Rendering Text as Ink with Color Syntax Highlighting on reMarkable Devices: An Analysis of rmscene and drawj2d

1. Introduction

The reMarkable paper tablet, particularly its Pro version with color capabilities, presents unique opportunities for displaying textual information, including source code with syntax highlighting. This report investigates methods for rendering text as editable ink within the reMarkable's native file formats (.rm and .rmdoc), with a specific focus on achieving color syntax highlighting. The analysis centers on two key community-developed tools: rmscene, a Python library for interpreting reMarkable files, and drawj2d, a Java-based program for generating vector graphics, including output compatible with reMarkable devices. The objective is to determine the feasibility and optimal strategies for creating syntax-highlighted documents that leverage the reMarkable Pro's color display.

2. Understanding reMarkable File Formats and Color Capabilities

A foundational understanding of the reMarkable's file structure and display technology is essential before exploring generation methods.

2.1. File Formats: .rm and .rmdoc

The reMarkable tablet primarily utilizes a proprietary file format for its notebooks.

- **.rm files**: These are the core files containing the actual drawing and note data. Historically, these files were known as .lines files.[1] They are structured as contiguous, uncompressed binary files, with data organized in a manner similar to plain old data structures in memory, using little-endian byte order.[2]
- **.rmdoc files**: This format is a ZIP archive that encapsulates one or more .rm page files along with metadata files (e.g., for page information, titles, preferences).[3] The .rmdoc structure is commonly used for transferring notebooks to the device via its USB web interface or the official desktop application.[3]
- **.rmn files**: Another packaging format encountered is .rmn, which is typically a tar archive of a notebook. This format is often associated with third-party tools like the reMarkable Connection Utility (RCU) for local device management.[3]

2.2. reMarkable Pro Color Display

The reMarkable Paper Pro introduces color capabilities, significantly expanding its potential

use cases.

- **Display Technology**: The Pro model features an 11.8-inch Canvas Color display, which is based on E Ink Gallery™ 3 technology.[5]
- **Resolution and Color Depth**: This screen offers a resolution of 2160×1620 pixels, resulting in a pixel density of 229 pixels per inch (PPI). It is capable of rendering approximately 20,000 colors.[5]
- **User-Selectable Colors**: The native note-taking application allows users to write and annotate using nine distinct base colors, which can also be blended to create a wider palette.[5] While the specific UI names of these nine base colors are not explicitly detailed in the general feature descriptions, tools like drawj2d provide a defined set of color names for interacting with the Pro's color system, as will be discussed later.

2.3. Color Encoding in .rm Files (General Context)

Color information is an integral part of the .rm file structure, evolving with the tablet's software versions.

- **Presence of Color Data**: The .rm file format, across versions 3 through 6, includes data for lines, color, and text.[7]
- **Early Versions (v3)**: The version 3 .rm file format, as documented by Plasma Ninja, utilized a 4-byte integer to specify the color of a line. At that stage, the display primarily supported black, gray, and white (effectively, using white as an eraser or for drawing on non-white backgrounds).[2]
- **Later Versions (v6)**: With reMarkable OS 3.0 and later, the file format advanced to version 6. This version is based around a Conflict-free Replicated Data Type (CRDT) sequence to manage content, including strokes and text.[8] The Python library rmscene is designed to parse these v6 CRDT sequences.[8]
- **Community Libraries**: The remarkable_lines Rust library is capable of parsing color information in .rm files from version 3 up to version 6.[7] Another Rust library, lines-are-rusty, also interacts with these files and notably added support for red and blue export colors, indicating early expansions of color capabilities beyond simple monochrome.[9] The reMarkable Connection Utility (RCU) also provides a thorough accounting of the v3, v5, and v6 lines formats, including color rendering.[8]

3. rmscene - Reading and Understanding reMarkable Files

The rmscene library is a significant community contribution for programmatically accessing data within reMarkable's .rm files.

3.1. Overview of rmscene

rmscene is a Python library specifically developed to read and interpret version 6 (.rm) files generated by reMarkable tablets running software version 3.x and above.[10] It is distributed

under the permissive MIT License.[10] While its initial motivation was to extract text content from these files, its capabilities extend to reading various data types, including drawn lines.[11] For users looking to convert .rm files into more standard formats like SVG, PDF, or Markdown, the related rmc tool leverages rmscene for the initial parsing phase.[11]

3.2. rmscene's Handling of Color Data

As the reMarkable platform evolved to include color, rmscene has incorporated functionalities to parse this information.

