**Michael O’Neill**

**CTViewer: A Customizable DICOM Loader for Medical Image Analysis**

**Dordt University**

**CMSC 393 – Individual Studies**

**Prof. Breems**

**August 14th, 2025**

**Abstract**

Medical imaging files, particularly in the DICOM format, are more than static pictures — they store pixel data and embedded patient information that must be processed before display. Customizable, lightweight viewing tools for such formats are limited. This independent study developed a C# Windows Presentation Foundation (WPF) application for viewing, annotating, and analyzing medical images, with a strong emphasis on per-pane customization. Key features include slice navigation, adjustable window width and center with numerical display, preset modes (bone, lung, soft tissue, brain), per-pane drawing tools with color and stroke controls, undo functions, annotation visibility toggles, and side-by-side viewing. Unique contributions include the ability to draw directly on images while preserving individual pane settings and the integration of annotation-based WW/WL value display. Results demonstrate a flexible, user-friendly system capable of functioning as both a medical image loader and an interactive note-taking platform, with potential for mobile synchronization and advanced export options.

**Introduction**

Medical imaging is a critical component of modern healthcare and biomedical research, enabling the visualization of internal body structures for diagnosis, treatment planning, and scientific investigation. Technologies such as computed tomography (CT), magnetic resonance imaging (MRI), and X-ray produce large volumes of complex image data that require specialized software for effective interpretation. The quality of visualization tools directly impacts the accuracy of clinical decision-making and the efficiency of research workflows.

The Digital Imaging and Communications in Medicine (DICOM) standard is the primary format for storing and transmitting medical images. Unlike conventional image formats, DICOM files contain both pixel data and extensive metadata — including patient identifiers, scan parameters, and modality details — which must be parsed and reconstructed before display. This makes developing a medical image viewer more technically demanding than simply rendering standard image files.

While professional-grade viewers exist, they are often commercial, closed-source, and designed for large-scale clinical environments, making them less accessible to students, researchers, or smaller organizations. Open-source alternatives can be either too limited in functionality or too complex for targeted, research-oriented use. In particular, lightweight applications that combine robust annotation tools, per-pane customization, and intuitive controls are uncommon.

This independent study project, CTViewer, aims to address that gap by developing a C# WPF desktop application for interactive DICOM image viewing, annotation, and analysis. The software will integrate features such as slice navigation, adjustable window width and center with numerical display, side-by-side viewing modes, customizable drawing tools, and flexible annotation visibility controls. By focusing on both accessibility and advanced functionality, CTViewer is designed to serve as a practical image loader and an interactive note-taking platform, providing a versatile tool for researchers, educators, and students in biomedical contexts.

**Project Goals**

The core functionality of the viewer includes loading and displaying DICOM images along with their associated metadata, as well as supporting slice navigation for multi-slice or 3D scan datasets. Annotation and interaction tools allow for freehand drawing with adjustable stroke size and color, with per-pane control over annotation visibility and undo functions. Window width (WW) and window level (WL) adjustments are implemented through sliders with numerical display, and preset options are available for specific tissues such as bone, lung, soft tissue, and brain. Image analysis features include a pixel intensity histogram, basic contrast adjustments, and support for optional filters and overlay tools. Multi-pane functionality enables side-by-side viewing with independent settings and annotations for each pane. The application also supports a non-destructive “Save As” feature to export annotated images without modifying the original file.

From a development perspective, the project aimed to reinforce C# and WPF skills, building on prior experience in Python and Java, while also practicing structured version control workflows using Git and feature branches. The primary constraint was time, particularly when troubleshooting complex interactive features. One major challenge was integrating zoom and drawing functionality. Significant effort was spent attempting to implement zoom and pan without breaking annotation alignment, but drawn strokes failed to scale correctly with zoom, leading to misaligned positions. This incompatibility required multiple rewrites and ultimately resulted in deprioritizing zoom in favor of stable annotation behavior. Another challenge involved optimizing the UI structure; the project initially used numerous XAML files for separate features, which proved inefficient. Consolidating the design into fewer XAML files, each containing multiple controls, improved workflow but required significant restructuring. Version control was adopted later than ideal, but once Git and feature branching were integrated, they greatly improved development speed by allowing for easy rollbacks and isolated feature testing—an earlier adoption could have saved time lost to trial-and-error coding. The side-by-side mode also posed complexity, requiring extensive fine-tuning to achieve smooth, independent control for each pane, which took longer than anticipated but was ultimately successful.

