



# **File System Driver Documentation**

**Project Name:** File System Driver

**Project supervisor:** Maxim Goncharenko

**Authors:**

- Adrian Jablonski
- Jakub Oleksy
- Mateusz Zukowski

**Date of presentation:** 30.09.2024

<b>1. Introduction</b>	<b>3</b>
1.1. Project Overview	3
1.2. Purpose	3
1.3. Supported Platforms	3
1.4. Console Utility Guide	3
<b>2. Project Requirements</b>	<b>5</b>
2.1. Functional Requirements	5
2.2. Non-Functional Requirements	5
2.3. Console Utility	5
2.4. Levels of Complexity	5
<b>3. Technologies Used</b>	<b>6</b>
<b>4. Architecture and Components</b>	<b>6</b>
4.1. System Architecture	7
4.1.1. Driver Initialization	7
4.1.2. File System Operations	7
4.1.3. Virtual Disk Management	7
4.1.4. Kernel and Hardware Interaction	7
4.2. Sequence of Operations	7
4.2.1. File Creation	8
4.2.2. File Read and Write	8
<b>5. Development Tasks and Timeline</b>	<b>8</b>
<b>6. Testing and Validation</b>	<b>9</b>
6.1. Test Environment	9
6.2. Functional Testing	9
6.3. Integration Testing	9
<b>7. Results and Conclusions</b>	<b>9</b>
7.1. Lessons Learned	9
7.2. Challenges	10

# 1. Introduction

In this part we will focus on the general overview, purpose of the project and a quick guide on how to use the utility.

## 1.1. Project Overview

The project is focused on developing a custom kernel mode file system driver for Windows that allows a user to mount and manage a virtual disk. The virtual disk is accessed through Windows applications like File Explorer, Notepad, and a console utility named Ultimate wfsDriver Utility (UwU). This utility provides various commands to handle the virtual disk, perform file operations, and manage the driver.

## 1.2. Purpose

The driver enables mounting of a virtual disk that can be manipulated using various file operations like creating, reading, writing, deleting, and querying file information. This project showcases how a custom file system driver is integrated with a user-mode console utility and Windows applications.

## 1.3. Supported Platforms

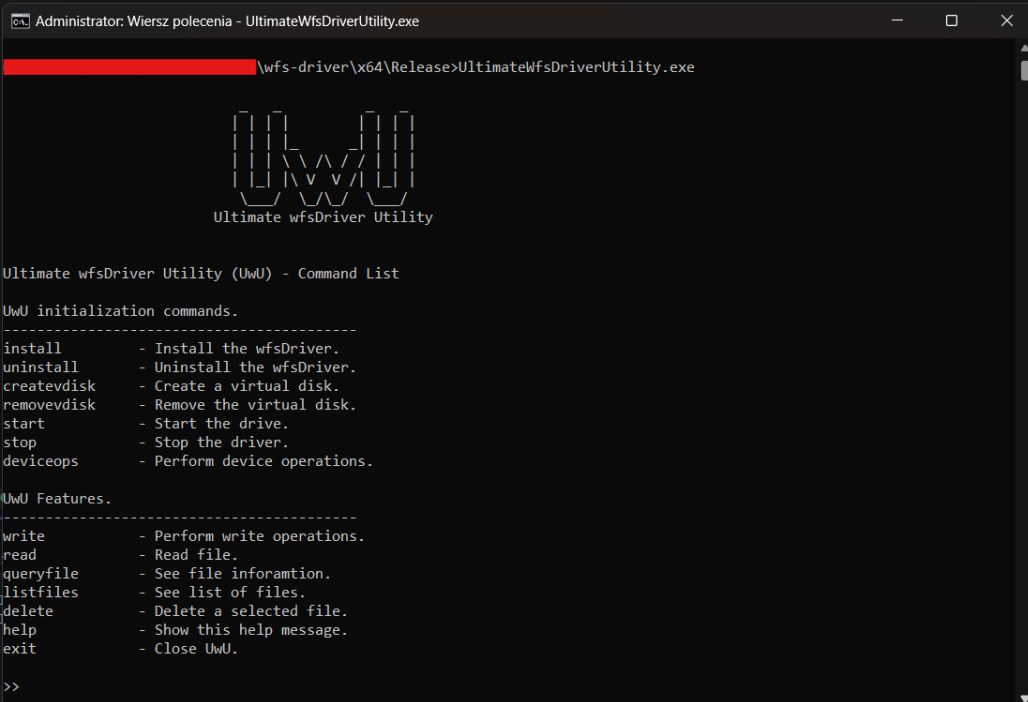
- Windows 10 x64

## 1.4. Console Utility Guide

To utilize the driver's utilities, such as **read**, **write**, **query**, and **list**, follow these steps:

