scaf

Generated by Doxygen 1.9.3

Chapter 1

scaf

Version

1.0.0

Created by Karl Miller for the Spring 2023 Code Jam at PennWest California.

Scaf is general purpose, command line, project initialization tool.

Scaf is purpose-agnostic and can be used to start-up ("scaffold") any type of project that has a directory structure.

Scaf works by maintaining directories of templates that the user supplies. When the user wants to scaffold a new project, they may use scaf to copy the contents from one of these directories into their current directory.

Chapter 2

Deprecated List

File Template.cpp

File Template.h

File TemplateTest.cpp

Chapter 3

Namespace Index

3.1 Namespace List

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

Chapter 6

Namespace Documentation

6.1 Filer Namespace Reference

Performs some filesystem operations and queries.

Classes

- struct copy_result

  Provides data about a recursive copy operation.

Functions

- copy_result copyRecursive (fs::path from, fs::path to)
- bool isEmpty (fs::path checkdir)
- bool clearDir (fs::path dirToClear)
- bool fillMapWithDirectories (const fs::path &p, map< string, fs::path > &m)
- bool fillVectorWithDirectories (const fs::path &p, vector< string > &m)
- bool fillVectorWithDirectories (const fs::path &p, vector< string > &v, string &filter_prefix)

6.1.1 Detailed Description

Performs some filesystem operations and queries.

Filer provides functions wrapping various filesystem operations, such as copyRecursive and isEmpty, used by Scaf.

Particularly, Filer ignores `.git` folders when determining whether a directory is empty and when copying recursively.

It also provides functions for getting information about the contents of a folder.

6.1.2 Function Documentation

6.1.2.1 clearDir()

```
bool Filer::clearDir (
             fs::path dirToClear )
```

Clears a directory of all its contents, except for '.git' folder.

Parameters

dirToClear	The directory to clear of contents.

Returns

true if path was valid and not a directory, false otherwise.

6.1.2.2 copyRecursive()

```
Filer::copy_result Filer::copyRecursive (
            fs::path from,
            fs::path to )
```

Recursively copies all files and folders in from to directory to.

Excludes .git folder and subdirs.

Precondition

from and to must be existing directories.

Parameters

from	The directory to copy from.
to	The directory to copy to.

Returns

The number of files copied.

6.1.2.3 fillMapWithDirectories()

```
bool Filer::fillMapWithDirectories (
            const fs::path & p,
            map< string, fs::path > & m )
```

Fills a map in-place with directory names inside of path.

Parameters

p	The path to use to populate the map.
m	The map to fill in-place.

Returns

true if path as valid and not a directory, false otherwise.

6.1.2.4 fillVectorWithDirectories() [1/2]

```
bool Filer::fillVectorWithDirectories (
            const fs::path & p,
            vector< string > & m )
```

Fills a vector in-place with directory names inside of path, then sorts the vector.

Parameters

p	The path to use to fill the vector.
v	The vector to fill in-place.

Returns

true if path was valid and not a directory, false otherwise.

6.1.2.5 fillVectorWithDirectories() [2/2]

```
bool Filer::fillVectorWithDirectories (
            const fs::path & p,
            vector< string > & v,
            string & filter_prefix )
```

Fills a vector in-place with directory names inside of path, then sorts the vector. Overload allows passing a string function to prefix-filter the names.

Parameters

p	The path to use to fill the vector.
v	The vector to fill in-place.
filter_prefix	A function to test each directory name. Directory names will only be added to v if callback(dirname) == true.

Returns

true if path was valid and not a directory, false otherwise.

6.1.2.6 isEmpty()

```
bool Filer::isEmpty (
             fs::path checkdir )
```

Checks whether a given directory is empty. (Excluding .git folder.)

Parameters

checkdir	The directory to examine.

Returns

Whether the given directory is empty.

Chapter 7

Class Documentation

7.1 Config Class Reference

loads scaf.conf.json from the process working directory.

```
#include <Config.h>
```

Public Member Functions

- Config ()
- Config (fs::path customPath)
- fs::path getPath ()
- void readConfig ()
- fs::path getTemplateDir ()
- bool setTemplateDir (fs::path newdir)
- bool hasTemplateDir ()
- string getInfo (string template_name)
- void setInfo (string key, string value)
- void writeConfig ()

7.1.1 Detailed Description

loads scaf.conf.json from the process working directory.