**System Overview**

CTViewer is a standalone C# WPF application designed for loading, displaying, annotating, and analyzing medical images in the DICOM format. The application parses pixel and metadata from DICOM files using the *fo-dicom* library, reconstructs the image, and applies histogram-based automatic window width (WW) and window level (WL) adjustments to enhance clarity. Users can interactively adjust WW/WL values, select tissue-specific presets, and annotate images with per-pane customization. CTViewer supports both single-pane and side-by-side viewing modes, enabling independent annotation and display settings for each pane. A “Save As” function allows annotated versions to be saved without modifying the original file.

A diagram of a computer

AI-generated content may be incorrect.

**Figure 1 Showing the chain of command in the program**

**Technologies, Frameworks, and Libraries**

***Language & Framework*:** C# with Windows Presentation Foundation (WPF) for desktop UI.

***DICOM Handling*:** [fo-dicom](https://github.com/fo-dicom/fo-dicom) for reading, parsing, and saving DICOM images.

***Charting/Histogram:*** OxyPlot (planned) for histogram visualization; percentile-based contrast adjustment implemented in code.

***UI Components:*** Custom WPF UserControls for FileButtons and DrawingTools panels.

***Version Control:*** GitHub repository with feature branch workflow.

***Development Tools:*** Visual Studio 2022, Git command line interface (CLI) for reliable commit pushes.

**Main Components & Workflow**

***Workflow Steps:***

1. File Input – User opens a DICOM file via the FileButtons panel.
2. DICOM Parsing – The *fo-dicom* library extracts pixel data and metadata.
3. Image Reconstruction – Pixel intensity values are scaled using a histogram-based auto WW/WL algorithm (1st–99th percentile) to improve contrast.
4. Rendering – Images are displayed in either single-pane or two-pane mode via WPF *Viewbox* controls.
5. Interaction & Annotation –
   * Adjustable WW/WL sliders and presets
   * Drawing tools with color and stroke size control
   * Toggle visibility for annotations and patient info
   * Undo and clear functions per pane
6. Save As – Annotated images can be saved as new DICOM files without overwriting originals.

**Implementation**

The application’s workflow begins with DICOM file loading and parsing. Using the fo-dicom library, the program opens the selected file, reads the dataset, and extracts both the pixel frames and key metadata such as patient details, study information, modality, and rescale slope/intercept values. The pixel data from the first frame is copied into a ushort[] buffer to enable high-precision 16-bit processing.

The image reconstruction pipeline converts these raw pixels into a displayable 16-bit grayscale bitmap using a window level/window width (WL/WW) mapping. On initial load, an automatic window is computed per pane, providing an optimal starting contrast. The user can then interactively adjust WL/WW values through sliders, triggering a re-render of the image in real time.

The two-pane rendering surface is designed so that each pane contains a fixed-size image grid wrapped by a WPF Viewbox set to uniform scaling. Inside each pane, the image bitmap is layered with an InkCanvas for annotations and additional overlays for metadata. This structure ensures that the image and drawings scale together, maintaining proper alignment between strokes and pixels.

