C++17'S STD::FILESYSTEM OVERVIEW

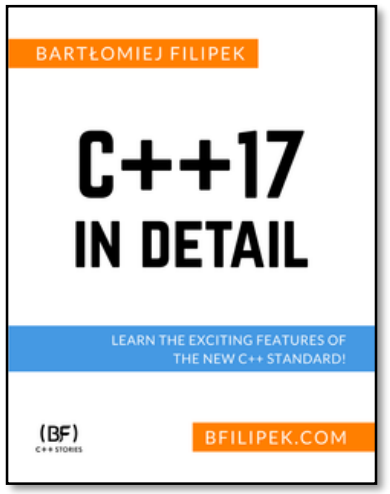
What's that? How does it work?

16th April 2019

Bartłomiej Filipek, bfilipek.com

About me

- See my coding blog at: www.bfilipek.com
- ~12y coding experience
- Microsoft MVP,
- C++ ISO Member (learning!)
- Currently @Xara.com
  - Text related features for advanced document editors
- Somehow addicted to C++ ☺



C++17 In Detail

Xara Cloud Demo

cpp-polska.pl
BLOG PROGRAMISTYCZNY

The plan

- Demos & Simplification
- The path class
- Directory_entry
- Support functions
- Permissions
- Errors & Exceptions
- More Examples
- Summary

Demo – file size

```
ifstream testFile("test.file", ios::binary);
const auto begin = myfile.tellg();
testFile.seekg(0, ios::end);
const auto end = testFile.tellg();
const auto fsize = (end - begin);
```

```
HANDLE hFile = /* get file/ open/create */

LARGE_INTEGER size;
if (!GetFileSizeEx(hFile, &size)) {
    CloseHandle(hFile);
    return -1;
}
```

Filesize – with std::filesystem

```
try {
    auto fsize = std::filesystem::file_size("test.file");
}
catch (fs::filesystem_error& ex) {
    std::cout << ex.what() << '\n';
}
```

```
std::error_code ec{};
auto size = std::filesystem::file_size("a.out", ec);
if (ec == std::error_code{})
    std::cout << "size: " << size << '\n';
else
    std::cout << "error when accessing test file, size is: "
              << size << " message: " << ec.message() << '\n';
```

Demo – directory iteration

```c
int main(int argc, const char**argv) {
    struct dirent *entry = nullptr;
    DIR *dp = nullptr;

    dp = opendir(argc > 1 ? argv[1] : "/");
    if (dp != nullptr) {
        while ((entry = readdir(dp)))
        printf("%s\n", entry->d_name);
    }

    closedir(dp);
    return 0;
}
```

```c
WIN32_FIND_DATA FindFileData;
HANDLE hFind = FindFirstFile(/*path*/, &FindFileData);
if (hFind == INVALID_HANDLE_VALUE) {
    printf("FindFirstFile failed (%d)\n", GetLastError());
    return;
}

do {
    if (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
        _tprintf(TEXT("  %s   <DIR>\n"), FindFileData.cFileName);
    else
        _tprintf(TEXT("  %s \n"), FindFileData.cFileName);
} while (FindNextFile(hFind, &FindFileData) != 0);

FindClose(hFind);
```

Directory iteration with std::filesystem

```
for (const auto& entry : std::filesystem::directory_iterator(pathToShow)) {
    const auto filenameStr = entry.path().filename().string();
    if (entry.is_directory()) {
        std::cout << "dir:  " << filenameStr << '\n’;
    }
    else if (entry.is_regular_file()) {
        std::cout << "file: " << filenameStr << '\n’;
    }
    else
        std::cout << "??    " << filenameStr << '\n';
}
```