- **Support for New Colors**: Version 0.6.0 of rmscene introduced support for new pen types and colors, enabling the library to read and interpret the richer color data present in .rm files from devices like the reMarkable Pro, which operate on software version 3.x.[11]
- **Highlighter Color Parsing**: The development of color support is an ongoing process. An issue (#38, "Parse highlight colour") in the rmscene GitHub repository indicates active work or consideration for parsing highlighter colors.[14] Furthermore, the rmc tool, which depends on rmscene, had a corresponding issue (#20, "Fix highlight colour once rmscene is updated").[15] These development artifacts suggest that handling the full spectrum of color features, especially newer additions like highlighters, involves nuanced parsing logic and is subject to refinement. The very existence of these issues points to the underlying complexity of the file format and the community's efforts to keep pace.
- **Internal Representation**: Although rmscene can read and interpret color data from reMarkable Pro files, the specific internal Python enumerations or numerical values it uses to represent this full range of colors are not detailed in the available documentation.[11] Such implementation details would typically be found by examining the library's source code directly. Nevertheless, its ability to process these files implies a robust internal mapping system.

3.3. rmscene and Text Data

A primary strength of rmscene lies in its ability to extract and interpret textual content embedded within .rm files.

- **Text Extraction**: The library is proficient at extracting text, including GlyphRange scene items, which represent highlighted text within imported PDF documents.[11]
- **Text Formatting**: rmscene can parse text formatting attributes such as bold and italic, features introduced in reMarkable OS version 3.3. It achieves this by interpreting optional text properties like font-weight and font-style found within CrdtStr (CRDT String) objects.[11] This capability is crucial for accurately reconstructing the intended appearance and semantic structure of text within modern .rm files.

3.4. Experimental Writing Capabilities

Beyond reading, rmscene possesses some experimental functionalities for writing .rm files.

- **Status**: The library includes features for writing data back into the .rm format, though

these are explicitly marked as experimental.[11]

- **Primary Purpose**: This writing capability is primarily intended for testing the integrity of the parsing and writing process (round-trip testing) and for emulating different reMarkable software versions (e.g., by passing an option like {"version": "3.2.2"}).[11] Such emulation is valuable for ensuring compatibility and for testing how the library handles files created by different firmware iterations as new data fields are added to the format. The focus on emulation and round-trip testing underscores the inherent challenges in correctly generating valid files for a proprietary and evolving binary format.
- **Applicability to Colored Text Generation**: The available documentation does not suggest that rmscene's current writing capabilities are mature or specifically designed for the de novo creation of complex, arbitrarily colored text strokes, such as those required for syntax highlighting.[16]

3.5. Limitations for Direct Syntax Highlighting Generation

While rmscene is an invaluable tool for analyzing .rm files, including those containing color and text, its documented capabilities present limitations when considering it as a direct generation tool for complex syntax-highlighted content.

- The effort to programmatically construct the necessary CRDT sequences with precise color attributes, text content, and positional information from scratch would be substantial and error-prone.
- rmscene functions effectively as a "decoder ring" for the proprietary .rm v6 format, particularly concerning elements like color and text. Its principal strength is in making this complex, reverse-engineered format accessible to Python developers for analysis and data extraction. Without such libraries, each developer would face the daunting task of reverse-engineering the format independently.
- The "experimental" designation of rmscene's writing features, with their focus on round-tripping and version emulation, implies that writing arbitrary, complex .rm content (like intricately colored text for syntax highlighting) is a significantly more challenging problem than reading existing content. Reading involves parsing a known (albeit reverse-engineered) structure, whereas writing demands the correct construction of all necessary data structures, including the CRDTs, in a way that the reMarkable device will interpret flawlessly. Even minor errors in this construction could lead to corrupted or unreadable files on the device.
- Consequently, for the specific goal of generating new, syntax-highlighted text, rmscene is more likely to serve as a tool for *verifying* the output generated by another process (such as one using drawj2d) or for *extracting* text and color data from existing documents for analytical purposes, rather than acting as the primary creation engine. The library's development trajectory, such as the addition of color support in version 0.6.0 and ongoing work on highlighter parsing, often mirrors the feature updates in the official reMarkable software, indicating a responsive and adaptive community effort to keep pace with the evolving native format.

4. drawj2d: Generating Custom reMarkable Content

with Color

drawj2d offers a different approach, focusing on the generation of graphical content, including output for reMarkable devices, from a descriptive language.

