SULF Reference Manual

0.3

Generated by Doxygen 1.4.2

Fri Apr 29 22:42:10 2005

Contents

Chapter 1

SULF - Stackable User-Level Filesystem.

1.1 Introduction

SULF allows you to write a Linux filesystem in C#. It uses the FUSE library to do the actual Linux filesystem integration in user-space.

By using FUSE, the C# code can run entirely in user-space and not require any special permissions to serve the filesystem.

SULF is meant to be useful to filesystem *developers*. It currently contains only a simple test filesystem to use for testing.

1.2 Recent Changes

2005.04.29 – 0.3, Queen's Day release

- major API changes. API has become simpler and is split into two libraries. The Fuse.dll library communicates with the FUSE kernel module and provides an API based on simple C# interfaces (Node, FileNode, DirNode). The Sulf.dll library builds on Fuse.dll and provides a more natural interface to C# objects (a Hashmap can be used as a directory, string as a file, etc).

- Updated FuseWrapper and Fuse interfaces for FUSE 2.3 (Kernel api version 6.1)

2005.01.15

- updated FuseWrapper and Fuse interfaces for FUSE 2.1 compatibility.
- implemented simple Hello-World filesystem in examples subdirectory, which is similar to libfuse's example/hello.c

2005.01.13

- release under LGPL license
- simplify build system to make it easier to build
- add snapshot (see Downloads)

1.3 Downloads

SULF is kept in revision control using Darcs (`http://abridgegame.org/darcs/`). The current development branch is available here: `http://arg0.net/users/vgough/devel/sulf`

An easier way to get started is to use the snapshot, which contains pre-built objects: `sulf-0.3.tar.gz`

See also Dependencies , and Getting Started

1.4 Previous Attempts

The old way

My first attempt at building a C# wrapper for fuse was based on capturing all the callbacks from libfuse in C code, and then have a conversion layer translate that into calls to C# code (using internal Mono C# API). This worked, but had some downsides:

- required internal Mono API, so it couldn't be compiled or run with other C# compilers or runtimes.
- could only run in single-threaded mode, since libfuse's threads created from C code would not play well with Mono's thread state.

The new way

The current method is significantly different. The C# code takes control immediately and talks directly to the FUSE kernel module with the help of a some C code (and wrappers generated automatically by SWIG).

This no longer requires the use of Mono internal API's, and it also allows the filesystem to run multi-threaded. The downside is that the FUSE kernel interface has been more variable then the libfuse interface, which means that the C# interface will work with fewer versions of the FUSE code tree.

1.5 Interface Design - Fuse.dll

Implementing a filesystem using Fuse.dll requires implementing three sets of interfaces. These interfaces correspond to the type of function calls that they will handle: filesystem level, directory level, and file level.

1. Fuse.FileSystem interface, which defines a Stat operation and a means of getting the top level directory.

2. Fuse.DirNode interface, which is used for access to directories (one of which is returned by the filesystem level – FileSystem.RootNode):

3. Fuse.FileNode interface is used for access to normal files

In addition to this three way split, the DirNode and FileNode interfaces also come in two flavors – one for read-only objects, and another with write support. The Fuse.dll dispatch handler knows how to treat a file based on the interfaces it implements. For example, if a DirNode returns an object which implements FileNode (but not MutableFileNode), then the file will be read-only.

1.6 Interface Design - Sulf.dll

The higher-level interface - Sulf.dll is meant to be a better / easier starting point for most filesystems, unless you need the full power of the underlying Fuse.dll interface.

The Sulf.dll design is meant to translate from the Fuse.dll api into something more transparent to C# objects. This layer is newer and is in development.

TODO: more to come. See examples and Sulf directories.

1.7 Dependencies

Sulf relies on these projects:

FUSE 2.3 (Filesystem in User SpacE): `http://sourceforge.net/projects/fuse`

Mono (C# compiler and runtime): `http://go-mono.org`

SWIG (Simplified Wrapper and Interface Generator): http://www.swig.org/ **Must use SWIG >= 1.3.24**. SWIG's C# output has been maturing recently, and SULF will not build with earlier versions of SWIG..

1.8 Getting Started

As long as you have the necessary dependencies statisfied, you can build the SULF libraries and a simple test filesystem by doing the following:

```
autoreconf      # build configure script

./configure
make

cd examples

make run        # run hello-fs example which mounts to /tmp/hellofs
```

Chapter 2

SULF Directory Hierarchy

2.1 SULF Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 3

SULF Namespace Index

3.1 SULF Namespace List

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

Chapter 4

SULF Hierarchical Index

4.1 SULF Class Hierarchy

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

Chapter 5

SULF Class Index

5.1 SULF Class List

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

Chapter 6

SULF Page Index

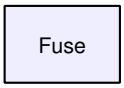
6.1 SULF Related Pages

Here is a list of all related documentation pages:

Chapter 7

SULF Directory Documentation

7.1 Fuse/ Directory Reference



Files

- file **Fuse/AssemblyInfo.cs**
- file **Channel.cs**
- file **DirNode.cs**
- file **ErrorCode.cs**
- file **FileNode.cs**
- file **FileSystem.cs**
- file **LookupTransaction.cs**
- file **Node.cs**
- file **NodeMap.cs**
- file **OpenTransaction.cs**
- file **Operation.cs**
- file **Reactor.cs**
- file **ShowFuseOptions.cs**
- file **StaleNodeException.cs**
- file **Stat.cs**
- file **StatFS.cs**
- file **StringArg.cs**