1. After downloading the project, locate the specific directory at: "PathToProject\x64\Release" and copy this path. If you don't have an x64 folder simply open wfsDriver.sln and make sure build options are set to x64 and Release. Then build the project. The folders should be created and you can continue.
2. Open a terminal with administrator privileges.
3. In the terminal, navigate to the directory by typing:  
`cd PathToProject\x64\Release` and pressing Enter.

4. Once in the correct directory, you can run the executable file by entering: `UltimateWfsDriverUtility.exe`.
5. A list of available commands will be displayed, as shown in the image.



```
Administrator: Wiersz polecenia - UltimateWfsDriverUtility.exe
\\wfs-driver\x64\Release>UltimateWfsDriverUtility.exe

      _____
     /  _  _  \  \
    /  /  \  \  \  \
   /  /    \  \  \  \
  /  /      \  \  \  \
 /  /        \  \  \  \
/  /          \  \  \  \
 \  \          /  /  /  /
  \  \        /  /  /  /
   \  \      /  /  /  /
    \  \    /  /  /  /
     \  _  /  \  \
      \___/

Ultimate wfsDriver Utility

Ultimate wfsDriver Utility (UwU) - Command List

UwU initialization commands.
-----
install      - Install the wfsDriver.
uninstall    - Uninstall the wfsDriver.
createvdisk  - Create a virtual disk.
removevdisk  - Remove the virtual disk.
start        - Start the drive.
stop         - Stop the driver.
deviceops    - Perform device operations.

UwU Features.
-----
write        - Perform write operations.
read         - Read file.
queryfile    - See file information.
listfiles    - See list of files.
delete       - Delete a selected file.
help         - Show this help message.
exit         - Close UwU.

>>
```

6. It is important to use UwU initialization commands before using features because features will not work.
7. Use “install” to install the driver, next “createvdisk” to create and mount virtual disk, lastly start the driver by typing “start”.
8. You are good to go! Now you can use all the features. It goes without saying that firstly you should create a file in order to write to it or read it.
9. If you’d like to see commands once again type “help”. Before closing you can stop the driver with “stop” and remove and unmount the virtual disk with “removevdisk”. Type “exit” to leave `UltimateWfsDriverUtility.exe`.

## 2. Project Requirements

### 2.1. Functional Requirements

The file system driver must support the following functionalities:

- Create, open, and close files.
- Read and write file contents.
- Query and display file attributes.
- List files and directories.
- Mount and unmount the virtual disk via the console utility.
- Integration with File Explorer and Notepad for file manipulation.

### 2.2. Non-Functional Requirements

- Compatibility with Windows 10 x64.
- The driver must pass Windows Driver Verifier checks to ensure stability and compliance.

### 2.3. Console Utility

The console utility will support the following commands:

- Install/uninstall the driver.
- Mount/unmount the virtual disk.

