

# Home Delivery

[Rob Hansen](#)

Documenting Home Delivery 1.1

## Changelog:

- 1.0: initial release. No PDF documentation was provided.
- 1.1: code restructuring, no new features. Full documentation provided. Migrated to a Noweb-based build system.

# Contents

<b>1</b>	<b>Preliminaries</b>	<b>1</b>
1.1	Foreword	1
1.2	Usage	1
1.3	Versioning	1
1.4	Homepage and bug reporting	2
1.5	Licensing	2
1.6	About this document	2
1.7	The application	2
1.8	Hacking	2
<b>2</b>	<b>Cargo.toml</b>	<b>5</b>
<b>3</b>	<b>config.yaml</b>	<b>7</b>
<b>4</b>	<b>main.rs</b>	<b>9</b>
4.1	Imports	9
4.2	Argument parsing	10
4.3	Function: <code>get_current_deliverables</code>	11
4.4	Function: <code>main</code>	12
4.4.1	Housekeeping	12
4.4.2	Sanity checking	13
4.4.3	Initializing logging	13
4.4.4	Main loop	13
<b>5</b>	<b>utility.rs</b>	<b>15</b>
5.1	Imports	15
5.2	Static initialization	16
5.3	Function: <code>sanity_check</code>	17
5.4	Function: <code>sleep_to_top_of_minute</code>	18
5.5	Function: <code>read_directory</code>	19
5.6	Function: <code>find_matching_files</code>	19
5.6.1	Compiling entries	20
5.6.2	Error reporting	21
5.7	Function: <code>capture_to_rfc3339</code>	22
5.7.1	Binding initialization	23

---

5.7.2	Sanity check	23
5.8	Function: <code>filenames_with_timestamps</code>	24
5.9	Function: <code>deliver</code>	25



# Chapter 1

## Preliminaries

### 1.1 Foreword

A friend came to me with the following statement of work:

“Create a command-line application for Windows 11 or later which has no dependencies. It must take two parameters, SRC and DST. Upon execution it looks for any files in SRC which match a certain pattern. That pattern denotes a future date and time in Universal Coordinated Time (UTC, also called ‘Zulu’). For instance, the file FOO-0800.TXT must be delivered from SRC to DST at 0800Z on the current day. If it is already past 0800Z, it must be delivered immediately. Further, the time may be prefixed with a number followed by D, denoting a day or days in the future: FOO-3D0800.TXT must be delivered at 0800Z three days hence. Zero is a valid day parameter. Upon making a round of deliveries, the program must wait for one minute and check again. New files may be introduced to SRC during program execution and must be recognized and delivered appropriately.”

This application meets all requirements.

### 1.2 Usage

```
home_delivery --source SRC --destination DST --config CFG
```

... where SRC is the path to the source directory, DST the path to the destination directory, and CFG the path to the logger configuration file. E.g., to move files from source\_dir to dest\_dir using longger configuration log\_config.yaml, you would type:

```
home_delivery --source source_dir --destination dest_dir --config log_config.yaml
```

You may also use -s, -d, and -c as shortcuts.

It also understands the -h/--help and -V/--version flags.

### 1.3 Versioning

This is Home Delivery 1.1 (March 11, 2025).

## 1.4 Homepage and bug reporting

This application is hosted on [GitHub](#).<sup>1</sup> There, you can find an “Issues” page, which is the project bug tracker.<sup>2</sup>

## 1.5 Licensing

The source code is released under the [Apache 2.0 license](#);<sup>3</sup> the documentation, the [Creative Commons 4.0 Attribution-NoDerivs](#) license.<sup>4</sup>

## 1.6 About this document

This is a literate program in the style of Donald E. Knuth, using Norman Ramsey’s excellent [Noweb system](#).<sup>5</sup> The program and documentation are a fused whole found in `src/code.nw`, and prior to Git checkin the documentation and code are extracted and generated so that they may always be in sync.