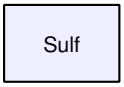
- file **Transactional.cs**
- file **XAttrNode.cs**

7.2 FuseWrapper/ Directory Reference

Files

- file **FuseWrapper/AssemblyInfo.cs**
- file **BufferInterfaces.cs**
- file **fuse.swig**
- file **FWBuffer.h**

7.3 Sulf/ Directory Reference



Files

- file **Sulf/AssemblyInfo.cs**
- file **CallbackFileNode.cs**
- file **GenericFileNode.cs**
- file **GenericNode.cs**
- file **MapFS.cs**
- file **Mounter.cs**
- file **StringNode.cs**

Chapter 8

SULF Namespace Documentation

8.1 Fuse Namespace Reference

8.1.1 Detailed Description

Provides a low-level FUSE kernel <-> C# API translation layer.

When implementing a filesystem based on the Fuse layer, you will need to create classes which implement the Fuse.FileSystem, Fuse.DirNode, and Fuse.FileNode interfaces.

Fuse.Channel is used to communicate with the FUSE kernel module. Fuse.Reactor handles incoming messages and forwards them to the appropriate node.

See also:
Interface Design - Fuse.dll

Author:
Valient Gough

Classes

- class Channel

  Communication channel to FUSE kernel module.

- interface DirNode

  Interface for operations on directories.

- interface FileNode

  Interface for operation on file types.

- class FileSystem

  Simple interface for a generic filesystem.

- class LookupTransaction

  Base transaction implementation for types returning fuse_entry_out.

- interface Node

  A generic node of a filesystem.

- class NodeMap

  Maintains mapping from integer node id to Fuse.Node.

- class OpenTransaction

  Transaction for Open command.

- struct Operation

  Emitted for every request from Fuse.Channel.

- class Reactor

  Decodes and dispatches requests comming from the FUSE kernel.

- class StaleNodeException

  Exception used internally by Fuse.dll.

- struct StatFS

  Structure for returning information about a filesystem.

- class StringArg
- interface Transactional

  Simple transaction definition.

- interface XAttrNode

  Interface for nodes supporting eXtended Attribute calls.

Enumerations

- enum ErrorCode {

  Success = 0, **EPERM** = -1, **ENOENT** = -2, **EINTR** = -4,

  EIO = -5, **EAGAIN** = -11, **ENOMEM** = -12, **EXDEV** = -18,

  ENODEV = -19, **ENOTDIR** = -20, **EISDIR** = -21, **EINVAL** = -22,

  ENOSYS = -38, **EPROTO** = -71, **ENOTSUP** = -95, **ESTALE** = -116 }

  Common error codes (negated).

- enum ModeFlags {

  Socket = 49152, **SymLink** = 40960, **RegularFile** = 32768, **BlockDevice** = 24576,

  Directory = 16384, **CharDevice** = 8192, **FIFO** = 4096, **ReadWriteAll** = 438,

  ReadAll = 292 }

Functions

- delegate int **ReadDirCallback** (string name, ulong inode, uint type, ulong next-Offset)

8.1.2 Enumeration Type Documentation

8.1.2.1 enum Fuse::ErrorCode

Common error codes (negated).

These are common error codes (negated for convenience) which may be returned by some filesystem operations. Since FUSE is meant to work with Linux, the error codes are made to match the Linux error codes.

Note that for most FUSE calls, an error code of 0 indicates success, and a negative value indicates the error code. This is different from most Unix system calls which may return -1 on error and set *errno* to indicate a (positive valued) error code.

8.1.2.2 enum Fuse::ModeFlags

Mode flags for some common types. Used for testing / checking Stat.mode value.

8.2 Sulf Namespace Reference

8.2.1 Detailed Description

Provides a high level C# interface on top of Fuse.

For example, this code:

```
Sulf.MapFS root = new Sulf.MapFS();
root["hello"] = "Hello world!\n"

Channel channel = Sulf.Mounter.Mount("HelloFS", root, "/tmp/hellofs");
channel.EventLoop();
```

Results in this filesystem:

```
> ls -l /tmp/hellofs
-r--r--r--  1 root root 13 2005-04-29 17:04 hello
> cat /tmp/hellofs/hello
Hello world!
```

The Mounter class is a simple helper to mount filesystems.

The MapFS class provides a Fuse.DirNode implementation which make a map (IDictionary) appear as a directory.

Note:
The Sulf library is under development.

See also:
Interface Design - Sulf.dll

Author:
Valient Gough

Classes

- class GenericFileNode

  Provide a default implementation of methods.

- class GenericNode

  Provides default implementation for Fuse.Node.

- class MapFS

  An in-memory filesystem node.

- class Mounter

  Provides helper methods for mounting a filesystem.

- class StringNode

  Provides a read-only Fuse.FileNode for a string.

Chapter 9

SULF Class Documentation

9.1 BufferReadable Interface Reference

`#include <FWBuffer.h>`

9.1.1 Detailed Description

Simple interface for objects that can be read from a FWBuffer.