### 2.4. Levels of Complexity

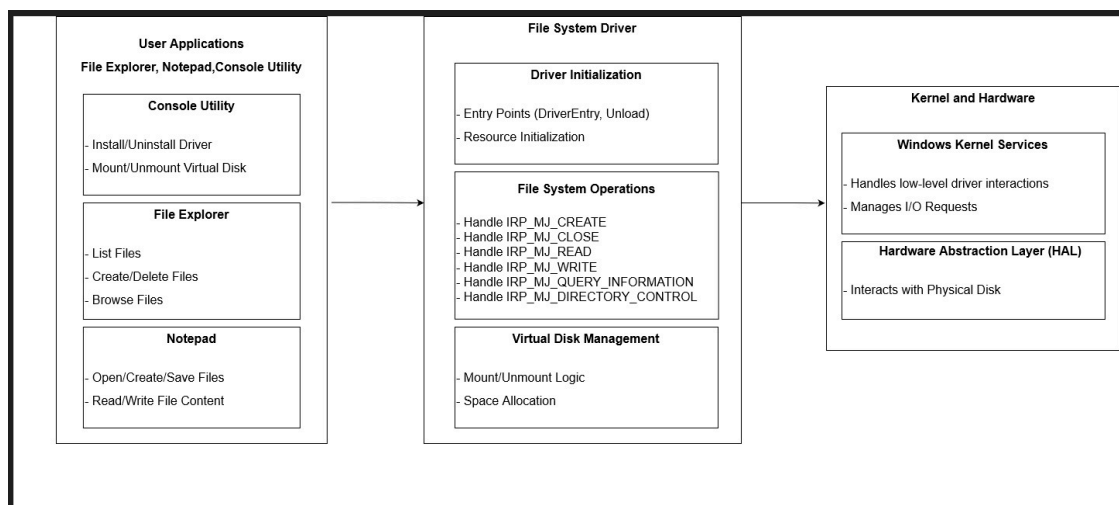
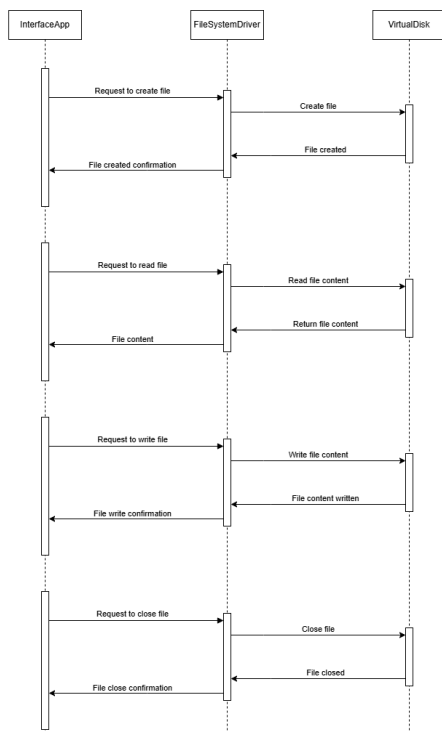
The implementation will consist of three levels of complexity:

1. **Minimal:** The disk is directly accessible by letter (or symbolic link) from the command line. Basic file operations (create, close, and attribute querying) will be supported.
2. **Extended:** Adds reading and writing functionality to the mounted disk.
3. **Full Feature Set:** Complete integration with File Explorer and Notepad. Users will be able to create, open, modify, and save files using the applications.

### 3. Technologies Used

- Development Tools: Visual Studio, Windows Driver Kit (WDK), C++, Windows API
- File System Driver Framework: NT Kernel APIs
- User Mode Application: Console Utility using Windows API
- Git: Source control
- Debugging: DebugView for real-time driver logs

### 4. Architecture and Components



## 4.1. System Architecture

The system consists of the following key components:

- **File System Driver:** Handles file operations and virtual disk management.
- **Console Utility:** Provides commands to install/uninstall the driver and manage the virtual disk.
- **User Applications:** File Explorer and Notepad are used to interact with the mounted disk.

### 4.1.1. Driver Initialization

- **Entry Points:** `DriverEntry`, `Unload`
- **Resource Initialization:** Initialize resources required by the file system.

### 4.1.2. File System Operations

- **IRP\_MJ\_CREATE:** Handles file creation and opening.
- **IRP\_MJ\_CLOSE:** Manages closing of files.
- **IRP\_MJ\_READ:** Reads data from a file.
- **IRP\_MJ\_WRITE:** Writes data to a file.
- **IRP\_MJ\_QUERY\_INFORMATION:** Queries file attributes.
- **IRP\_MJ\_DIRECTORY\_CONTROL:** Lists directories and files.

### 4.1.3. Virtual Disk Management

- Handles mounting and unmounting logic for the virtual disk.
- Manages space allocation and deallocation for files.

### 4.1.4. Kernel and Hardware Interaction

- **Windows Kernel Services:** Handles low-level driver interactions.
- **Hardware Abstraction Layer (HAL):** Interfaces with physical disk storage.

## 4.2. Sequence of Operations

The following sequence diagrams illustrate the interaction between components:

### 4.2.1. File Creation

1. The user application (e.g., Notepad) requests to create a file.
2. The request is passed to the **FileSystemDriver**, which interacts with the virtual disk to create the file.
3. The system returns a confirmation to the user application.