Demo code:
http://coliru.stacked-crooked.com/a/f7d38eece2272502

Compiler Support

- `#include <filesystem>, namespace std::filesystem`
- Adapted from BOOST and TS

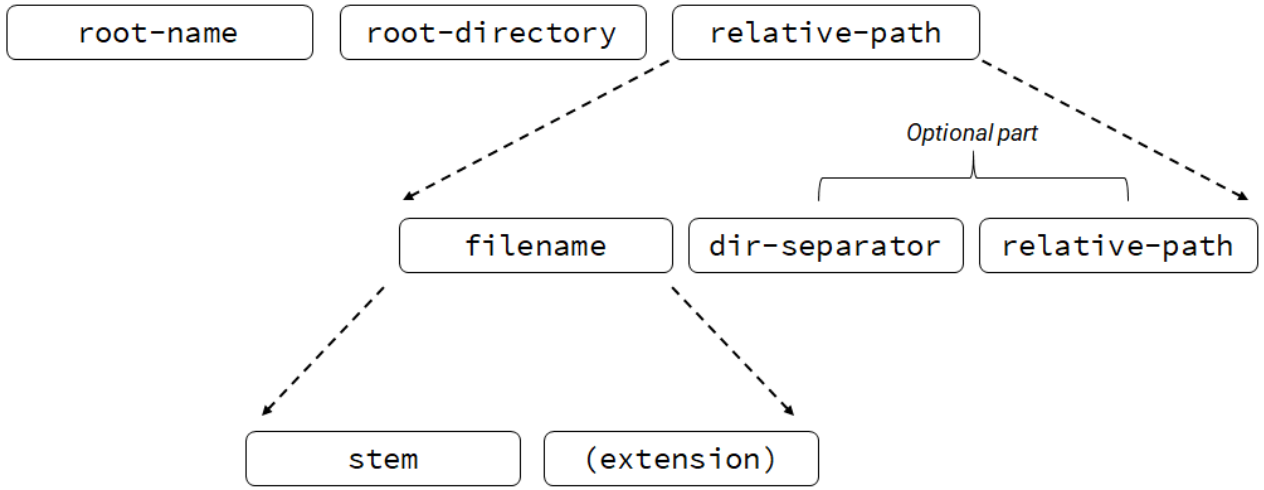
- GCC 8.0 + lstdc++fs
- Clang 7.0 + lstdc++fs (??)
- MSVC VS 2017 15.7

- Plus experimental (TS) (`#include <experimental/filesystem>`)
  - GCC 5.3, Clang 3.9 and VS 2012

General overview

- Path
- Dir entry
- Support functions
- Permissions
- Errors & Exceptions

The Path class



C:/temp/data/file.txt
/dev/notes/file.txt
/tmp/743984iuhgj/backup.bin

Path – info and query

Method	Description
path::root_name()	returns the root-name of the path
path::root_directory()	returns the root directory of the path
path::root_path()	returns the root path of the path
path::relative_path()	returns path relative to the root path
path::parent_path()	returns the path of the parent path
path::filename()	returns the filename path component
path::stem()	returns the stem path component
path::extension()	returns the file extension path component

Query name	Description
path::has_root_path()	queries if a path has a root
path::has_root_name()	queries if a path has a root name
path::has_root_directory()	checks if a path has a root directory
path::has_relative_path()	checks if a path has a relative path component
path::has_parent_path()	checks if a path has a parent path
path::has_filename()	checks if a path has a filename
path::has_stem()	checks if a path has a stem component
path::has_extension()	checks if a path has an extension

Path – example, path info

http://coliru.stacked-crooked.com/a/a19df0279ae8c8fe

Path - operations

Operation	Description
path::append()	appends one path to the other, with a directory separator
path::concat()	concatenates the paths, without a directory separator
path::clear()	erases the elements and makes it empty
path::remove_filename()	removes the filename part from a path
path::replace_filename()	replaces a single filename component
path::replace_extension()	replaces the extension
path::swap()	swaps two paths
path::compare()	compares the lexical representations of the path and another path, returns an integer
path::empty()	checks if the path is empty

Path - compare

- Compares native representation of the path
- From cppreference: `path::compare(path p)`
  - If `root_name().native().compare(p.root_name().native())` is nonzero, returns that value.
  - Otherwise, if `has_root_directory() != p.has_root_directory()`, returns a value less than zero if `has_root_directory()` is false and a value greater than zero otherwise.
  - Otherwise: Comparison is performed element-wise, as if by iterating both paths from `begin()` to `end()` and comparing the result of `native()` for each element.