Public Member Functions

- int **copyFrom** (FWBuffer buf)
- int **copyFrom** (FWBuffer buf)

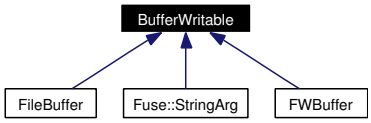
The documentation for this interface was generated from the following files:

- BufferInterfaces.cs
- FWBuffer.h

9.2 BufferWritable Interface Reference

`#include <FWBuffer.h>`

Inheritance diagram for BufferWritable:



9.2.1 Detailed Description

Simple interface for objects that can be written to a FWBuffer.

Public Member Functions

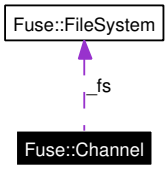
- int **copyTo** (FWBuffer buf)
- int **copyTo** (FWBuffer buf)

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

- BufferInterfaces.cs
- FWBuffer.h

9.3 Fuse::Channel Class Reference

Collaboration diagram for Fuse::Channel:



9.3.1 Detailed Description

Communication channel to FUSE kernel module.

Fuse.Channel communicates with the FUSE kernel module. Each Channel instance is tied to a filesystem mount point.

When a channel receives a request from the kernel, it decodes the request header and emits the Channel.OnNewCommand event. The event contains one argument - an Operation. The Operation specifies which filesystem is involved, the command header, and the buffer of further data related to the request.

OnNewCommand is emitted using an asynchronous callback, so subscribers should be thread safe. TODO: this might be configureable in the future..

A Channel controls a single mount-point. Multiple Channel instances may be created and served by either a single Reactor, or multiple Reactor instances. A separate thread will be required for each Channel instance's event loop, since there is currently no support for multiplexing Channel event loops.

Author:
Valient Gough

Public Member Functions

- delegate object ProcessCommand (Operation op)
- Channel (string mountPoint, FileSystem fs)
- bool EventLoop ()
- void **Mount** (params string[ ] options)
- void **Unmount** ()

Static Public Member Functions

- static void Usage ()

Public Attributes

- event ProcessCommand OnNewCommand

Properties

- string **MountPoint**

9.3.2 Constructor & Destructor Documentation

9.3.2.1 Fuse::Channel::Channel (string *mountPoint*, FileSystem *fs*) [inline]

Instanciate a channel for the given filesystem mount point.

This tells FUSE that we will implement the filesystem at the specified mount point.

Parameters:

mountPoint A fully qualified directory name which will be the root of the filesystem.

9.3.3 Member Function Documentation

9.3.3.1 bool Fuse::Channel::EventLoop () [inline]

Waits for events, and does not return until the filesystem is unmounted.

Normally this should not be used directly - see Reactor which provides a higher level interface.

Returns *true* if exited normally.

9.3.3.2 delegate object Fuse::Channel::ProcessCommand (Operation *op*)

Command processing delegate.

Process an operation and return a result object. What's done with the result will depend on its type. The following are handled:

(null) : no result sent for operation (int) : an error code to return to the caller (BufferWritable) : error code 0, encode buffer as reply (Transactional) : play transaction, send result anything else: return generic error code to caller

9.3.3.3 static void Fuse::Channel::Usage () [inline, static]

abuse fuse_main to have it show usage information. Unfortunatly, libfuse will then directly call exit(), so we can't continue after this..

9.3.4 Member Data Documentation

9.3.4.1 event ProcessCommand Fuse::Channel::OnNewCommand

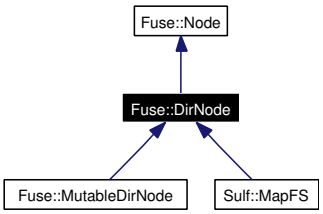
event handler for new commands.. Every time a command is received from the kernel, it is emitted using this event.

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

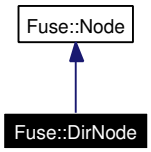
- Channel.cs

9.4 Fuse::DirNode Interface Reference

Inheritance diagram for Fuse::DirNode:



Collaboration diagram for Fuse::DirNode:



9.4.1 Detailed Description

Interface for operations on directories.

Public Member Functions

- long **Open** (uint flags)
- int **Release** (ulong fh, uint flags)
- int **ReadDir** (ulong fh, ulong offset, ReadDirCallback filler)
- Node **Lookup** (string name, ref int errCode)

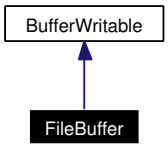
The documentation for this interface was generated from the following file:

- DirNode.cs

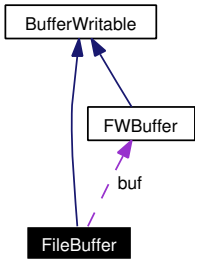
9.5 FileBuffer Class Reference

`#include <FWBuffer.h>`

Inheritance diagram for FileBuffer:



Collaboration diagram for FileBuffer:



9.5.1 Detailed Description

Helper class for storing readdir information.

The FileBuffer.addFile method is designed to be compatible with the ReadDirCallback delegate, so it can be used directly for filling information from a readdir call.

Public Member Functions

- int addFile (const char *name, __u64 nodeId, __u32 type, __u64 off)
- int **copyTo** (struct FWBuffer *outBuf)

Public Attributes

- FWBuffer * **buf**
- int **error**

9.5.2 Member Function Documentation

9.5.2.1 int FileBuffer::addFile (const char * *name*, __u64 *nodeId*, __u32 *type*, __u64 *off*)

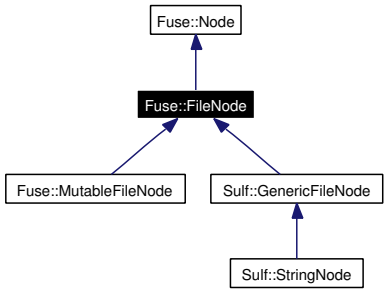
add the filename with given type, node id and [next] offset.

