Software Requirements Specification

for

File System Watcher

Version 1.0 approved

Prepared by Mansur Yassin and Tairan Zhang

University of Washington Tacoma

April 21st, 2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

<Identify the product whose software requirements are specified in this document, including the revision or release number. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem.>

1.2 Document Conventions

<Describe any standards or typographical conventions that were followed when writing this SRS, such as fonts or highlighting that have special significance. For example, state whether priorities for higher-level requirements are assumed to be inherited by detailed requirements, or whether every requirement statement is to have its own priority.>

1.3 Intended Audience and Reading Suggestions

<Describe the different types of reader that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers. Describe what the rest of this SRS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections and proceeding through the sections that are most pertinent to each reader type.>

1.4 Project Scope

<Provide a short description of the software being specified and its purpose, including relevant benefits, objectives, and goals. Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here. An SRS that specifies the next release of an evolving product should contain its own scope statement as a subset of the long-term strategic product vision.>

1.5 References

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

2. Overall Description

2.1 Product Perspective

<Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.>

2.2 Product Features

<Summarize the major features the product contains or the significant functions that it performs or lets the user perform. Details will be provided in Section 3, so only a high-level summary is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top-level data flow diagram or a class diagram, is often effective.>

2.3 User Classes and Characteristics

<Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience. Describe the pertinent characteristics of each user class. Certain requirements may pertain only to certain user classes. Distinguish the favored user classes from those who are less important to satisfy.>

2.4 Operating Environment

<Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.>

2.5 Design and Implementation Constraints

<Describe any items or issues that will limit the options available to the developers. These might include: corporate or regulatory policies; hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer's organization will be responsible for maintaining the delivered software).>

2.6 User Documentation

<List the user documentation components (such as user manuals, on-line help, and tutorials) that will be delivered along with the software. Identify any known user documentation delivery formats or standards.>

2.7 Assumptions and Dependencies

<List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project, unless they are already documented elsewhere (for example, in the vision and scope document or the project plan).>

3. System Features

<This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.>

3.1 System Feature 1

<Don't really say "System Feature 1." State the feature name in just a few words.>

3.1.1 Description and Priority

<Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).>

3.1.2 Stimulus/Response Sequences

<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>

3.1.3 Functional Requirements

<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use "TBD" as a placeholder to indicate, when necessary, information is not yet available.>

<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>

REQ-1:
REQ-2:

3.2 System Feature 2 (and so on)

4. External Interface Requirements

4.1 User Interfaces

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>

4.2 Hardware Interfaces

<Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.>

4.3 Software Interfaces

<Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.>

4.4 Communications Interfaces

<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.>

5. Other Nonfunctional Requirements

5.1 Performance Requirements

The File Watcher system is expected to respond to file events almost instantaneously. Events such as creation, deletion, renaming, or modification of files should be detected and logged within a 2-second window from the moment the system receives the notification. The application must maintain a responsive user interface at all times, even when processing high-frequency file system events. The software should also be capable of handling bursts of up to 100 file events per second without freezing or failing. All databases write operations (such as saving logged events) should be completed within 1 second of initiation. When querying the database, results for standard filters (e.g., by file extension or date) must return within 3 seconds for datasets up to 10,000 records. The software must remain operational for extended monitoring sessions (at least 8 hours continuously) without memory leaks or degradation in performance.

5.2 Safety Requirements

The File Watcher software does not perform any write or delete operations on user files; it operates purely in a read-only monitoring mode. However, there are safety concerns associated with potentially unexpected crashes or misinterpreted file paths. Therefore, the application must include safeguards such as validating all input paths before monitoring and catching all unhandled exceptions to prevent crashes. If the database has not yet been written at the time of application closure, the user must be prompted with a dialog asking whether to save the currently logged information. To prevent user errors, the "Clear Database" feature must include a confirmation dialog. The system will never attempt to open, modify, or interact with any files beyond registering their events. External dependencies like SQLite.NET must be verified secure and kept up to date to prevent vulnerabilities related to unsafe data access.

5.3 Security Requirements

While the application is strictly local and does not interface with any network components, it still requires attention to file and data security. The SQLite database file, which logs monitored events, must be saved in a location that honors system-level user access controls, such as the user's Documents or AppData directory. The application must never transmit data externally. Only the user who initiated monitoring and wrote the data to the database should have access to that database file by default. The application should not run with administrative privileges unless explicitly required. Since this software does not include user authentication, its access model is entirely dependent on the operating system's user session. The application must prevent command injection, script execution, or dynamic loading of files beyond its scope. Optional read-only database access may be provided in a future release to support audit use cases.

5.4 Software Quality Attributes

The File Watcher software is designed with reliability and usability as top priorities. It must operate consistently across supported environments, including Windows 10+ and macOS Monterey and later using .NET 6+. The user interface must clearly indicate system status (monitoring/not monitoring) and visually disable unavailable controls to guide user behavior and avoid mis-operations. Error messages must be user-friendly and specific. From a maintainability *perspective, the application code must be structured using a modular design, separating concerns between the UI, file monitoring logic, and database handling. This separation allows for future enhancements (e.g., CSV export, cloud sync) to be added without refactoring the entire system. Testability is built-in via the use of logging to both UI and database, allowing easy verification of output correctness. Portability is supported on macOS via frameworks such as .NET MAUI or Avalonia UI. Overall, this software favors robustness and reliability over feature complexity, in line with its primary purpose: dependable file system change tracking.*