Config provides the functions and classes needed to load scaf's configuration.

When it is constructed, it looks for `scaf.config.json` in the process working directory. If it doesn't exist, it is created. If it is exists as a directory, a runtime error is thrown.

Otherwise, it parses that configuration file to load scaf's saved settings.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 Config() [1/2]

```
Config::Config ( )
```

Initializes the Config. Looks for scaf.config.json in the process working directory. If it doesn't exist, it is created. If it exists as a directory, a runtime error is thrown.

7.1.2.2 Config() [2/2]

```
Config::Config (
             fs::path customPath )
```

Generally the 0 parameter constructor should be called. Passing a custom configuration is useful for testing purposes.

Parameters

customPath	The path to configuration file.

7.1.3 Member Function Documentation

7.1.3.1 getInfo()

```
string Config::getInfo (
             string template_name )
```

Gets the info for a given template. Returns empty string if no such info exists.

7.1.3.2 getPath()

```
fs::path Config::getPath ( )
```

Returns the path that was loaded at the time of Config's construction.

7.1.3.3 getTemplateDir()

```
fs::path Config::getTemplateDir ( )
```

Returns the template directory read from the config file.

7.1.3.4 hasTemplateDir()

```
bool Config::hasTemplateDir ( )
```

Checks if Config has a template directory loaded.

7.1.3.5 readConfig()

```
void Config::readConfig ( )
```

Uses json library to read the configuration file.

If the json object is misconfigured, it may simply skip parsing that step and print an error to the console.

7.1.3.6 setInfo()

```
void Config::setInfo (
             string key,
             string value )
```

Sets the info for a given template.

Parameters

key	The name of the template.
value	The new info to set.

7.1.3.7 setTemplateDir()

```
bool Config::setTemplateDir (
             fs::path newdir )
```

Sets a new template directory. Will not be set and an error will be printed if the directory doesn't exist. If the directory is relative, it will be converted to absolute.

Parameters

newdir	The new template directory.

Returns

true if directory could be set. false otherwise.

7.1.3.8 writeConfig()

```
void Config::writeConfig ( )
```

Writes the config as a json. Called on program end to record updates in the config file. Pretty-prints the JSON with indentation.

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

- src/Config.h
- src/Config.cpp

7.2 ConfigTest Class Reference

Tests the Config class.

Public Member Functions

- **ConfigTest** (std::ostream &out, int verbose_level=QUnit::verbose)
- int **run** ()

7.2.1 Detailed Description

Tests the Config class.

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

- tests/ConfigTest.cpp

7.3 Filer::copy_result Struct Reference

Provides data about a recursive copy operation.

```
#include <Filer.h>
```

Public Attributes

- int filescopied
- int folderscopied
- bool gitskipped

7.3.1 Detailed Description

Provides data about a recursive copy operation.

Provides data about a copy operation.

Returned by Filer::copyRecursive.

7.3.2 Member Data Documentation

7.3.2.1 filescopied

```
int Filer::copy_result::filescopied
```

The number of files copied.

7.3.2.2 folderscopied

```
int Filer::copy_result::folderscopied
```

The number of folders copied.

7.3.2.3 gitskipped

```
bool Filer::copy_result::gitskipped
```

Whether a '.git' folder was skipped.

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

- src/Filer.h

7.4 FilerTest Class Reference

Tests the Filer namespace.

Public Member Functions

- **FilerTest** (std::ostream &out, int verbose_level=QUnit::verbose)
- int **run** ()

7.4.1 Detailed Description

Tests the Filer namespace.

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

- tests/FilerTest.cpp

7.5 Scaf Class Reference

Parses and executes the command line arguments.

```
#include <Scaf.h>
```

Public Member Functions

- Scaf ()
- Scaf (filesystem::path config_path)
- bool Start (int argc, char **argv)

7.5.1 Detailed Description

Parses and executes the command line arguments.

Scaf drives the program execution. It loads the config and parses the command line arguments into the correct commands.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Scaf() [1/2]

```
Scaf::Scaf ( )
```

Constructor.

7.5.2.2 Scaf() [2/2]

```
Scaf::Scaf (
              filesystem::path config_path )
```

Constructor with config override. For use with testing.

Parameters

config_path	A different configuration path to provide to the Config object.

