Detailed Description

This class represents filesystem entry points (roots, drives).

On Linux like systems, the only entry point is '/', however, user home, root and removable devices are also listed.
On Windows all drives are listed.

See Also

[EntryPoints](#)

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Gorgon::Filesystem::EntryPoint::EntryPoint ()

Default constructor.

5.1.3 Member Data Documentation

5.1.3.1 `std::string Gorgon::Filesystem::EntryPoint::Name`

Name or label of the entry point.

5.1.3.2 `std::string Gorgon::Filesystem::EntryPoint::Path`

The path of the entry point.

5.1.3.3 `bool Gorgon::Filesystem::EntryPoint::Readable`

Whether the entry point is readable. Currently all entry points are readable.

5.1.3.4 `bool Gorgon::Filesystem::EntryPoint::Writable`

Whether the entry point is writable.

Notice that even an entry point is writable it doesn't mean that the immediate path of the entry point is writable. It is possible that the user has no write access to the root of the entry point. If false this denotes the entry point is fully read-only (like a CDROM)

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

- Filesystem.h

5.2 `Gorgon::Filesystem::Iterator` Class Reference

This iterator allows iteration of directories.

Public Member Functions

- `Iterator` (const `std::string` &directory, const `std::string` &pattern="*")
Creates a new iterator from the given directory and pattern.
- `Iterator` (`Iterator` &&dir)
Move constructor.
- `Iterator` (const `Iterator` &other)
Copy constructor.
- `Iterator` ()
Empty constructor. Effectively generates end iterator.
- `Iterator` & `operator=` (`Iterator` other)
Assignment.
- `~Iterator` ()
Destructor.
- void `Swap` (`Iterator` &other)
Swaps iterators, used for move semantics.
- `std::string` `Get` () const
Returns the current path.
- const `std::string` * `operator->` ()
Returns the current path.
- `operator std::string` () const

- Returns the current path.*
- `std::string operator* () const`
Returns the current path.
- `std::string Current ()`
Returns the current path.
- `Iterator & operator++ ()`
Move to the next path in the directory.
- `Iterator & operator+= (int i)`
Moves directory by i elements.
- `Iterator operator++ (int)`
Move to the next path in the directory, return unmodified iterator.
- `bool Next ()`
Next path in the directory.
- `void Destroy ()`
Destroys the current iterator.
- `bool IsValid () const`
Checks whether the iterator is valid.
- `bool operator== (const Iterator &other) const`
Compares two iterators.
- `bool operator!= (const Iterator &other) const`
Compares two iterators.

5.2.1 Detailed Description

This iterator allows iteration of directories.

It is a forward only iterator. An empty iterator can be used for end(). Also instead of comparing the iterator with end, `IsValid()` function could be used.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Gorgon::Filesystem::Iterator::Iterator (const std::string & directory, const std::string & pattern = "*")

Creates a new iterator from the given directory and pattern.

Parameters

<i>directory</i>	is the directory to be iterated. Should exists, otherwise it will throw PathNotFoundError
<i>pattern</i>	wildcard pattern to match paths against

Exceptions

PathNotFoundError	if the given directory does not exists
-----------------------------------	--

5.2.2.2 Gorgon::Filesystem::Iterator::Iterator (Iterator && dir)

Move constructor.

5.2.2.3 Gorgon::Filesystem::Iterator::Iterator (const Iterator & other)

Copy constructor.

5.2.2.4 Gorgon::Filesystem::Iterator::Iterator ()

Empty constructor. Effectively generates end iterator.

5.2.2.5 Gorgon::Filesystem::Iterator::~~Iterator ()

Destructor.

5.2.3 Member Function Documentation

5.2.3.1 std::string Gorgon::Filesystem::Iterator::Current ()

Returns the current path.

Exceptions

<i>std::runtime_error</i>	(debug only) if the iterator is not valid
---------------------------	---

5.2.3.2 void Gorgon::Filesystem::Iterator::Destroy ()

Destroys the current iterator.