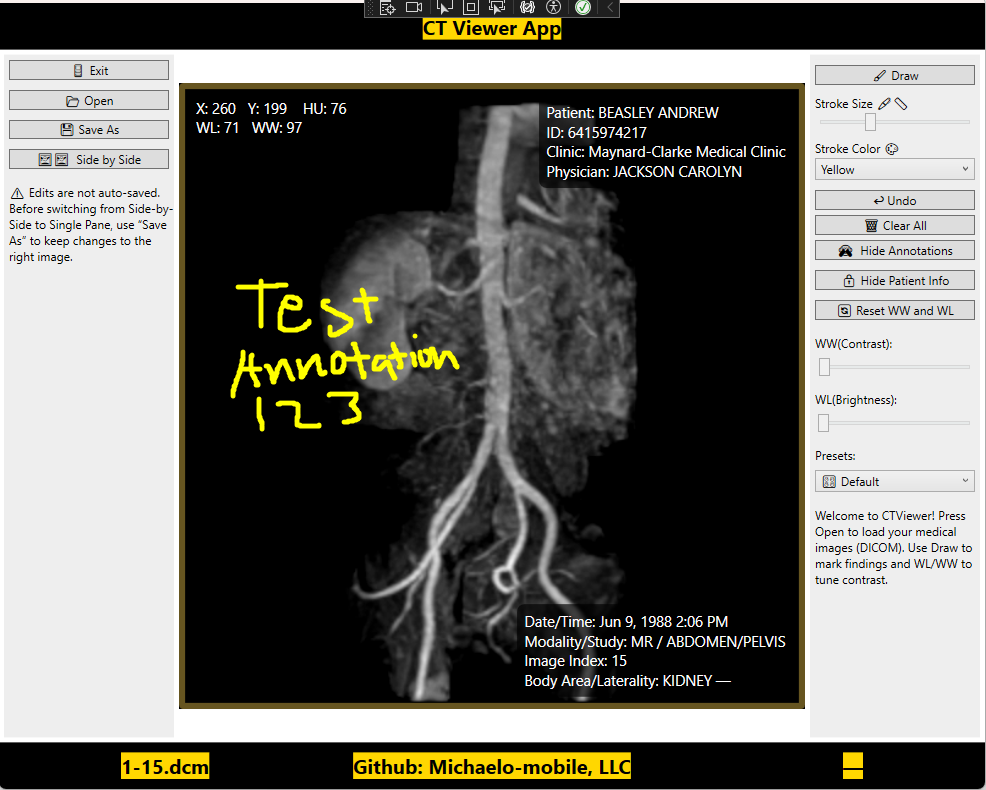
Each pane maintains its own independent state and controls. These state objects store the raw pixels, dimensions, WL/WW settings, file paths, stroke attributes, and visibility flags. While the application’s UI elements—sliders, preset buttons, and drawing tools—are shared, they operate only on the active pane, which is visually highlighted to guide the user.

The annotation system allows for freehand drawing on the InkCanvas, with user-selectable stroke size and color. A per-pane undo stack supports one-stroke-at-a-time reversal, while a clear function removes all annotations for that pane. Users can also hide annotations without deleting them by toggling the InkCanvas visibility.

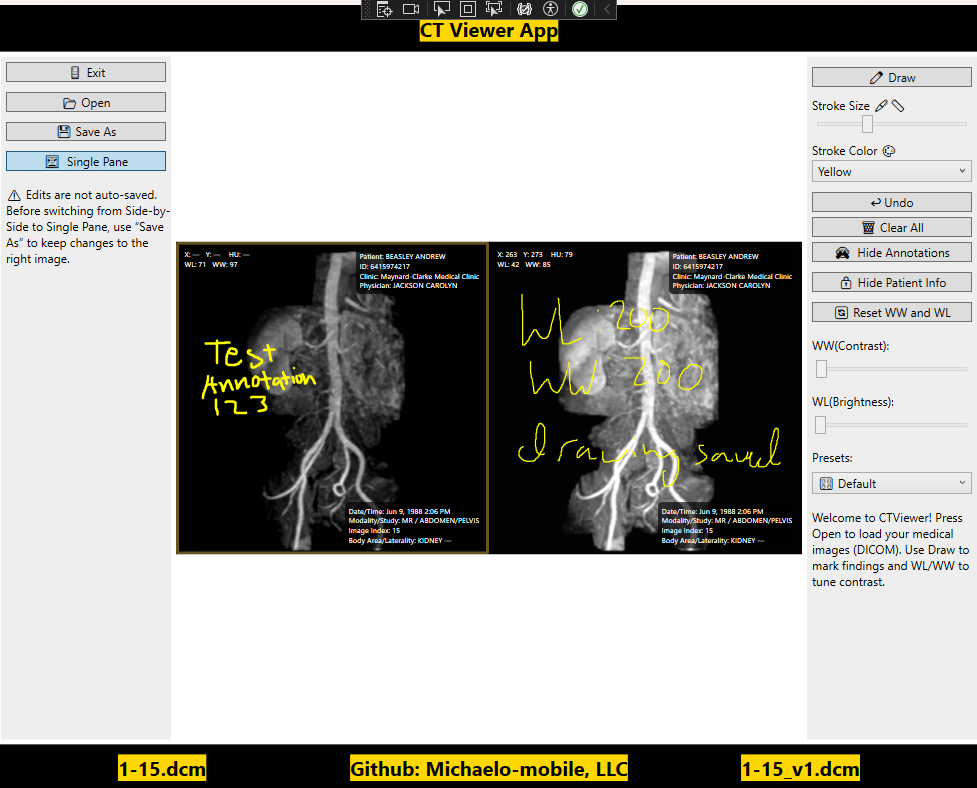
A side-by-side mode enables comparative viewing. When activated, the layout expands to include a second column for the right pane, automatically loading the next DICOM file from the folder if available. Each pane preserves its own WL/WW values, annotations, and overlays independently, allowing simultaneous but isolated interaction.