4.1. Overview of drawj2d

drawj2d is a command-line program implemented in Java, designed to produce vector graphics drawn to scale based on a descriptive input language.[3]

- **Input Language**: It processes .hcl (a Human Config Language-like format) text files. These files contain commands that describe the drawing, such as line for drawing lines, label or text for adding text, font for setting font properties, and pen for defining stroke characteristics like color.[3]
- **Output Formats**: A key feature of drawj2d is its ability to output directly to reMarkable's native file formats:
    - .rm: Raw page data for a single page.[3]
    - .rmn: A tar archive of a notebook, suitable for transfer using tools like RCU.[3]
    - .rmdoc: A ZIP archive of a notebook, compatible with the reMarkable's USB web interface or the official desktop application for cloud synchronization.[3]
- **Editable Output**: Significantly, notebook pages created by drawj2d are editable on the reMarkable tablet itself, meaning that strokes and elements generated by drawj2d can be selected, moved, erased, or copied on the device, just like natively drawn content.[3] This makes drawj2d a powerful tool for creating content that integrates seamlessly into the reMarkable user experience.

4.2. Defining and Applying Colors for reMarkable Pro with drawj2d

With the advent of the reMarkable Pro, drawj2d has been updated to support its color capabilities.

- **Color Support**: Version 1.3.4 and later of drawj2d introduced or significantly enhanced color support specifically for users of the reMarkable Pro.[3]
- **Syntax for Setting Color**: The color for subsequent drawing operations is defined using the pen <color_name> command within the .hcl script. For instance, pen blue would set the drawing color to blue, and pen red would set it to red.[3]
- **Supported Colors and Mappings**: drawj2d recognizes a specific set of color names and handles mappings to the reMarkable Pro's palette. This provides a user-friendly abstraction over the device's internal color codes. The supported colors are detailed in Table 1.
- **RGB Matching for PDF/SVG Conversion**: When drawj2d is used to convert PDF or SVG files to reMarkable format, it applies colors from the source file only if their RGB values *exactly* match the predefined RGB values for the supported reMarkable Pro colors (e.g., red must be precisely 0xff0000). If a color in the source PDF/SVG is a close match but not exact, it will typically be rendered as black, grey, or white.[20] While this is more

pertinent to direct conversion workflows, it highlights the precision involved in color mapping.

Table 1: drawj2d Color Support for reMarkable Pro

drawj2d Color Name	Mapped From (if any)	reMarkable Pro Tool	Effective Color on Pro	Hex (if known from)	Notes
black	-	Pen	Black	000000 (assumed)	Standard black pen.
grey / gray	brown	Pen	Grey	N/A	
white	-	Pen	White	ffffff (assumed)	Used for drawing on dark backgrounds or as an eraser.
blue	inkblue	Pen	Blue	0000ff	
red	inkred, darkorange	Pen	Red	ff0000	
green	darkgreen	Pen	Green	00ff00	
yellow	orange	Pen	Yellow	ffff00	
cyan	-	Pen	Cyan	00ffff	
magenta	violet, purple	Pen	Magenta	ff00ff	
pink	-	Highlighter	Pink	ffafaf	Experimental highlighter color.
lightgray	-	Highlighter	Light Gray	N/A	Experimental highlighter color.
lightgreen	-	Highlighter	Light Green	a9fa5c	Experimental highlighter color, subject to change.
lightyellow	-	Highlighter	Light Yellow	fefd60	Experimental highlighter color, subject to change.

Note: drawj2d maps other specified color names (e.g., darkgray to black, darkgreen to green, orange to yellow pen, darkorange to red, brown to gray, violet and purple to magenta). Any other unrecognized color is mapped to black, grey, or white depending on its brightness.[3]

4.3. Text Rendering with drawj2d

drawj2d provides commands for rendering text, which can be combined with its color capabilities.

- **Text Commands**: The primary commands for adding text are label and text.[3] An example script (write.hcl from [22]), demonstrates reading text from an external file and rendering it:
  Terraform
  # In write.hcl, invoked with a text file as an argument
  # (Assuming Tcl-like syntax for.hcl as per typical drawj2d examples)
  # if {< $argc 1} { puts {usage: drawj2d write.hcl textfile}; return } ;# Basic argument check
  set textfile [lindex $argv 0]
  set f [open $textfile r] ; # Open file for reading
  font Lines 3.5 ; # Sets font to "Lines" at 3.5mm height
  m 15 5 ; # Moves to starting position (15mm from left, 5mm from top)
  text {} 138 ; # Initializes text block with max line length 138mm
  while {[gets $f line] >= 0} { ; # Read line by line
  text $line ; # Draws the current line
  }
  close $f