Specifically, the code in this document was used to create files with the following MD5 checksums:

Table 1.1:

<code>src/main.rs</code>	5595A3EE46465F5C4D422CC937017365
<code>src/utility.rs</code>	39AC781F5CC53977EE39575DBA4AE098
<code>Cargo.toml</code>	D0A9343DE3C9ABE27F0A2FA5F6FEC79B
<code>config.yaml</code>	16BF03E7976C0C25A6E815ACB0D6DD28

## 1.7 The application

This application was written using Rust 1.85 and should compile cleanly on Windows 11 (or later) targets. It has also been successfully tested on Fedora Linux 41 and MacOS Sequoia 15.3.1. There are no per-platform differences.

## 1.8 Hacking

Two different build systems are provided: a Bash script for UNIX hosts (to include MacOS) and a PowerShell 7.5 script (for Windows hosts). Run these noweb-build scripts to extract source code from the NoWeb file and build the documentation and application.

You will need:

<sup>1</sup>[https://github.com/rjhansen/home\\_delivery](https://github.com/rjhansen/home_delivery)

<sup>2</sup>[https://github.com/rjhansen/home\\_delivery/issues](https://github.com/rjhansen/home_delivery/issues)

<sup>3</sup><https://www.apache.org/licenses/LICENSE-2.0>

<sup>4</sup><https://creativecommons.org/licenses/by-nd/4.0/deed.en>

<sup>5</sup><https://www.cs.tufts.edu/~nr/noweb/>

1. A Rust development environment on your system PATH
2. A Texlive installation on your system PATH
3. Either the Bourne Again shell or PowerShell 7.5 or later
4. **MacOS only:** gsed installed from Homebrew





## Chapter 2

### Cargo.toml

The following table lists the third-party libraries we depend upon for compilation,<sup>1</sup> their required version numbers, and a brief description of why we need them. Each library may bring in compile-time dependencies of its own.

Table 2.1:

<b>clap</b>	4.5.x	Command line parsing
<b>log</b>	0.4.x	Standard logging façade
<b>log4rs</b>	1.3.x	The particular logging implementation
<b>regex</b>	1.11.x	Provides regular expressions
<b>lazy_static</b>	1.5.x	Provides static initialization
<b>chrono</b>	0.4.x	Adds UTC capabilities to the standard DateTime class

```
5  <Cargo.toml 5>≡
    [package]
    name = "home_delivery"
    version = "1.1.0"
    edition = "2021"
    authors = ["Rob Hansen <rob@hansen.engineering>"]
    license = "Apache-2.0"
    readme = "README.md"
    description = "Moves files between two directories on a schedule"
    homepage = "https://github.com/rjhansen/home_delivery"
    repository = "https://github.com/rjhansen/home_delivery.git"
```

---

<sup>1</sup>Not execution. Home Delivery is free of external dependencies in its runtime, but it needs these libraries to compile.

```
[dependencies]
clap = { version = "4.5", features = ["derive"] }
log = "0.4"
log4rs = "1.3"
regex = "1.11"
lazy_static = "1.5"
chrono = "0.4"

[[bin]]
name = "home_delivery"
```

## Chapter 3

### config.yaml

This is a simple logging configuration file. To make sense of it, please see the `log4rs` documentation.

```
7 <config.yaml 7>≡
  appenders:
    screen:
      kind: console
      encoder:
        pattern: "{h({d(%Y-%m-%dT%H:%M:%SZ)(utc)} - {l}: {m}{n})}"
    file:
      kind: rolling_file
      path: "log/home_delivery.txt"
      encoder:
        pattern: "{h({d(%Y-%m-%dT%H:%M:%SZ)(utc)} - {l}: {m}{n})}"
      policy:
        trigger:
          kind: size
          limit: 50kb
        roller:
          kind: delete
  root:
    level: info
    appenders:
      - screen
      - file
```



# Chapter 4

## main.rs

In its broad strokes, *main.rs* looks like:

```
9a  <main.rs 9a>≡  
    <main: imports 9b>  
    <main: argument parsing 10>  
    <function: get_current_deliverables 11>  
    <function: main 12a>
```

We will address each in turn.