The overlay and HU readout system displays key patient and study metadata extracted from the dataset, such as patient name, ID, clinic, physician, date/time, modality, study name, image index, and body part/laterality. Additionally, a real-time X, Y, and HU (Hounsfield Unit) readout is provided by calculating the HU value from the pixel position using the rescale slope and intercept.

Finally, the application supports a non-destructive “Save As” workflow. Saving creates a new DICOM file containing updated VOI attributes (WindowWidth and WindowCenter) from the active pane, an application-specific private creator tag, and a private OB tag storing serialized annotation strokes. A new Service-Object Pair (SOP) Instance UID is generated to ensure the saved file is treated as a distinct object. Versioned filenames (e.g., \_v1, \_v2) are assigned automatically, ensuring the original file remains unmodified. Figures 2 and 3 below show the program in working single and side by side mode.



**Figure 2 – Single Pane Mode with Drawing Annotation**



**Figure 3 – Side by Side Mode with Drawing Annotations**

**Notable algorithms or data handling**

***Auto window/level via robust percentiles***

* + Sort 16-bit pixel buffer; take the 1st and 99th percentiles as minVal/maxVal.
  + Compute WW = maxVal − minVal, WL = (maxVal + minVal)/2.
  + Rationale: exclude outliers to enhance clinically relevant contrast.

***Linear window/level mapping (16-bit)***

* + Let min = WL − WW/2 and max = WL + WW/2.
  + For each pixel v:
    - 0 if v ≤ min, 65535 if v ≥ max, else ((v − min)/WW) × 65535.
  + Chosen over logarithmic mapping after experiments showed log scaling washed out highlights.

***Hounsfield Units (HU)***

* + For hover readout, compute HU = Raw16 \* RescaleSlope + RescaleIntercept (from DICOM tags) per pixel.

***Annotation persistence in DICOM***

* + Use a **private creator** tag (Group 0011) to mark app ownership, and a private OB element to store InkCanvas.Strokes bytes.
  + On load, detect creator + strokes to restore annotations; otherwise start clean.

***Per-pane state model***

* + Encapsulates each pane’s pixels, WL/WW, UI handles, stroke style, visibility, and “dirty” baselines, enabling independent interaction and clean synchronization with shared controls.

**Important design decisions and reasoning**

***Viewbox over manual transforms***

* + *Decision:* Wrap each pane’s fixed-size surface in a Viewbox (uniform scaling).
  + *Why:* Keeps Image and InkCanvas in the **same coordinate space**, so strokes stay aligned with pixels during resize and side-by-side layout. This avoids the common bug where zooming shifts strokes.

***Deprioritize zoom/pan (for now)***

* + *Decision:* Ship stable drawing + per-pane features first; omit zoom/pan after repeated misalignment issues (strokes not scaling with zoom).
  + *Why:* Time constraint. Correct zoom requires either (a) scaling the entire surface (image + ink) together, or (b) applying identical transforms to both with precise input hit-testing. This will be future work.

***Non-destructive “Save As” with new SOP Instance***

* + *Decision:* Always create a new DICOM instance and store annotations in private tags.
  + *Why:* Preserves source data integrity; keeps provenance clear; allows versioned outputs without touching originals.

***Percentile-based auto WL/WW***

* + *Decision:* Use 1–99% histogram clipping to compute WL/WW.
  + *Why:* Robust to outliers and gave images closer to professional viewers in testing; simpler and faster than full histogram equalization.