7.5.3 Member Function Documentation

7.5.3.1 Start()

```
bool Scaf::Start (
              int argc,
              char ** argv )
```

Begins the parse process with the command-line args.

Parameters

argc	The arg count.
argv	The arg values.

Returns

True for succesful parse. False if there were errors.

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

- src/Scaf.h
- src/Scaf.cpp

7.6 Template Class Reference

Holds information about a template. Not used!

```
#include <Template.h>
```

Public Member Functions

- **Template** (string a_path, string an_alias, string some_info)
- **Template** (string a_path, string an_alias)
- **Template** (string a_path)
- string getPath ()
- string getAlias ()
- void setAlias (string an_alias)
- string getInfo ()
- void setInfo (string some_info)

7.6.1 Detailed Description

Holds information about a template. Not used!

Template holds information about a given template, including its folder location and saved-user info.

NOTE: TEMPLATE IS NOT CURRENTLY ACTUALLY USED IN THE PROGRAM!

Instead of instancing templates, scaf scans the directories within the root folder to get template names instead! As Config reads the saved program data, it instances Templates. (It doesn't!)

Template has not been removed because it is well-tested and may be useful in a future refactor.

7.6.2 Member Function Documentation

7.6.2.1 getAlias()

```
string Template::getAlias ( )
```

Gets the template's name.

7.6.2.2 getInfo()

```
string Template::getInfo ( )
```

If info is an empty string, returns a message saying that there is no info.

7.6.2.3 getPath()

```
string Template::getPath ( )
```

Gets the template's absolute path on the file system.

7.6.2.4 setAlias()

```
void Template::setAlias (
             string an_alias )
```

Sets the alias.

7.6.2.5 setInfo()

```
void Template::setInfo (
             string some_info )
```

Sets the template's info.

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

- src/Template.h
- src/Template.cpp

7.7 TemplateTest Class Reference

Tests the Template class.

Public Member Functions

- **TemplateTest** (std::ostream &out, int verbose_level=QUnit::verbose)
- int **run** ()

7.7.1 Detailed Description

Tests the Template class.

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

- tests/TemplateTest.cpp

Chapter 8

File Documentation

8.1 src/Config.cpp File Reference

Implementation for Config class.

```
#include <filesystem>
#include <fstream>
#include <Config.h>
#include <json.hpp>
```

Typedefs

- using **json** = nlohmann::json

Functions

- fs::path GetFullExePath ()

8.1.1 Detailed Description

Implementation for Config class.

Author

Karl Miller

Date

April 2023

8.1.2 Function Documentation

8.1.2.1 GetFullExePath()

```
fs::path GetFullExePath ( )
```

GetFullExePath is used to get the actual fs location of scaf. This is different from where the current working directory is. It's often referred to as the process working directory. Apparently pwd functions are not cross-platform in the built-in libraries, so directives are used to make scaf compatible on linux and windows.

This is used to locate scaf's config in the same directory as the scaf executable.

Returns

A path for the folder containing scaf.exe

From `https://stackoverflow.com/questions/50889647/best-way-to-get-exe-folder-path`

8.2 src/Config.h File Reference

Declarations for Config class.

```
#include <string>
#include <filesystem>
#include <iostream>
#include <map>
```

Classes

- class Config
  loads scaf.conf.json from the process working directory.

Functions

- fs::path GetFullExePath ()

Variables

- const string dirKey = "templateDir"
- const string infoKey = "infos"

8.2.1 Detailed Description

Declarations for Config class.

Author

Karl Miller

Date

April 2023

8.2.2 Function Documentation

8.2.2.1 GetFullExePath()

```
fs::path GetFullExePath ( )
```

GetFullExePath is used to get the actual fs location of scaf. This is different from where the current working directory is. It's often referred to as the process working directory. Apparently pwd functions are not cross-platform in the built-in libraries, so directives are used to make scaf compatible on linux and windows.

This is used to locate scaf's config in the same directory as the scaf executable.

Returns

A path for the folder containing scaf.exe

From `https://stackoverflow.com/questions/50889647/best-way-to-get-exe-folder-path`

8.2.3 Variable Documentation

8.2.3.1 dirKey

```
const string dirKey = "templateDir"
```

The JSON key for the template directory. Should key a string.