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

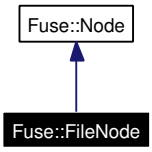
- FWBuffer.h

9.6 Fuse::FileNode Interface Reference

Inheritance diagram for Fuse::FileNode:



Collaboration diagram for Fuse::FileNode:



9.6.1 Detailed Description

Interface for operation on file types.

The FileNode interface defines operations which are possible on various types of files. All operations may not be valid for every file type.

Public Member Functions

- string ReadLink ()

  If the file is a symbolic link, return the link target.

- long Open (uint flags)
- int Release (ulong fh, uint flags)
- int Flush (ulong fh)
- int **Read** (ulong fh, FWBuffer data, uint size, ulong offset)

9.6.2 Member Function Documentation

9.6.2.1 int Fuse::FileNode::Flush (ulong *fh*)

Flush is called on a close().. However this does not mean that the filehandle should be closed, because it might have been cloned. Instead Release() will be called when the last reference is closed.

Implemented in Sulf::GenericFileNode.

9.6.2.2 long Fuse::FileNode::Open (uint *flags*)

Open the file with the given flags and return a file descriptor.

Returns:

File descriptor on success (> 0), or a negative error code on failure (See Fuse.ErrorCode)

Implemented in Sulf::GenericFileNode.

9.6.2.3 int Fuse::FileNode::Release (ulong *fh*, uint *flags*)

Release the previously opened file descriptor.

Returns:

Zero on success, or a negative error code on failure (See Fuse.ErrorCode)

Implemented in Sulf::GenericFileNode.

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

- FileNode.cs

9.7 Fuse::FileSystem Class Reference

9.7.1 Detailed Description

Simple interface for a generic filesystem.

This provides a very basic interface for a filesystem implementation. Basically a filesystem contains just a way to get StatFS information, a way to get the root node (which should be a DirNode), and a way to get the mountpoint.

Author:
Valient Gough

Public Member Functions

- virtual int Statfs (out StatFS stat)

Properties

- abstract Fuse.DirNode RootNode
- abstract string Name
- virtual string[ ] RequiredMountArgs

9.7.2 Member Function Documentation

9.7.2.1 virtual int Fuse::FileSystem::Statfs (out StatFS *stat*) [inline, virtual]

Get filesystem statistics

See also:
man:statfs(2)

9.7.3 Property Documentation

9.7.3.1 abstract string Fuse::FileSystem::Name [get]

The filesystem's name. This can be an implementation name, such as "RawFS".

9.7.3.2 virtual string [ ] Fuse::FileSystem::RequiredMountArgs [get]

If special mount flags are required for operation, they should be provided by this property.

9.7.3.3 abstract Fuse.DirNode Fuse::FileSystem::RootNode [get]

Return node corresponding to the root directory of the filesystem.

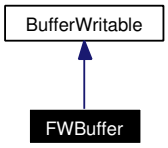
The documentation for this class was generated from the following file:

- FileSystem.cs

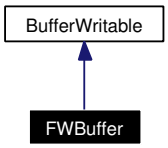
9.8 FWBuffer Class Reference

`#include <FWBuffer.h>`

Inheritance diagram for FWBuffer:



Collaboration diagram for FWBuffer:



9.8.1 Detailed Description

A C-managed buffer with convenience methods.

FWBuffer is a SWIG produced wrapper for a buffer in C-space (unmanaged code in .net terms). Convenience functions are provided for reading / writing to the buffer, and adding and pulling out data.

All read and write functions operation based on the current offset. The offset can be reset to 0 using Reset().

In addition, all read and write functions will increment the offset. Read functions will also increment the initialized counter.

Public Member Functions

- FWBuffer (int size)

  Allocate a buffer with the specified initialize size.

- void Rewind ()

  Rewind buffer. Sets current offset to 0.

- void Reset ()
- int EnsureFreeSpace (int freeBytes)
- int Read (int fd)
- int Read (int fd, size_t count, __u64 offset)
- int Write (int fd, size_t count, __u64 offset)
- int Write (int fd)
- const char * GetString ()
- int GetBytes (char *data, int len) int GetByte() int AddBytes(const char *data
- int AddString (const char *str)
- int AddSubString (const char *str, int offset, int len)
- int AddBuffer (struct FWBuffer *arg)
- int copyTo (struct FWBuffer *outBuf)

Public Attributes

- int **initialized**
- int length
- int offset
- char * data
- int int **len**

9.8.2 Member Function Documentation

9.8.2.1 int FWBuffer::AddBuffer (struct FWBuffer * *arg*)

Add another buffer contents to the buffer at the current offset.

9.8.2.2 int FWBuffer::AddString (const char * *str*)

Add a string to the buffer (including the terminating null) at the current offset.

9.8.2.3 int FWBuffer::AddSubString (const char * *str*, int *offset*, int *len*)

Add a portion of a string.