6. Other Requirements

The File Watcher system must support extensive logging to a lightweight, embedded SQLite database. Each entry in the database must include: the name of the file, the absolute path, the type of event (create, delete, modify, rename), and a timestamp in ISO 8601 format. While the software does not need to support internationalization for its interface in the current release, it must properly handle and display file paths and filenames containing UTF-8 characters, such as those in non-English languages (e.g., Japanese, Russian, Arabic). All time data must be consistent across time zones and saved using a 24-hour format. The application must include a Help to About menu that lists the program's usage instructions, the current version, and the developer's name. There are currently no legal or regulatory constraints on the software; however, the software must respect user permissions and never monitor system directories without explicit permission. The reuse objective is met through modular design: components such as the 'FileMonitorController' and 'DatabaseHandler' are implemented as reusable classes that could be embedded into other desktop monitoring utilities or server-side file logging tools. The only required third-party library is SQLite.NET, which must be bundled with the application in a redistributable format for both macOS and Windows environments.

Appendix A: Glossary

This glossary defines terms and concepts essential for understanding the File Watcher system. These definitions support clarity across stakeholders, especially for those without prior experience working with file system monitors or embedded databases. The ***File Watcher*** *refers to the name of the desktop application developed as part of this project. It allows users to observe and log changes to the file system in real time, targeting specific file types based on user-defined extensions. The application uses the* ***FileSystemWatcher*** *class in .NET (or equivalent wrappers in cross-platform frameworks) to detect file events, including creation, deletion, renaming, and modification. These events are logged and, optionally, saved to a local* ***SQLite*** *database for later querying. An* ***event type*** *describes the nature of a file system change. For this system, the core event types are: Created, Deleted, Changed, and Renamed. A* ***monitoring path*** *refers to the folder selected by the user where these file system events will be watched. The monitoring scope can be either shallow (non-recursive) or deep (including all subdirectories). A* ***file extension*** *is the suffix at the end of a file name indicating its format or type (e.g., .txt, .md, .mov). Users may select from a preset list or define custom extensions to watch. The* ***database file*** *is a local .db file in SQLite format that stores event logs with fields for file name, path, event type, and timestamp. The* ***query window*** *is a GUI component that provides filters and search functionality, allowing users to find specific events from the database history. Supporting this are GUI elements like a* ***menu strip*** *(housing dropdowns for File, Help, and About) and a* ***tool strip*** *(with clickable buttons for Start, Stop, Save, and Query actions).While initially developed with Windows and .NET in mind, the File Watcher system also supports* ***macOS*** *through .NET 6+ compatibility and file system APIs* *available in macOS Monterey and later. For Mac users, the application may be run natively via Visual Studio for Mac or built using cross-platform frameworks like Avalonia UI or .NET MAUI, ensuring UI and file system access behaves consistently across platforms. Lastly, terms like **TBD** ("To Be Determined") and **read-only mode** refer to ongoing development elements and optional features that may be implemented in future releases (e.g., enabling users to view the database without making modifications).*

Appendix B: Analysis Models

*The File Watcher system follows a modular design that separates core responsibilities into three logical components: user interface, file monitoring, and data persistence. These are implemented via MainForm, FileMonitorController, and DatabaseHandler classes; This separation allows for more maintainable code and easier unit testing, particularly when adapting the application for different operating systems like macOS. From a use case perspective, the application supports the following core user interactions: starting the monitoring of a selected folder and file extension, stopping that monitoring process, saving the current event log to a database, querying past events from the database, clearing the database with confirmation, and exiting the application safely. Each of these actions transitions the system between defined internal states, typically from an idle state to an active one and then back. For instance, the system transitions from **Idle to Monitoring** when the user clicks "Start Monitoring," and from **Monitoring to Idle** when the user clicks "Stop." If the user chooses to save data, the state moves temporarily to **Writing to Database** and then returns to idle. During querying, the state becomes **Query Mode**, where the main interface is disabled until the user returns. These transitions are clearly controlled and reflected in the UI, so that the system cannot enter invalid or overlapping states. Diagrams such as class diagrams and state-transition models are currently under development. The class diagram will illustrate the relationship between the core logic classes and event-handling mechanisms, while the state-transition diagram will visualize the application's behavior during various user actions. Both models will be finalized and included prior to project submission.*

Appendix C: Issues List

*Several implementation and design considerations remain open as the File Watcher system approaches completion. These items reflect features under development, planned enhancements, or unresolved decisions that may affect the final scope of the project. One of the current development goals is to implement a **CSV export feature** in the Query Window which allows users to export filtered results for use in spreadsheet tools or reporting. The internal logic is complete, but the file dialog integration for both macOS and Windows is undergoing testing. Similarly, the **iconography and visual styling** of the toolbar is being revised to provide platform-consistent visuals; macOS users will see native-style icons and menu alignment appropriate for Aqua/HIG standards. Another ongoing task is the improvement of **query filtering logic**. At present, the user can search based on file extension and timestamp. Additional filters for event type and full-text file path search are planned for inclusion in the final release. Moreover, handling edge cases such as **rename operations across volumes** (i.e., between different drives or partitions) has proven complex due to differing file system behaviors on macOS versus Windows. These cases are under evaluation and unit tests are being developed to handle them. The **Help and About menu** functionality are complete, but the optional inclusion of a downloadable PDF manual is still pending a decision on packaging and distribution. Finally, while multi-directory support and concurrent monitoring sessions have been requested, they are deferred to a potential v2.0 release due to increased architectural complexity. This issues list will remain dynamic and will be updated* *continuously until the final deployment and submission milestone. It serves both as a project management tool and a roadmap for future improvements beyond the current academic scope.*