8.2.3.2 infoKey

```
const string infoKey = "infos"
```

The JSON key for the saved infos. Should key an object where the keys for all values are strings.

8.3 Config.h

Go to the documentation of this file.

```
#pragma once
#include <string>
#include <filesystem>
#include <iostream>
#include <map>

using namespace std;
namespace fs = std::filesystem;

const string dirKey = "templateDir";
const string infoKey = "infos";

fs::path GetFullExePath();

class Config {
    private:
        fs::path configPath;
        fs::path templateDir;
        map<string, string> infos;

    public:
        Config();
        Config(fs::path customPath);

        fs::path getPath();

        void readConfig();

        fs::path getTemplateDir();

        bool setTemplateDir(fs::path newdir);

        bool hasTemplateDir();

        string getInfo(string template_name);

        void setInfo(string key, string value);


        void writeConfig();
};

```

8.4 src/Filer.cpp File Reference

Definitions for Filer namespace.

```
#include "Filer.h"
#include <iostream>
#include <bits/stdc++.h>
```

8.4.1 Detailed Description

Definitions for Filer namespace.

Program entry point.

Author

Karl Miller

Date

April 2023

8.5 src/Filer.h File Reference

Declarations for Filer namespace.

```
#include <filesystem>
#include <map>
#include <vector>
#include <functional>
```

Classes

- struct Filer::copy_result

  Provides data about a recursive copy operation.

Namespaces

- namespace Filer

  Performs some filesystem operations and queries.

Functions

- copy_result Filer::copyRecursive (fs::path from, fs::path to)
- bool Filer::isEmpty (fs::path checkdir)
- bool Filer::clearDir (fs::path dirToClear)
- bool Filer::fillMapWithDirectories (const fs::path &p, map< string, fs::path > &m)
- bool Filer::fillVectorWithDirectories (const fs::path &p, vector< string > &m)
- bool Filer::fillVectorWithDirectories (const fs::path &p, vector< string > &v, string &filter_prefix)

8.5.1 Detailed Description

Declarations for Filer namespace.

Author

Karl Miller

Date

April 2023

8.6 Filer.h

Go to the documentation of this file.

```
#pragma once
#include <filesystem>
#include <map>
#include <vector>
#include <functional>
using namespace std;
namespace fs = std::filesystem;


namespace Filer {

    typedef struct {
        int filescopied;
        int folderscopied;
        bool gitskipped;
    } copy_result;
    copy_result copyRecursive(fs::path from, fs::path to);

    bool isEmpty(fs::path checkdir);

    bool clearDir(fs::path dirToClear);

    bool fillMapWithDirectories(const fs::path &p, map<string, fs::path>& m);

    bool fillVectorWithDirectories(const fs::path &p, vector<string>& m);

    bool fillVectorWithDirectories(const fs::path &p, vector<string>& v, string &filter_prefix);
}
```

8.7 src/Scaf.h File Reference

Declarations for Scaf class.

```
#include <vector>
#include <filesystem>
#include "Config.h"
#include "Filer.h"
```

Classes

- class Scaf

    Parses and executes the command line arguments.

Functions

- void stringLower (string &s)
- bool promptYN (bool default_yn)
- void printCopyResult (Filer::copy_result &copied)

8.7.1 Detailed Description

Declarations for Scaf class.

Author

Karl Miller

Date

April 2023

8.7.2 Function Documentation

8.7.2.1 printCopyResult()

```
void printCopyResult (
            Filer::copy_result & copied )
```

Prints a copy result on 3 lines with correct pluralization.

Parameters

copied	The result of a recursive directory copy.

8.7.2.2 promptYN()

```
bool promptYN (
            bool default_yn )
```

Looks for a 'y' or 'n'.

8.7.2.3 stringLower()

```
void stringLower (
            string & s )
```

Converts a string to all lower-case in place.