9.8.2.4 int FWBuffer::copyTo (struct FWBuffer * *outBuf*)

Part of BufferWritable protocol - copies contents to another buffer. bufA.copyTo(bufB) is the same as bufB.AddBuffer(bufA)

9.8.2.5 int FWBuffer::EnsureFreeSpace (int *freeBytes*)

Make sure there are at least *freeBytes* of space starting at the current offset. Reallocates the buffer if necessary.

9.8.2.6 int FWBuffer::GetBytes (char * *data*, int *len*) const

Add bytes to the buffer at the current offset.

9.8.2.7 const char* FWBuffer::GetString ()

Return a string from the buffer. The returned string is allocated with strdup, and becomes the responsibility of the caller.

9.8.2.8 int FWBuffer::Read (int *fd*, size_t *count*, __u64 *offset*)

Read into the buffer at the current offset. Space will be guarenteed for *count* bytes. The offset specifies where to read the data from, not where the data will go in the buffer.

9.8.2.9 int FWBuffer::Read (int *fd*)

Read into the buffer at the current offset. The max read size depends on the remaining space in the buffer.

9.8.2.10 void FWBuffer::Reset ()

Reset to uninitialized. Sets current offset to 0, and resets initialized to 0.

9.8.2.11 int FWBuffer::Write (int *fd*)

Write the entire buffer (starting at the current offset) into the file, as its current offset.

9.8.2.12 int FWBuffer::Write (int *fd*, size_t *count*, __u64 *offset*)

Write data from the buffer at the current offset. Count bytes will be written to the specified offset of the file.

9.8.3 Member Data Documentation

9.8.3.1 char* FWBuffer::data

current copy offset

9.8.3.2 int FWBuffer::length

number of bytes initialized

9.8.3.3 int FWBuffer::offset

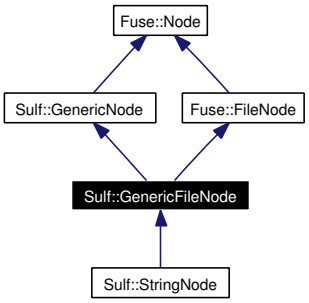
real length of array

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

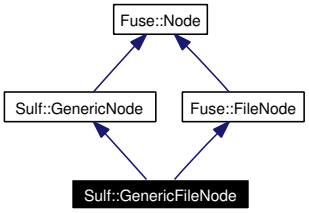
- FWBuffer.h

9.9 Sulf::GenericFileNode Class Reference

Inheritance diagram for Sulf::GenericFileNode:



Collaboration diagram for Sulf::GenericFileNode:



9.9.1 Detailed Description

Provide a default implementation of methods.

Public Member Functions

- **GenericFileNode** (string name)
- virtual string ReadLink ()

  If the file is a symbolic link, return the link target.

- virtual long Open (uint flags)
- virtual int Release (ulong fh, uint flags)
- virtual int Flush (ulong fh)
- abstract int **Read** (ulong fh, FWBuffer data, uint size, ulong offset)

9.9.2 Member Function Documentation

9.9.2.1 virtual int Sulf::GenericFileNode::Flush (ulong *fh*) [inline, virtual]

Flush is called on a close().. However this does not mean that the filehandle should be closed, because it might have been cloned. Instead Release() will be called when the last reference is closed.

Implements Fuse::FileNode.

9.9.2.2 virtual long Sulf::GenericFileNode::Open (uint *flags*) [inline, virtual]

Open the file with the given flags and return a file descriptor.

Returns:

File descriptor on success (> 0), or a negative error code on failure (See Fuse.ErrorCode)

Implements Fuse::FileNode.

9.9.2.3 virtual int Sulf::GenericFileNode::Release (ulong *fh*, uint *flags*) [inline, virtual]

Release the previously opened file descriptor.

Returns:

Zero on success, or a negative error code on failure (See Fuse.ErrorCode)

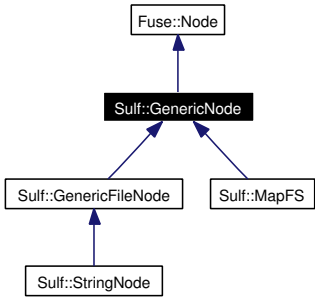
Implements Fuse::FileNode.

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

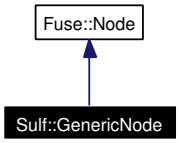
- GenericFileNode.cs

9.10 Sulf::GenericNode Class Reference

Inheritance diagram for Sulf::GenericNode:



Collaboration diagram for Sulf::GenericNode:



9.10.1 Detailed Description

Provides default implementation for Fuse.Node.

Used as base class for Sulf nodes.

Public Member Functions

- **GenericNode** (string name)
- virtual int GetStat (out Stat stat)

Properties

- uint **StatMode**
- uint **StatLink**
- virtual string Name

  return local name (eg a file "/tmp/foo/bar" would return "bar");

- virtual ulong **NodeId**
- virtual ulong **NodeGeneration**

9.10.2 Member Function Documentation

9.10.2.1 virtual int Sulf::GenericNode::GetStat (out Stat *stat*) [inline, virtual]

Get statistics for this node.

See also:
man:stat(2)

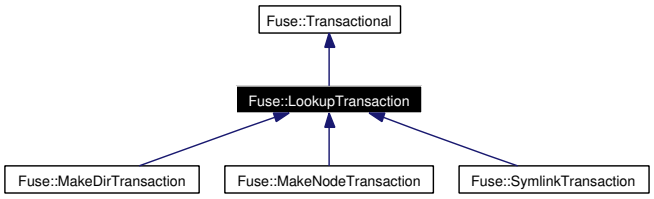
Implements Fuse::Node.