Demo code:
http://coliru.stacked-crooked.com/a/9dbbae456ea9ce05

Path ops, example

```
// append:
fs::path p1{ "C:\\temp" };
p1 /= "user";
p1 /= "data";
cout << p1 << '\n';

// concat:
fs::path p2{"C:\\temp\\“};
p2 += "user";
p2 += "data";
cout << p2 << '\n';
```

```
The output:
C:\temp\user\data
C:\temp\userdata
```

Path Formats and Conversion

- Native and generic format
- On Windows \ rather than /
- On Windows `wstring (wchar_t)` rather than `string (char)`
  - fs::*path*::*value_type*
  - fs::*path*::*preferred_separator*
- Conversions:
  - *path*::*u8string*(), *path*::*wstring*(), *path*::*string*(), *path*::*u16string*(), *path*::*u32string*()

Directory Entry

- Models path + flags + cache
- directory_entry::path()

Operation	Description
directory_entry::assign()	replaces the path inside the entry and calls refresh() to update the cached attributes
directory_entry::replace_filename()	replaces the filename inside the entry and calls refresh() to update the cached attributes
directory_entry::refresh()	updates the cached attributes of a file
directory_entry::exists()	checks if a directory entry points to existing file system object
directory_entry::is_block_file()	returns true if the file entry is a block file
directory_entry::is_character_file()	returns true if the file entry is a character file
directory_entry::is_directory()	returns true if the file entry is a directory
directory_entry::is_fifo()	returns true if the file entry refers to a named pipe
directory_entry::is_other()	returns true if the file entry is refers to another file type
directory_entry::is_regular_file()	returns true if the file entry is a regular file
directory_entry::is_socket()	returns true if the file entry is a named IPC

Directory iterators (input iterators)

```
for (auto const & entry : fs::directory_iterator(pathToShow)) {
      ...
}
```

```
std::filesystem::recursive_directory_iterator dirpos{ inPath };

std::copy_if(begin(dirpos), end(dirpos), std::back_inserter(paths), predicate);
```

- Plus iteration options: follow sym links, skip permission denied
- recursive_directory_iterator::depth, recursion_pending
- http://coliru.stacked-crooked.com/a/27068ef47c665e13

Support functions - query

function	description
filesystem::is_block_file()	checks whether the given path refers to block device
filesystem::is_character_file()	checks whether the given path refers to a character device
filesystem::is_directory()	checks whether the given path refers to a directory
filesystem::is_empty()	checks whether the given path refers to an empty file or directory
filesystem::is_fifo()	checks whether the given path refers to a named pipe
filesystem::is_other()	checks whether the argument refers to another file
filesystem::is_regular_file()	checks whether the argument refers to a regular file
filesystem::is_socket()	checks whether the argument refers to a named IPC socket
filesystem::is_symlink()	checks whether the argument refers to a symbolic link
filesystem::status_known()	checks whether file status is known
filesystem::exists()	checks whether path refers to existing file system object
filesystem::file_size()	returns the size of a file
filesystem::last_write_time()	gets or sets the time of the last data modification

Support Functions – path related

function	description
filesystem::absolute()	composes an absolute path
filesystem::canonical(), weakly_canonical()	composes a canonical path
filesystem::relativeproximate()	composes a relative path
filesystem::current_path()	returns or sets the current working directory
filesystem::equivalent()	checks whether two paths refer to the same file system object

Demo code:
http://coliru.stacked-crooked.com/a/11f9d62fcec54a82

Support Functions – dir & file management

function name	description
filesystem::copy()	copies files or directories
filesystem::copy_file()	copies file contents
filesystem::copy_symlink()	copies a symbolic link
filesystem::create_directory(), filesystem::create_directories()	creates new directory
filesystem::create_hard_link()	creates a hard link
filesystem::create_symlink(), filesystem::create_directory_symlink()	creates a symbolic link
filesystem::hard_link_count()	returns the number of hard links referring to the specific file
filesystem::permissions()	modifies file access permissions
filesystem::read_symlink()	obtains the target of a symbolic link
filesystem::remove(), filesystem::remove_all()	removes a single file or whole directory recursively with all its content
filesystem::rename()	moves or renames a file or directory
filesystem::resize_file()	changes the size of a regular file by truncation or zero-fill
filesystem::space()	determines available free space on the file system
filesystem::status(), filesystem::symlink_status()	determines file attributes, determines file attributes, checking the symlink target
filesystem::temp_directory_path()	returns a directory suitable for temporary files