### 4.1 Imports

We use a number of third party packages, all listed in *Cargo.toml*. In our *main.rs* file we begin by declaring a utility module we'll be using as well as the third-party code we're importing.

```
9b  <main: imports 9b>≡  
    mod utility;  
  
    use chrono::{DateTime, Utc};  
    use clap::Parser;  
    use log::info;  
    use log4rs;  
    use std::path::Path;  
    use std::process::exit;  
    use utility::{deliver, filenames_with_timestamps,  
                sanity_check, sleep_to_top_of_minute};
```

## 4.2 Argument parsing

Argument parsing is provided by the [Clap 4.5](#) crate, using the new Derive API.

```
10 <main: argument parsing 10>≡
    #[derive(Parser, Debug)]
    #[command(version)]
    #[command(about =
        "Delivers files from one directory to another on a schedule.")]
    #[command(author =
        "Rob Hansen <rob@hansen.engineering>")]
    #[command(before_help =
        "
                ** NOTE: ALL TIMES ARE IN UTC **\n
    ")]
    #[command(after_help =
        "
                ** NOTE: ALL TIMES ARE IN UTC **\n
    ")]
    #[command(help_template(
        "\n
{before-help}{name} {version}
Copyright (c) 2025 by {author}
Homepage: https://github.com/rjhansen/home_delivery

This is open source software distributed under terms of the Apache
Software Foundation's Apache-2.0 license. For the full text of the
license, please see https://www.apache.org/licenses/LICENSE-2.0.html

{about-with-newline}
{usage-heading} {usage}

{all-args}{after-help}
"
    ))]
    struct Args {
        #[arg(short, long,
            long_help = "Path to source directory")]
        source: String,

        #[arg(short, long,
            long_help = "Path to destination directory")]
        destination: String,

        #[arg(short, long,
            long_help = "Path to logging configuration file")]
```

```

    config: String,
}

```

### 4.3 Function: get\_current\_deliverables

Table 4.1:

<b>Input</b>	A borrowed reference to an immutable vector of two-element tuples, where each tuple consists of a Unicode string representing a complete path and an RFC3339-formatted timestamp representing when it should be delivered.
<b>Output</b>	A vector of Unicode strings where each string represents a complete path to a file which should be delivered now.
<b>Side effects</b>	None
<b>Called by</b>	main() in main.rs.
<b>Memory-safe?</b>	Yes.
<b>Error handling?</b>	None.
<b>Bugs</b>	None known.

```

11 <function: get_current_deliverables 11>≡
    fn get_current_deliverables(deliv: &Vec<(String, DateTime<Utc>>>)
        -> Vec<String> {
        deliv.clone()
            .into_iter()
            .filter(|(_, date)| date < &Utc::now())
            .map(|(path, _)| path)
            .collect::<Vec<String>>()
    }

```



## 4.4 Function: main

The main function is fairly straightforward, although it makes heavy use of utility functions defined elsewhere.

Table 4.2:

<b>Input</b>	Command-line flags.
<b>Output</b>	Logging messages (if configured).
<b>Side effects</b>	None
<b>Called by</b>	Program execution.
<b>Memory-safe?</b>	Yes.
<b>Error handling?</b>	The program terminates with an error code if valid parameters aren't specified for SRC, DST, and the logger configuration file.
<b>Bugs</b>	None known.

```

12a  <function: main 12a>≡
      fn main() {
          <main: housekeeping 12b>
          <main: sanity checking 13a>
          <main: initialize logging 13b>
          <main: loop 13c>
      }

```

### 4.4.1 Housekeeping

Following standard Rust practice, we open our function by declaring a handful of bindings, all of them immutable.

```

12b  <main: housekeeping 12b>≡
      let args = Args::parse();
      let source = Path::new(&args.source);
      let destination = Path::new(&args.destination);
      let config = &args.config;
      let source_str: &str;

```

### 4.4.2 Sanity checking

```
13a <main: sanity checking 13a>≡
    sanity_check(source, destination, config);
    source_str = source.to_str().unwrap_or_else(|| {
        eprintln!("Error: source path is not a valid UTF-8 string");
        exit(1);
    });
```

### 4.4.3 Initializing logging

```
13b <main: initialize logging 13b>≡
    log4rs::init_file(config, Default::default())
        .expect("couldn't init logger!");
    info!("home_delivery is starting");
```