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

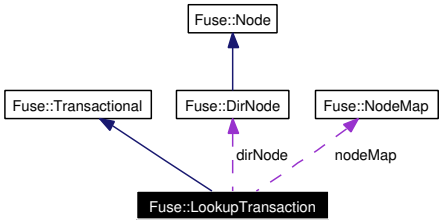
- GenericNode.cs

9.11 Fuse::LookupTransaction Class Reference

Inheritance diagram for Fuse::LookupTransaction:



Collaboration diagram for Fuse::LookupTransaction:



9.11.1 Detailed Description

Base transaction implementation for types returning fuse_entry_out.

If the lookup is aborted, then NodeMap.DropNode() is called to drop the node.

Public Member Functions

- **LookupTransaction** (NodeMap nodeMap, DirNode dirNode, string name)
- virtual object beginTransaction ()

  start the transaction

- virtual bool commitTransaction (object transaction)
- virtual void abortTransaction (object transaction)

Protected Attributes

- int **res** = 0

9.11.2 Member Function Documentation

9.11.2.1 virtual void Fuse::LookupTransaction::abortTransaction (object *transaction*) [inline, virtual]

abort the transaction – argument is the return value from beginTransaction

Implements Fuse::Transactional.

9.11.2.2 virtual bool Fuse::LookupTransaction::commitTransaction (object *transaction*) [inline, virtual]

complete the operation – argument is the return value from beginTransaction

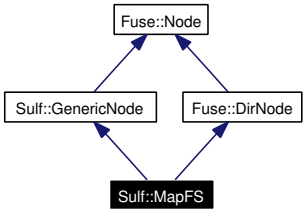
Implements Fuse::Transactional.

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

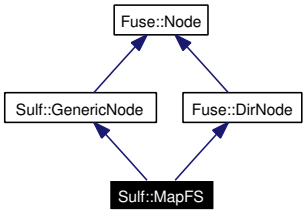
- LookupTransaction.cs

9.12 Sulf::MapFS Class Reference

Inheritance diagram for Sulf::MapFS:



Collaboration diagram for Sulf::MapFS:



9.12.1 Detailed Description

An in-memory filesystem node.

The way MapFS handles child nodes depends on their type:

- implements Fuse.Node: Passed through directly to Fuse.dll layer.
- implements IDictionary: Treated as a subdirectory and a MapFS object is created to serve its contents.
- all others: A string type is served by StringFileNode

For any other types, ToString() is used to convert the object to a string, and StringFile-Node is used.

See example/HelloFS.cs

See also:
StringFileNode

Public Member Functions

- **MapFS** (string name, IDictionary map)
- **MapFS** (string name)
- virtual long **Open** (uint flags)
- virtual int **Release** (ulong fh, uint flags)
- virtual int **ReadDir** (ulong fh, ulong offset, ReadDirCallback cb)
- virtual Fuse.Node **Lookup** (string name, ref int errCode)

Properties

- object this [string name]

9.12.2 Property Documentation

9.12.2.1 object Sulf::MapFS::this[string name] [get, set]

Allow get or set of filesystem objects.

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

- MapFS.cs

9.13 Sulf::Mounter Class Reference

9.13.1 Detailed Description

Provides helper methods for mounting a filesystem.

See also:
sulf-design

Static Public Member Functions

- static Fuse.Channel Mount (string fsName, Fuse.DirNode rootDir, string mountPoint)
- static Fuse.Channel Mount (Fuse.FileSystem fs, string mountPoint)

9.13.2 Member Function Documentation

9.13.2.1 static Fuse.Channel Sulf::Mounter::Mount (Fuse.FileSystem *fs*, string *mountPoint*) [inline, static]

Mount a filesystem at the given mountPoint. Simply a wrapper around standard calls to Fuse.Channel and Fuse.Reactor.

Returns Fuse.Channel, which must have eventLoop() called in order to process filesystem commands.

9.13.2.2 static Fuse.Channel Sulf::Mounter::Mount (string *fsName*, Fuse.DirNode *rootDir*, string *mountPoint*) [inline, static]

Mount a filesystem at the given mountPoint. Uses the provided Fuse.DirNode as the root of the filesystem.

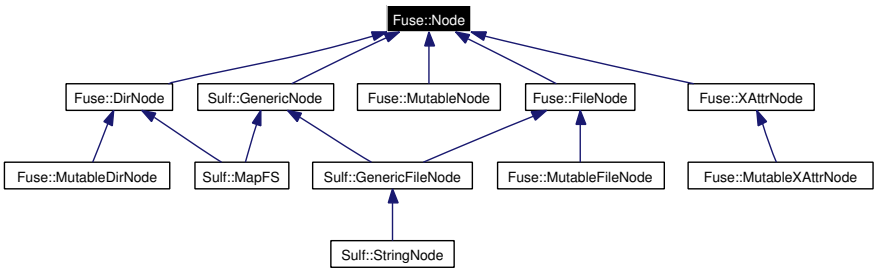
Returns Fuse.Channel, which must have eventLoop() called in order to process filesystem commands.

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

- Mounter.cs

9.14 Fuse::Node Interface Reference

Inheritance diagram for Fuse::Node:



9.14.1 Detailed Description

A generic node of a filesystem.