File permissions

```
std::ostream& operator<< (std::ostream& stream, fs::perms p)
{
stream << "owner: "
       << ((p & fs::perms::owner_read) != fs::perms::none ? "r" : "-")
       << ((p & fs::perms::owner_write) != fs::perms::none ? "w" : "-")
       << ((p & fs::perms::owner_exec) != fs::perms::none ? "x" : "-");
stream << " group: "
       << ((p & fs::perms::group_read) != fs::perms::none ? "r" : "-")
       << ((p & fs::perms::group_write) != fs::perms::none ? "w" : "-")
       << ((p & fs::perms::group_exec) != fs::perms::none ? "x" : "-");
stream << " others: "
       << ((p & fs::perms::others_read) != fs::perms::none ? "r" : "-")
       << ((p & fs::perms::others_write) != fs::perms::none ? "w" : "-")
       << ((p & fs::perms::others_exec) != fs::perms::none ? "x" : "-");
return stream;
}
```

File permissions

```
// remove "owner_read"
fs::permissions(myPath, fs::perms::owner_read, fs::perm_options::remove);

// add "owner_read"
fs::permissions(myPath, fs::perms::owner_read, fs::perm_options::add);
```

From Microsoft Docs filesystem documentation:
The supported values are essentially “readonly” and all. For a readonly file, none of the *_write bits are set. Otherwise the all bit (0777) is set.

Exceptions & errors

- Two overloads: one with exceptions and one with error codes (possibly with noexcept)

```
uintmax_t file_size(const path& p);
uintmax_t file_size(const path& p, error_code& ec) noexcept;
```

```
std::error_code ec{};
auto size = std::filesystem::file_size(testPath, ec); // <<
if (ec == std::error_code{})
    std::cout << "size: " << size << '\n';
else
    std::cout << "error when accessing test file, size is: "
              << size << " message: " << ec.message() << '\n';
```

Demo – directory watcher

- All credits goes to Solarian Programmer
  - https://solarianprogrammer.com/2019/01/13/cpp-17-filesystem-write-file-watcher-monitor/
  - https://github.com/sol-prog/cpp17-filewatcher
  - https://www.reddit.com/r/cpp/comments/4rb294/is_there_any_particular_reason_why/

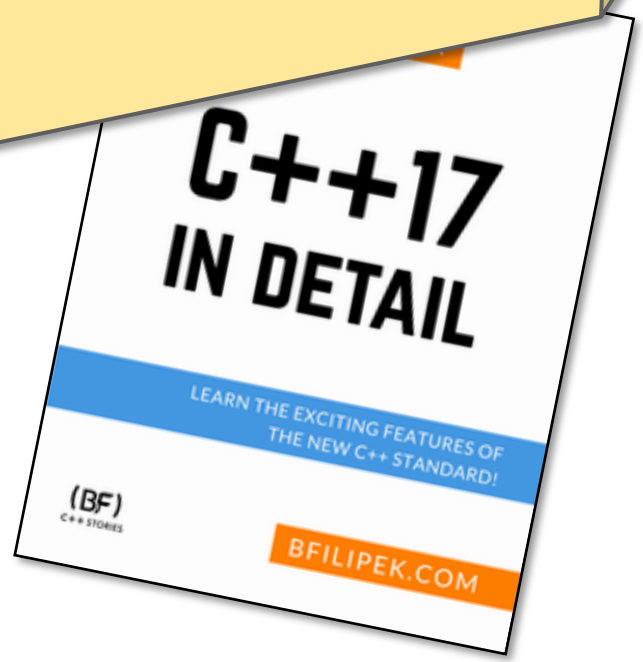
Perf notes

- System calls are expensive
  - Try to use directory_entry records if already there
- Directory iterators are only „input iterators”, so not ready for par execution
- Remember that path is stored in a native representation, so calling .string() might perform extra conversions (like on Windows, that uses wstring)

Summary

- #include <filesystem>
- Path
- Drectory_entry
- Directory_iterator, recursive_directory_iterator
- Support functions
- Adapted from BOOST and TS

70% off
Limited offer!
(till the end of April)



https://leanpub.com/cpp17indetail/c/cppkrk

c/cppkrk

https://cpp-polska.pl/

cpp-polska.pl

BLOG PROGRAMISTYCZNY

http://cpp-polska.pl/slack