5.2.3.3 std::string Gorgon::Filesystem::Iterator::Get () const

Returns the current path.

Exceptions

<i>std::runtime_error</i>	(debug only) if the iterator is not vali
---------------------------	--

5.2.3.4 bool Gorgon::Filesystem::Iterator::IsValid () const

Checks whether the iterator is valid.

5.2.3.5 bool Gorgon::Filesystem::Iterator::Next ()

Next path in the directory.

Returns

true if the iterator is valid

Exceptions

<i>std::runtime_error</i>	(debug only) if the iterator is not valid
---------------------------	---

5.2.3.6 Gorgon::Filesystem::Iterator::operator std::string () const

Returns the current path.

Exceptions

<code>std::runtime_error</code>	(debug only) if the iterator is not valid
---------------------------------	---

5.2.3.7 `bool Gorgon::Filesystem::Iterator::operator!= (const Iterator & other) const`

Compares two iterators.

5.2.3.8 `std::string Gorgon::Filesystem::Iterator::operator* () const`

Returns the current path.

Exceptions

<code>std::runtime_error</code>	(debug only) if the iterator is not valid
---------------------------------	---

5.2.3.9 `Iterator& Gorgon::Filesystem::Iterator::operator++ ()`

Move to the next path in the directory.

5.2.3.10 `Iterator Gorgon::Filesystem::Iterator::operator++ (int)`

Move to the next path in the directory, return unmodified iterator.

5.2.3.11 `Iterator& Gorgon::Filesystem::Iterator::operator+=(int i)`

Moves directory by i elements.

5.2.3.12 `const std::string* Gorgon::Filesystem::Iterator::operator-> ()`

Returns the current path.

Exceptions

<code>std::runtime_error</code>	(debug only) if the iterator is not vali
---------------------------------	--

5.2.3.13 `Iterator& Gorgon::Filesystem::Iterator::operator= (Iterator other)`

Assignment.

5.2.3.14 `bool Gorgon::Filesystem::Iterator::operator== (const Iterator & other) const`

Compares two iterators.

5.2.3.15 `void Gorgon::Filesystem::Iterator::Swap (Iterator & other)`

Swaps iterators, used for move semantics.

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

- Filesystem/Iterator.h
- Filesystem/Linux.cpp

5.3 Gorgon::Filesystem::PathNotFoundError Class Reference

This object is thrown from functions that return information rather than status.

Public Member Functions

- [PathNotFoundError](#) ()
Default constructor.
- [PathNotFoundError](#) (const std::string &what)
Constructor that sets error text.

5.3.1 Detailed Description

This object is thrown from functions that return information rather than status.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Gorgon::Filesystem::PathNotFoundError::PathNotFoundError ()

Default constructor.

5.3.2.2 Gorgon::Filesystem::PathNotFoundError::PathNotFoundError (const std::string & what)

Constructor that sets error text.