A node can be any type of file (even non-existing files are represented by a Node instance).

Author:
Valient Gough

Public Member Functions

- int GetStat (out Stat stat)

Properties

- string Name

  return local name (eg a file "/tmp/foo/bar" would return "bar");

- ulong **NodeId**
- ulong **NodeGeneration**

9.14.2 Member Function Documentation

9.14.2.1 int Fuse::Node::GetStat (out Stat *stat*)

Get statistics for this node.

See also:
man:stat(2)

Implemented in Sulf::GenericNode.

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

- Node.cs

9.15 Fuse::NodeMap Class Reference

9.15.1 Detailed Description

Maintains mapping from integer node id to Fuse.Node.

Public Member Functions

- void DropNode (ulong nodeId)
- void **TrackNode** (Node node)

Properties

- Node **this** [ulong nodeId]

9.15.2 Member Function Documentation

9.15.2.1 void Fuse::NodeMap::DropNode (ulong *nodeId*) [inline]

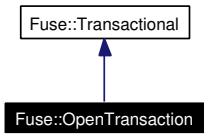
Drop the specified node from the map.

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

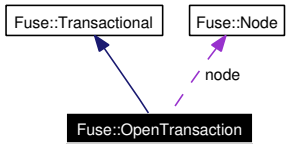
- NodeMap.cs

9.16 Fuse::OpenTransaction Class Reference

Inheritance diagram for Fuse::OpenTransaction:



Collaboration diagram for Fuse::OpenTransaction:



9.16.1 Detailed Description

Transaction for Open command.

Public Member Functions

- **OpenTransaction** (FileNode fnode, uint flags)
- **OpenTransaction** (DirNode dnode, uint flags)
- object beginTransaction ()

  start the transaction

- bool commitTransaction (object data)
- void abortTransaction (object data)

9.16.2 Member Function Documentation

9.16.2.1 void Fuse::OpenTransaction::abortTransaction (object *data*) [inline]

If the command was aborted, then issue a Release to counteract the effect of an Open..

Implements Fuse::Transactional.

9.16.2.2 bool Fuse::OpenTransaction::commitTransaction (object *data*) [inline]

complete the operation – argument is the return value from beginTransaction

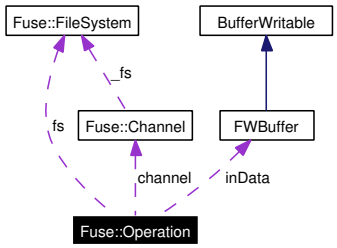
Implements Fuse::Transactional.

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

- OpenTransaction.cs

9.17 Fuse::Operation Struct Reference

Collaboration diagram for Fuse::Operation:



9.17.1 Detailed Description

Emitted for every request from Fuse.Channel.

Public Attributes

- Channel **channel**
- FileSystem fs

  the filesystem data as specified during the Channel constructor

- fuse_in_header header

  header which accompanies every operation

- FWBuffer inData

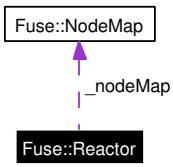
  arguments (which arguments are encoded depends on the operation)

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

- Operation.cs

9.18 Fuse::Reactor Class Reference

Collaboration diagram for Fuse::Reactor:



9.18.1 Detailed Description

Decodes and dispatches requests comming from the FUSE kernel.

Reactor subscribes to events from a Fuse.Channel, decodes them, and converts them into requests on particular file or directory nodes (FileNode or DirNode).

Typical usage for mounting a filesystem:

```
// select a mount point where the filesystem will be visible
string mountPoint = "/tmp/foo";

// Create the filesystem which will handle the requests.
Fuse.FileSystem fs = new RawFS.RawFileSystem( "/tmp/foo-raw" );

// create the channel to the FUSE kernel module.
Channel channel = new Fuse.Channel( mountPoint, fs );

// create a reactor to dispatch the requests
Fuse.Reactor reactor = new Fuse.Reactor();

// subscribe to one or more channels..
reactor.SubscribeTo( channel );

// have channel process events until it is unmounted..
channel.EventLoop();
```

See also:
Channel

Author:
Valient Gough

Public Member Functions

- Reactor SubscribeTo (Channel channel)

Static Public Member Functions

- static void **ConvertStat** (Fuse.Stat stat, out fuse_attr attr)
- static void **ConvertStat** (fuse_attr attr, out Fuse.Stat stat)
- static void **ConvertStatFS** (Fuse.StatFS stat, out fuse_kstatfs kstat)

Public Attributes

- const int **ENTRY_REVALIDATE_TIME** = 1
- const int **ATTR_REVALIDATE_TIME** = 1

9.18.2 Member Function Documentation

9.18.2.1 Reactor Fuse::Reactor::SubscribeTo (Channel *channel*) [inline]

Subscribe the Reactor to events from the given Channel.

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

- Reactor.cs

9.19 Fuse::StaleNodeException Class Reference

9.19.1 Detailed Description

Exception used internally by Fuse.dll.

If the system requests a node by id, but that id is no longer in use, then a StaleNodeException exception is thrown. This results in an ESTALE error being returned to the kernel for the operation.

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

- StaleNodeException.cs

9.20 Fuse::StatFS Struct Reference

9.20.1 Detailed Description

Structure for returning information about a filesystem.