### 4.4.4 Main loop

```
13c <main: loop 13c>≡
    loop {
        info!("polling {:?}", source_str);
        let all_deliverables = filenames_with_timestamps(source);
        let deliver_now = get_current_deliverables(&all_deliverables);
        if deliver_now.is_empty() {
            info!("no files left | exiting normally");
            exit(0);
        }
        info!("[{}] file(s) ready for delivery",
            deliver_now.len());
        deliver(&deliver_now, destination);
        sleep_to_top_of_minute();
    }
```



# Chapter 5

## utility.rs

This file is a sort of junk drawer for various functions useful in achieving desired application functionality. Its structure is fairly conventional:

```
15a  <utility.rs 15a>≡  
    <utility: imports 15b>  
    <utility: static initialization 16>  
    <function: sanity_check 17>  
    <function: sleep_to_top_of_minute 18>  
    <function: read_directory 19>  
    <function: capture_to_rfc3339 22>  
    <function: find_matching_files 20a>  
    <function: filenames_with_timestamps 24>  
    <function: deliver 25>
```

Each is covered in its own section.

### 5.1 Imports

This is essentially boilerplate code: there's nothing of particular interest, nor deserving of particular explanation, here.

```
15b  <utility: imports 15b>≡  
    use chrono::{DateTime, Duration, Utc};  
    use lazy_static::lazy_static;  
    use log::{error, info, warn};  
    use regex::Regex;  
    use std::fs::{DirEntry, ReadDir};  
    use std::path::Path;  
    use std::process::exit;  
    use std::thread::sleep;
```

## 5.2 Static initialization

Rather than recompile our filename recognition regex unnecessarily every time we run the file harvesting function, we statically initialize it here. Please note it has no visibility outside this file.

Further, note the use of `.unwrap()`. Although this is normally a bad code smell, here it's justified by the fact the regex is well-debugged, not changing, and not subject to the whims of the user. This `.unwrap()` is about as safe as we can make it.

```
16  <utility: static initialization 16>≡
    static REXP: &str =
        r"^.*(((?<day>\d+)[dD])?(?<hour>\d\d)(?<minute>\d\d))(.*)$";
    lazy_static! {
        static ref RE: Regex = Regex::new(REXP).unwrap();
    }
```

## 5.3 Function: sanity\_check

Table 5.1:

<b>Input</b>	source, a borrowed reference to a Path; destination, a borrowed reference to a Path; config, a borrowed reference to a String.
<b>Output</b>	( ), or terminates execution.
<b>Side effects</b>	None
<b>Called by</b>	main() in main.rs.
<b>Memory-safe?</b>	Yes.
<b>Error handling?</b>	The program terminates with an error code if valid parameters aren't specified for SRC, DST, and the logger configuration file.
<b>Bugs</b>	None known.

This function ensures the user-specified command line parameters meet at least basic sanity checks.

```

17 <function: sanity_check 17>≡
    pub fn sanity_check(source: &Path, destination: &Path,
        config: &String) {
        let source_ok = source.exists() && source.is_dir();
        let destination_ok = destination.exists() &&
            destination.is_dir();
        let config_ok = {
            let configpath = Path::new(config);
            configpath.exists() && configpath.is_file()
        };

        if !(source_ok && destination_ok && config_ok) {
            error!("Error: invalid parameters. Try '--help' for help.");
            exit(1);
        }
    }

```

## 5.4 Function: sleep\_to\_top\_of\_minute

Table 5.2:

<b>Input</b>	None.
<b>Output</b>	().
<b>Side effects</b>	Pauses execution between 1-60 seconds.
<b>Called by</b>	main() in main.rs.
<b>Memory-safe?</b>	Yes.
<b>Error handling?</b>	Recovers sensibly from internal errors.
<b>Bugs</b>	None known.

This function is a little more complicated than one would like. We take the current time as and advance it one minute (potentially adding us into a new day, month, or year), then convert that into an RFC3339-formatted string with the seconds field zeroed out. Converting that back into a DateTime lets us compute the difference between the two timestamps, representing how many seconds to sleep to get to the top of the next minute.