***Per-pane architecture***

* + *Decision:* Everything (WL/WW, strokes, overlays, visibility) is pane-scoped; shared controls bind to the *active* pane.
  + *Why:* Enables true side-by-side comparison with independent adjustments, a key UX differentiator.

***Consolidated XAML***

* + *Decision:* Fewer XAML files with cohesive panels (FileButtons, DrawingButtons, Sliders/Presets) instead of many small windows.
  + *Why:* Simplifies state management, reduces wiring, and sped up iteration.

***Git & feature branches***

* + *Decision:* Move to GitHub with feature branches mid-project.
  + *Why:* Enabled safe experimentation (e.g., side-by-side mode, save-as) and easy rollback; should be adopted earlier next time.

***Patient info privacy toggle***

* + *Decision:* Single control to show/hide overlays (applies to both panes).
  + *Why:* Keeps demos/screenshots clean and reduces accidental PHI exposure during teaching/testing.

**Testing & Results**

The project successfully opened valid DICOM files from CT sample datasets, fo-dicom examples, and anonymized hospital exports. Automatic window and level adjustments applied correctly, with sliders matching the initial WL/WW values. Presets such as Default, Bone, Lung, Soft Tissue, and Brain re-rendered images in real time with corresponding overlay updates. When loading a file, debug outputs below show confirmed successful parsing and initial window/level assignment.

[OPEN] DICOM loaded. Size: 512x512

[LOAD] WW=200 WL=100

Annotation tools allowed per-pane stroke size and color adjustments, undo actions, and clearing without affecting the other pane. Switching panes restored prior stroke attributes, as shown by debug lines below.

[STATE: ApplyInkEditingModeToActive] pane=Right active=True drawToggle=True and

[INK] Right stroke count=41.

Visibility toggles for annotations and patient info updated both the interface and the console logs, ensuring the intended state was always clear during testing. Not much testing was done as this was one of the methods that had the least amount of bugs.

Side-by-side mode maintained independent settings, annotations, and active pane highlighting for each view. The “Save As” function wrote WL/WW and serialized annotations to private DICOM tags, generated unique SOP Instance UIDs, and appended version numbers without altering the source file. Console confirmations below verified that both window/level values and drawings were preserved after saving and reloading.

[SAVE AS] pane=Right WL=100 WW=200 -> C:\...\image\_v2.dcm and

[LOAD] Ink strokes reloaded: 24

Performance remained strong, with typical 512×512 CT slices rendering in under 0.5 seconds (ApplyWindowLevelTo16Bit O(N) + BitmapSource.Create). UI responsiveness was instantaneous in Release build, and stability was validated by the absence of crashes after final refactoring.

**Challenges & Lessons Learned**

A key technical hurdle in development was implementing the Save As feature to reliably store both WL/WW adjustments and ink annotations without altering the original file. While the saving and loading methods (SaveWorkingDicomWithInk() and LoadInkAndWwWlFromDicom()) functioned correctly in isolation, integrating them into the live UI proved challenging. Debugging required targeted logging to trace the workflow from save to load to render, identifying that the issue lay in the loading stage where the interface failed to refresh after restoring state. Once corrected, the save/load cycle became stable and repeatable.

Ensuring that annotations remained perfectly aligned with the underlying DICOM image was another early challenge. Initial designs led to drift or scaling mismatches, especially when resizing the application window. The issue was resolved by binding both the image and its InkCanvas to a shared Grid within a ViewBox, guaranteeing identical scaling and consistent alignment across all size changes.

Midway through development, a PaneState class was introduced to replace the single hard-coded image/annotation pairing with a fully independent per-pane architecture. This refactor required tracking additional state variables, routing UI events to the active pane, and rewriting several methods to operate on dynamically selected targets. While the change increased complexity, it was essential for supporting features such as side-by-side mode and per-pane WL/WW, annotations, and visibility controls.

Finally, a series of small quality-of-life enhancements significantly improved usability without adding heavy development time. Persistent WL/WW values for each pane, sensible default stroke sizes, retained annotation colors, and clear file name displays made the viewer more intuitive and efficient to use. Testing confirmed that these refinements meaningfully improved the workflow, reinforcing the importance of thoughtful UI/UX design alongside core functionality.