### 4.2.2. File Read and Write

1. The user application requests to read/write a file.
2. The request is passed to the **FileSystemDriver**, which interacts with the virtual disk to read/write the file.
3. The driver returns the file contents or write confirmation to the user application.

## 5. Development Tasks and Timeline

The following table outlines the tasks required for development, along with the assignees and due dates:

	Task ID	Task description	Duration	Due date	Assignee
Driver Development	1.1	Driver Initialization	1d	24.08.2024	teamwork
	1.2	Implement driver entry points (DriverEntry, Unload)	1d	24.08.2024	teamwork
	1.3	Initialize driver configuration and resources	1d	24.08.2024	teamwork
	1.4	Implement IRP_MJ_CREATE for file creation and opening	3d	27.08.2024	teamwork
	1.5	Implement IRP_MJ_CLOSE for closing files	3d	30.08.2024	Mateusz Żukowski
	1.6	Implement IRP_MJ_READ for reading file data	3d	30.08.2024	Jakub Oleksy
	1.7	Implement IRP_MJ_WRITE for writing file data	3d	30.08.2024	Adrian Jabłoński
	1.8	Implement IRP_MJ_QUERY_INFORMATION to get file attributes	3d	2.09.2024	Adrian Jabłoński
	1.9	Implement IRP_MJ_DIRECTORY_CONTROL for directory operations and file listing	3d	5.09.2024	Adrian Jabłoński
	1.10	Implement disk mounting and unmounting logic	5d	4.09.2024	Mateusz Żukowski
	1.11	Implement space allocation and deallocation for files	5d	4.09.2024	Jakub Oleksy
Console Utility Development	2.1	Develop utility commands for driver installation and uninstallation	3d	7.09.2024	Mateusz Żukowski
	2.2	Implement commands for mounting and unmounting the virtual disk	3d	7.09.2024	Jakub Oleksy
Integration with User Interfaces	3.1	Ensure the file system is accessible via File Explorer	3d	8.09.2024	Adrian Jabłoński
	3.2	Test file creation, deletion, renaming, and browsing through File Explorer	2d	10.09.2024	Adrian Jabłoński
	3.3	Ensure files can be created, edited, and saved via Notepad	3d	10.09.2024	Mateusz Żukowski
	3.4	Test opening and saving files in Notepad	2d	12.09.2024	Mateusz Żukowski
Testing and Validation	4.1	Test all file system operations (create, read, write, delete)		17.09.2024	teamwork
	4.2	Verify directory operations (listing, navigation)		17.09.2024	teamwork
	4.3	Test integration with File Explorer		17.09.2024	teamwork
	4.4	Test integration with Notepad		17.09.2024	teamwork
	4.5	Run Driver Verifier to check stability and compliance	3d	20.09.2024	Adrian Jabłoński
Documentation and Support	5.1	Document driver architecture and design	1d	21.09.2024	Jakub Oleksy
Deployment	6.1	Create an installer package for the driver and console utility	2d	23.09.2024	Mateusz Żukowski



## 6. Testing and Validation

### 6.1. Test Environment

The test environment consists of:

- A Windows 10 x64 virtual machine where the driver is installed and tested.
- **DebugView**: Used to monitor real-time logging for the driver to track file operations, resource allocation, and errors.

### 6.2. Functional Testing

- **File Creation**: Verified through Notepad and File Explorer, ensuring files are created on the virtual disk and accessible for reading/writing.
- **Read/Write Operations**: Conducted using Notepad to ensure files are readable and writable with proper content persistence.
- **Console Utility Tests**: Commands such as `write`, `read`, `listfiles`, and `delete` were tested to ensure accurate interaction with the driver.

### 6.3. Integration Testing

- Verified integration with File Explorer for browsing, creating, and deleting files.
- Ensured proper synchronization between File Explorer, Notepad, and the UwU utility when handling files on the virtual disk.

## 7. Results and Conclusions

### 7.1. Lessons Learned

Developing a kernel mode driver requires careful attention to resource management and synchronization. Integrating the driver with user-mode applications like Notepad and File Explorer provided valuable insights into file system operations at both the kernel and user levels.

## 7.2. Challenges

- **Kernel-Mode Debugging:** Debugging kernel mode components required using specialized tools like **DebugView** and was challenging due to limited error feedback.
- **Synchronization Issues:** Ensuring correct synchronization between kernel and user mode, especially during concurrent I/O operations, was a key challenge.