Warning:
This must match the definition in librawfs/rawfs.h, as it is used in P/Invoke calls.

Public Attributes

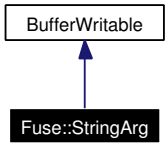
- int **blockSize**
- long **blocks**
- long **blocksFree**
- long **blocksAvail**
- long **files**
- long **filesFree**
- int **namelen**

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

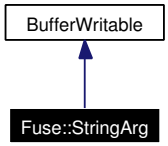
- StatFS.cs

9.21 Fuse::StringArg Class Reference

Inheritance diagram for Fuse::StringArg:



Collaboration diagram for Fuse::StringArg:



9.21.1 Detailed Description

Wrapper around a simple string to provide a BufferWritable interface.

Public Member Functions

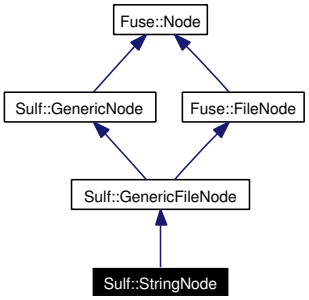
- **StringArg** (string arg)
- int **copyTo** (FWBuffer buf)

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

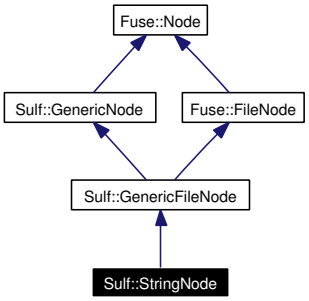
- StringArg.cs

9.22 Sulf::StringNode Class Reference

Inheritance diagram for Sulf::StringNode:



Collaboration diagram for Sulf::StringNode:



9.22.1 Detailed Description

Provides a read-only Fuse.FileNode for a string.

Public Member Functions

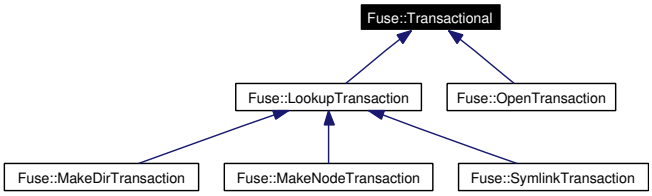
- **StringNode** (string name, object data)
- override int **Read** (ulong fh, FWBuffer data, uint size, ulong offset)
- override int **GetStat** (out Fuse.Stat stat)

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

- StringNode.cs

9.23 Fuse::Transactional Interface Reference

Inheritance diagram for Fuse::Transactional:



9.23.1 Detailed Description

Simple transaction definition.

Fuse.Channel supports transactions. This are used for operations which may be canceled by the kernel. beginTransaction() is called before returning the result to the kernel. If the kernel accepts, then commitTransaction() is called, otherwise abortTransaction() is called.

See also:
Fuse.Channel

Public Member Functions

- object beginTransaction ()

  start the transaction

- bool commitTransaction (object data)
- void abortTransaction (object data)

9.23.2 Member Function Documentation

9.23.2.1 void Fuse::Transactional::abortTransaction (object *data*)

abort the transaction – argument is the return value from beginTransaction

Implemented in Fuse::LookupTransaction, and Fuse::OpenTransaction.

9.23.2.2 bool Fuse::Transactional::commitTransaction (object *data*)

complete the operation – argument is the return value from beginTransaction

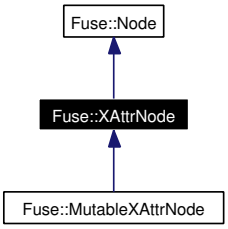
Implemented in Fuse::LookupTransaction, and Fuse::OpenTransaction.

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

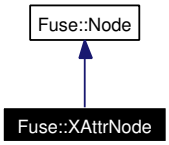
- Transactional.cs

9.24 Fuse::XAttrNode Interface Reference

Inheritance diagram for Fuse::XAttrNode:



Collaboration diagram for Fuse::XAttrNode:



9.24.1 Detailed Description

Interface for nodes supporting eXtended Attribute calls.

Public Member Functions

- int **GetXAttr** (string attrKey, FWBuffer output, int maxSize)

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

- XAttrNode.cs

Chapter 10

SULF Page Documentation

10.1 FuseWrapper bindings

10.1.1 FuseWrapper

FuseWrapper provides a C# layer around FUSE structures, and some helper classes which help in communicating with the FUSE kernel module. This is a very low-level API and probably not useful to you unless you are hacking Fuse.dll.

The FUSE kernel module (fuse.ko), communicates using various structures which are packed into memory in whatever order the C compiler says. Rather then reverse engineer what the packets look like, FuseWrapper provides methods to read and write structures to a memory buffer (FWBuffer) in C code, so that it should always match, no matter how the compiler does it, whatever byte order, etc.

So, when sending data to the kernel, the idea is to allocate the various strutcures required for the message (normally a fuse_out_header followed by the command specific struct), and then copy them to a buffer, eg:

```
FWBuffer buf = ...;

fuse_out_header header = new fuse_out_header();
header.unique = unique;
header.error = errorCode;

header.copyTo( buf );
```

Receiving data would require copying from the buffer to one of the structures and then using the struct accessors:

```
FWBuffer buf = ...;
```

```
fuse_in_header header = new fuse_in_header();
header.copyFrom( buf );

Console.WriteLine("serving request number {0}", header.unique);
```

Index