**Conclusion & Future Improvements**

The CTViewer project successfully achieved its primary objectives by implementing reliable DICOM loading with full per-pane independence for WL/WW adjustments and annotations. Non-destructive saving was accomplished through embedding WL/WW values and ink annotations into private DICOM tags, preserving original files while enabling reloading without data loss. The application maintained smooth, responsive performance even with side-by-side viewing and numerous annotation strokes, while quality-of-life features—such as persistent per-pane settings, clear active-pane indicators, and per-pane drawing state memory—ensured a stable and intuitive user experience. Although some planned features, such as zoom and pan integration with annotation scaling, were deferred due to complexity and time constraints, the resulting application provides a reliable and efficient workflow for both single- and multi-pane viewing scenarios.

Looking ahead, several enhancements could further expand CTViewer’s capabilities. Image processing features such as gamma correction for non-linear brightness control, LUT/pseudo-coloring for highlighting subtle intensity changes, negative/inverted colors for improved X-ray fracture detection, edge enhancement using Sobel or Laplacian filters, and histogram equalization for global contrast improvement could greatly benefit clinical training and demonstration. UI and workflow upgrades could include zoom and pan with proper annotation scaling, a forward navigation button paired with undo, text box annotations, interface reorganization for better tool placement, save-as confirmation prompts, and full-screen or borderless viewing modes. From an architectural standpoint, continued method reorganization, logical grouping of controls, and modular development of new features would preserve maintainability while ensuring per-pane independence remains intact. These improvements would build upon the strong foundation already established, moving CTViewer toward a more feature-rich, customizable, and professional-grade DICOM viewer suitable for both research and clinical education.

**References**

* Throughout the development of the CT Viewer application, a variety of tools, libraries, tutorials, and reference materials were used to guide implementation and design decisions. These resources influenced both the technical execution and the feature set of the project.
* **Reference Software**
* **RadiAnt DICOM Viewer** – Used as a reference to understand typical medical imaging viewer tools, features, and workflows. Helped in determining which features to implement and how to structure user interactions.
* **Tutorials & Learning Resources**
* **C# WPF Tutorial – Kampa Plays**  
  *YouTube Playlist:* <https://youtube.com/playlist?list=PLih2KERbY1HHOOJ2C6FOrVXIwg4AZ-hk1&si=Ben62oNbNGHm4I7R>  
  Provided step-by-step WPF application development guidance, covering layouts, controls, data binding, and event handling.
* **YouTube Tutorials**
* <https://youtu.be/eLS9nDVJx5Y?si=4rLU67HC_7UyGjyk>
* <https://youtu.be/12METYFgXNI?si=0uRcrpsQ4TiU7RSP>
* <https://youtu.be/eLS9nDVJx5Y?si=gpPxjc7ew9eImHPY>  
  Covered practical C# coding patterns, UI techniques, and implementation of interactive features.
* **Libraries**
* **Fo-DICOM (Fellow Oak DICOM)**
* *GitHub Repository:* <https://github.com/fo-dicom/fo-dicom> – Full source code, sample projects, and developer discussions.
* *API Documentation:* <https://fo-dicom.github.io/html/N_FellowOak_Dicom.htm> – Lists namespaces, classes, and methods such as DicomFile.Open, GetSingleValueOrDefault, and Dataset.AddOrUpdate.  
  Primary library for reading, parsing, and writing DICOM files, as well as handling embedded metadata and pixel data.
* **Microsoft WPF / .NET Framework**
* *WPF Controls & Classes Reference:* <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/controls/> – Lists and explains all built-in WPF controls (InkCanvas, Grid, Viewbox, Slider, TextBlock, etc.) with links to method/property documentation.
* *InkCanvas Class:* <https://learn.microsoft.com/en-us/dotnet/api/system.windows.controls.inkcanvas> – Methods, events, and properties for InkCanvas (e.g., Strokes, EditingMode, DefaultDrawingAttributes).
* *BitmapSource.Create Method:* <https://learn.microsoft.com/en-us/dotnet/api/system.windows.media.imaging.bitmapsource.create> – Explains creating in-memory images from raw pixel buffers, used in ApplyWindowLevelTo16Bit.