8.8 Scaf.h

Go to the documentation of this file.

```
#pragma once
#include <vector>
#include <filesystem>

#include "Config.h"
#include "Filer.h"

using namespace std;

void stringLower(string& s);

bool promptYN(bool default_yn);

void printCopyResult(Filer::copy_result & copied);

class Scaf {
    public:
        Scaf();
        Scaf(filesystem::path config_path);
        bool Start(int argc, char ** argv);


    private:
        Config config;
        bool Help(int index, vector<string>& args);

        bool Root(int index, vector<string>& args);

        bool Add(int index, vector<string>& args);

        bool Load(int index, vector<string>& args);

        bool List(int index, vector<string>& args);

        bool Info(int index, vector<string>& args);

        bool Set(int index, vector<string>& args);

        bool Remove(int index, vector<string>& args);

        bool Rename(int index, vector<string>& args);

        void printHelp();

        void printHelpRoot();
        void printHelpAdd();
        void printHelpInfo();
        void printHelpSet();
        void printHelpRemove();
        void printHelpLoad();
        void printHelpRename();
        void printHelpList();

};
```

8.9 src/Template.cpp File Reference

Definitions for Template class (deprecated).

```
#include "Template.h"
#include <filesystem>
```

8.9.1 Detailed Description

Definitions for Template class (deprecated).

Author

Karl Miller

Date

April 2023

Deprecated

Note

Template is not currently used in the program!

It may be utilized on a future iteration and refactor.

8.10 src/Template.h File Reference

Declarations for Template class.

```
#include <string>
#include <vector>
```

Classes

- class Template

  Holds information about a template. Not used!

8.10.1 Detailed Description

Declarations for Template class.

Author

Karl Miller

Date

April 2023

Deprecated

Note

Template is not currently used in the program!

8.11 Template.h

Go to the documentation of this file.

```
#pragma once
#include <string>
#include <vector>

using namespace std;

class Template {
    private:
        string path;
        string alias;
        string info;

    public:

        Template(string a_path, string an_alias, string some_info);
        Template(string a_path, string an_alias);
        Template(string a_path);
        Template();

        string getPath();

        string getAlias();

        void setAlias(string an_alias);

        string getInfo();

        void setInfo(string some_info);

};
```

8.12 tests/ConfigTest.cpp File Reference

Provides tests for Config. See Config.h.

```
#include "QUnit.hpp"
#include <iostream>
#include <filesystem>
#include <fstream>
#include "Config.h"
```

Classes

- class ConfigTest

  Tests the Config class.

Variables

- const string **tmp_conf** = "./tmp/testconf.json"
- const string **tmp_tdir** = "./tmp/tmpl"

8.12.1 Detailed Description

Provides tests for Config. See Config.h.

Author

Karl Miller

Date

April 2023

8.13 tests/FilerTest.cpp File Reference

Provides tests for Filer. See Filer.h.

```
#include <iostream>
#include <filesystem>
#include <fstream>
#include "QUnit.hpp"
#include "Filer.h"
```

Classes

- class FilerTest

  Tests the Filer namespace.

Variables

- const string **flr_tmp_tdir** = "./tmp/tmpl2"
- const string **flr_tmp_tdir2** = "./tmp/tmpl2/tt"
- const string **flr_tmp_fl** = "./tmp/tmpl2/t.x"
- const string **flr_tmp_fl2** = "./tmp/tmpl2/tt/t2.x"
- const string **flr_tmp_odir** = "./tmp/tout"

8.13.1 Detailed Description

Provides tests for Filer. See Filer.h.

Author

Karl Miller

Date

April 2023

8.14 tests/TemplateTest.cpp File Reference

Provides tests for Template. See Template.h.

```
#include "QUnit.hpp"
#include <iostream>
#include "Template.h"
```

Classes

- class TemplateTest

  Tests the Template class.

8.14.1 Detailed Description

Provides tests for Template. See Template.h.

Author

Karl Miller

Date

April 2023

Note

TemplateTest is deprecated! Template is not currently used in the program!

Deprecated

8.15 tests/testAll.cpp File Reference

Calls appropriate unit tests by parsing command line arguments.

```
#include <string>
#include <iostream>
#include "TemplateTest.cpp"
#include "ConfigTest.cpp"
#include "FilerTest.cpp"
```

Functions

- int main (int argc, char **argv)

  Parses command line arguments to run all or one test.

8.15.1 Detailed Description

Calls appropriate unit tests by parsing command line arguments.

Author

Karl Miller

Date

April 2023

8.15.2 Function Documentation

8.15.2.1 main()

```
int main (
          int argc,
          char ** argv )
```

Parses command line arguments to run all or one test.

Runs the tests.

Format for CLI args are as follows:

test { template | all } { noisy | normal | quiet | silent }

Defaults to normal verbosity and all tests.

Index