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

- Filesystem.h

Index

- ~Iterator
 - Gorgon::Filesystem::Iterator, [20](#)
- ApplicationDirectory
 - Gorgon::Filesystem, [8](#)
- Canonize
 - Gorgon::Filesystem, [8](#)
- ChangeDirectory
 - Gorgon::Filesystem, [9](#)
- Copy
 - Gorgon::Filesystem, [9](#)
- CreateDirectory
 - Gorgon::Filesystem, [9](#)
- Current
 - Gorgon::Filesystem::Iterator, [20](#)
- CurrentDirectory
 - Gorgon::Filesystem, [10](#)
- Delete
 - Gorgon::Filesystem, [10](#)
- Destroy
 - Gorgon::Filesystem::Iterator, [20](#)
- EntryPoint
 - Gorgon::Filesystem::EntryPoint, [17](#)
- EntryPoints
 - Gorgon::Filesystem, [10](#)
- Get
 - Gorgon::Filesystem::Iterator, [20](#)
- GetDirectory
 - Gorgon::Filesystem, [10](#)
- Gorgon::Filesystem, [7](#)
 - ApplicationDirectory, [8](#)
 - Canonize, [8](#)
 - ChangeDirectory, [9](#)
 - Copy, [9](#)
 - CreateDirectory, [9](#)
 - CurrentDirectory, [10](#)
 - Delete, [10](#)
 - EntryPoints, [10](#)
 - GetDirectory, [10](#)
 - Initialize, [11](#)
 - IsDirectory, [11](#)
 - IsExists, [11](#)
 - IsFile, [11](#)
 - IsHidden, [11](#)
 - IsWritable, [12](#)
 - Load, [12](#)
 - LocateResource, [12](#)
 - Move, [13](#)
 - Relative, [13](#)
 - Save, [13](#)
 - Size, [13](#)
 - StartupDirectory, [14](#)
 - swap, [14](#)
- Gorgon::Filesystem::EntryPoint, [17](#)
 - EntryPoint, [17](#)
 - Name, [18](#)
 - Path, [18](#)
 - Readable, [18](#)
 - Writable, [18](#)
- Gorgon::Filesystem::Iterator, [18](#)
 - ~Iterator, [20](#)
 - Current, [20](#)
 - Destroy, [20](#)
 - Get, [20](#)
 - IsValid, [20](#)
 - Iterator, [19](#)
 - Next, [20](#)
 - operator std::string, [20](#)
 - operator*, [21](#)
 - operator++, [21](#)
 - operator+==, [21](#)
 - operator->, [21](#)
 - operator=, [21](#)
 - operator==, [21](#)
 - Swap, [21](#)
- Gorgon::Filesystem::PathNotFoundError, [22](#)
 - PathNotFoundError, [22](#)
- Gorgon::Threading, [14](#)
 - RunAsync, [14](#)
 - RunInParallel, [14](#)
- Initialize
 - Gorgon::Filesystem, [11](#)
- IsDirectory
 - Gorgon::Filesystem, [11](#)
- IsExists
 - Gorgon::Filesystem, [11](#)
- IsFile
 - Gorgon::Filesystem, [11](#)
- IsHidden
 - Gorgon::Filesystem, [11](#)
- IsValid
 - Gorgon::Filesystem::Iterator, [20](#)
- IsWritable
 - Gorgon::Filesystem, [12](#)
- Iterator
 - Gorgon::Filesystem::Iterator, [19](#)

- Load
 - Gorgon::Filesystem, [12](#)
- LocateResource
 - Gorgon::Filesystem, [12](#)
- Move
 - Gorgon::Filesystem, [13](#)
- Name
 - Gorgon::Filesystem::EntryPoint, [18](#)
- Next
 - Gorgon::Filesystem::Iterator, [20](#)
- operator std::string
 - Gorgon::Filesystem::Iterator, [20](#)
- operator*
 - Gorgon::Filesystem::Iterator, [21](#)
- operator++
 - Gorgon::Filesystem::Iterator, [21](#)
- operator+=
 - Gorgon::Filesystem::Iterator, [21](#)
- operator->
 - Gorgon::Filesystem::Iterator, [21](#)
- operator=
 - Gorgon::Filesystem::Iterator, [21](#)
- operator==
 - Gorgon::Filesystem::Iterator, [21](#)
- Path
 - Gorgon::Filesystem::EntryPoint, [18](#)
- PathNotFoundError
 - Gorgon::Filesystem::PathNotFoundError, [22](#)
- Readable
 - Gorgon::Filesystem::EntryPoint, [18](#)
- Relative
 - Gorgon::Filesystem, [13](#)
- RunAsync
 - Gorgon::Threading, [14](#)
- RunInParallel
 - Gorgon::Threading, [14](#)
- Save
 - Gorgon::Filesystem, [13](#)
- Size
 - Gorgon::Filesystem, [13](#)
- StartupDirectory
 - Gorgon::Filesystem, [14](#)
- Swap
 - Gorgon::Filesystem::Iterator, [21](#)
- swap
 - Gorgon::Filesystem, [14](#)
- Writable
 - Gorgon::Filesystem::EntryPoint, [18](#)