By all sense, there shouldn't be any way for there to be format conversion errors. In the strange occasion there is one anyway, we trap the error and simply sleep for one minute in order to continue execution in a somewhat degraded statement (no longer guaranteed to be pausing until the top of the minute).

```
18 <function: sleep_to_top_of_minute 18>≡
    pub fn sleep_to_top_of_minute() {
        let now = Utc::now();
        let next_minute = now + Duration::minutes(1);
        let top_of = next_minute.format("%Y-%m-%dT%H:%M:00Z").to_string();
        if let Ok(future) = DateTime::parse_from_rfc3339(top_of.as_str()) {
            if let Ok(as_std) = (future.to_utc() - now).to_std() {
                sleep(as_std);
            } else {
                sleep(Duration::minutes(1).to_std().unwrap());
            }
        } else {
            sleep(Duration::minutes(1).to_std().unwrap());
        }
    }
}
```

## 5.5 Function: read\_directory

Table 5.3:

<b>Input</b>	A borrowed reference to a Path.
<b>Output</b>	A set of directory entries, or program termination.
<b>Side effects</b>	May terminate program execution.
<b>Called by</b>	files_with_timestamps() in utility.rs.
<b>Memory-safe?</b>	Yes.
<b>Error handling?</b>	Terminates execution on error.
<b>Bugs</b>	None known.

```

19 <function: read_directory 19>≡
    fn read_directory(path: &Path) -> ReadDir {
        match path.read_dir() {
            Ok(dir) => dir,
            Err(e) => {
                error!("Error reading source directory: {}", e);
                exit(1);
            }
        }
    }

```

## 5.6 Function: find\_matching\_files

Table 5.4:

<b>Input</b>	Takes ownership of a ReadDir.
<b>Output</b>	A vector of Strings representing filenames in the input directory which match our statically-initialized regex.
<b>Side effects</b>	None.
<b>Called by</b>	main() in main.rs.
<b>Memory-safe?</b>	Yes.

Continued on next page



Table 5.4: (Continued)

<b>Input</b>	Takes ownership of a ReadDir.
<b>Error handling?</b>	Recovers gracefully and logs.
<b>Bugs</b>	None known.

We begin by initializing our return value, and filtering from our input set those files which have valid entries. Those valid entries are ultimately collected into a vector of Strings representing complete path names and are returned.

```

20a <function: find_matching_files 20a>≡
    fn find_matching_files(contents: ReadDir) -> Vec<String> {
        let mut matching_filenames = Vec::<String>::new();
        <compile entries 20b>
        for dir_entry in entries {
            if let Some(filename) = dir_entry.file_name().to_str() {
                if let Some(_) = RE.captures(filename) {
                    if let Some(good_name) = dir_entry.path().to_str() {
                        matching_filenames.push(good_name.to_string());
                        <fmf: error reporting 21>
                    }
                }
            }
        }
        matching_filenames
    }

```

### 5.6.1 Compiling entries

Compiling the entries is a fairly straightforward pipeline of filter and map operations.

```

20b <compile entries 20b>≡
    let entries = contents
        .filter_map(Result::ok)
        .filter(|entry| entry.path().is_file())
        .collect::<Vec<DirEntry>>();

```

### 5.6.2 Error reporting

Error reporting is handled through the logging facility. In no case will errors be propagated to the caller, nor result in termination of the function or program: we just stand up, dust ourselves off, and continue.

```
21 <fmf: error reporting 21>≡
    } else {
        info!("invalid character in file path: skipping");
    }
    } else {
        info!("file '{}' is not a deliverable", filename);
    }
    } else {
        warn!("invalid character in file name: skipping");
    }
}
```

## 5.7 Function: capture\_to\_rfc3339

Table 5.5:

<b>Input</b>	Borrows a reference to a <code>regex::Captures</code> .
<b>Output</b>	An <code>Option&lt;DateTime&lt;Utc&gt;&gt;</code> representing the UTC time described by the captured timestamp.
<b>Side effects</b>	None.
<b>Called by</b>	<code>filenames_with_timestamps()</code> in <code>utility.rs</code> .
<b>Memory-safe?</b>	Yes.
<b>Error handling?</b>	Recovers gracefully and continues execution.
<b>Bugs</b>	None known.

```

22  <function: capture_to_rfc3339 22>≡
    fn capture_to_rfc3339(capture: &regex::Captures) -> Option<DateTime<Utc>> {
        <ctf: binding initialization 23a>
        <ctf: sanity check 23b>

        let days = day.parse::<i64>().unwrap_or(0);
        let duration = Duration::seconds(86400 * days);
        let date_string = (Utc::now() + duration).format("%Y-%m-%d").to_string();
        let rfc3339_string = format!("{}", T{}:{}.to_string(), date_string, hour, minute);
        Some(
            DateTime::parse_from_rfc3339(&rfc3339_string.as_str())
                .unwrap()
                .to_utc(),
        )
    }

```

### 5.7.1 Binding initialization

As is par for the course with Rust, initializing bindings is rather verbose.

```
23a <ctf: binding initialization 23a>≡
    let day: &str;
    let hour: &str;
    let minute: &str;

    if let Some(d) = capture.name("day") {
        day = d.as_str();
    } else {
        day = "0";
    };

    if let Some(h) = capture.name("hour") {
        hour = h.as_str();
    } else {
        hour = "00";
    };

    if let Some(m) = capture.name("minute") {
        minute = m.as_str();
    } else {
        minute = "00";
    }
}
```

### 5.7.2 Sanity check

```
23b <ctf: sanity check 23b>≡

// sanity-check: a file with an invalid timestamp won't be picked up
if hour.parse::<u32>().unwrap() >= 24 ||
    minute.parse::<u32>().unwrap() >= 60 {
    return None;
}
```

## 5.8 Function: filenames\_with\_timestamps

Table 5.6:

<b>Input</b>	Borrows a reference to a Path.
<b>Output</b>	A <code>Vec&lt;(String, DateTime&lt;Utc&gt;)&gt;</code> representing the name of each file matching our statically assigned regex, and the UTC time described by the captured timestamp.
<b>Side effects</b>	None.
<b>Called by</b>	<code>main()</code> in <code>main.rs</code> .
<b>Memory-safe?</b>	Yes.
<b>Error handling?</b>	Recovers gracefully and continues execution.
<b>Bugs</b>	None known.

```

24  <function: filenames_with_timestamps 24>≡
    pub fn filenames_with_timestamps(src: &Path)
        -> Vec<(String, DateTime<Utc>)> {
        let mut rv: Vec<(String, DateTime<Utc>)> = vec![];

        for path in find_matching_files(read_directory(src)) {
            if let Some(capture) = RE.captures(&path) {
                if let Some(rfc3339) = capture_to_rfc3339(&capture) {
                    rv.push((path, rfc3339));
                }
            }
        }
        rv.sort_by(|a, b| a.1.cmp(&b.1));
        rv
    }

```

## 5.9 Function: deliver

Table 5.7:

<b>Input</b>	Borrows a reference to a <code>Vec&lt;String&gt;</code> denoting the file paths to be moved.
<b>Output</b>	<code>()</code>
<b>Side effects</b>	Reorganizes files on disk.
<b>Called by</b>	<code>main()</code> in <code>main.rs</code> .
<b>Memory-safe?</b>	Yes.
<b>Error handling?</b>	Recovers gracefully and continues execution.
<b>Bugs</b>	None known.

Due to a bug in Rust, moving files between different Windows drives with `std::fs::move` is not supported. Hence, on all platforms we do a copy followed by removing the original.

```

25 <function: deliver 25>≡
    pub fn deliver(filenamees: &Vec<String>, destination: &Path) {
        for path in filenamees {
            if let Some(name_1) = Path::new(&path).file_name() {
                if let Some(name_2) = name_1.to_str() {
                    if let Some(dest) = destination.join(name_2).to_str() {
                        info!("moving {} to {}", &path, dest);
                        if let Err(e) = std::fs::copy(&path, &dest) {
                            error!("Error copying file {}: {}", &path, e);
                        }
                        if let Err(e) = std::fs::remove_file(&path) {
                            error!("Error moving file {}: {}", &path, e);
                        }
                    }
                }
            }
        }
    }
}

```