- **Font Options**:
  - **font Lines**: drawj2d includes a built-in single-line vector font, often referred to as a "Hershey" font, accessible via font Lines. It also offers font Lines italic and font LinesMono variants.[3] This font family supports Basic Latin, Latin-1, and several other European scripts (including Greek, Latvian, Polish, Russian letters).[22]
  - **TrueType Fonts**: While TrueType fonts can be specified, they are rendered as outlines only when outputting to the reMarkable format.[3] This behavior is a consequence of how the reMarkable's native file format handles (or doesn't fully embed) external font data, as interpreted and generated by drawj2d.
- **Applying Color to Text**: To render text in a specific color, the pen <color_name> command must be issued *before* the text or label command that creates the text. An example from the drawj2d documentation [21] illustrates this:
  Terraform
  font Lines
  m 15 10
  pen blue
  text {Some text in blue.}
  pen red
  linerel 50 0; # a red line follows

- **Limitations**: drawj2d does not support the "keyboard text" (typed text objects) feature introduced in reMarkable OS 3.0 and later.[22] Additionally, drawj2d does not perform automatic page breaks for very long blocks of text; however, the reMarkable's continuous page scrolling functionality, introduced in software version 3.0, can

accommodate long content generated as a single, scrollable page.[22]

4.4. Strategies for Implementing Color Syntax Highlighting with drawj2d

Leveraging drawj2d for color syntax highlighting requires an external script or program to pre-process the source code and generate the appropriate .hcl instructions.

- **External Parsing and .hcl Generation**:
  1. **Source Code Lexical Analysis**: The source code (e.g., Python, Java, C++) intended for highlighting must first be processed by a lexical analyzer (lexer). This lexer will break the code into a sequence of tokens, each tagged with a type (e.g., KEYWORD, COMMENT, STRING, IDENTIFIER, OPERATOR).
  2. **Token-to-Color Mapping**: A predefined scheme will map these token types to specific drawj2d-supported reMarkable Pro colors (referencing Table 1).
  3. **.hcl Script Generation**: The external script will then dynamically generate a .hcl file. For each token from the source code, it will:
     - Insert a pen <color_name> command to set the color for that token.
     - Use a text (or label) command to render the token's string content.
     - Employ moveto (absolute positioning) or linerel (relative positioning) commands to manage the cursor and place tokens correctly, replicating the original code's layout, including line breaks and indentation.
  4. **drawj2d Execution**: Finally, drawj2d is invoked with the generated .hcl file to produce the .rmdoc (or .rm) file. drawj2d -Trmdoc my_code_syntax.hcl -o my_code_highlighted.rmdoc
- **Conceptual Workflow Example**:
  1. **Input**: A source code file, e.g., example.py.
  2. **External Script (e.g., a Python script using a lexing library like Pygments):**
     - Lexes example.py into tokens: ('def', TokenType.Keyword), (' ', TokenType.Whitespace), ('my_function', TokenType.NameFunction), (':', TokenType.Punctuation), etc.
     - Maps token types to drawj2d colors: Keyword → blue, NameFunction → black, String → green, Comment → grey.
     - Generates an example.hcl file:

```
Terraform
font LinesMono 3.0 ; Set a monospaced font and size
m 10 10          ; Initial cursor position (x=10mm, y=10mm)

pen blue         ; Set color for keyword
text {def}       ; Render 'def'
m_offset 3*char_width 0 ; Advance cursor (char_width needs calculation)

pen black        ; Set color for whitespace (or handle spacing differently)
text { }         ; Render space
```

```
m_offset 1*char_width 0

pen black       ; Set color for function name
text {my_function}
m_offset 11*char_width 0

pen black       ; Set color for punctuation
text {:}
;... and so on, managing positions for all tokens, newlines, and indentation.
```

3. **drawj2d Processing**: java -jar drawj2d.jar -Trmdoc example.hcl -o example_highlighted.rmdoc
4. **Transfer**: The resulting example_highlighted.rmdoc is transferred to the reMarkable Pro.

This workflow illustrates that drawj2d provides the fundamental drawing primitives (specifically, colored text rendering and basic layout control) necessary to act as a rendering backend for syntax highlighting. However, the critical intelligence for parsing the source code, understanding its syntactic structure, and assigning appropriate styles must reside in the external pre-processing script. The .hcl file, in this context, becomes an intermediate, descriptive representation of the already styled text.

The choice of font within drawj2d (e.g., font Lines versus TrueType outlines) will significantly influence the visual appearance and readability of the syntax-highlighted code on the reMarkable. The font Lines family, particularly font LinesMono, is likely to provide a more "handwritten" or "inked" aesthetic that aligns with the reMarkable's design philosophy, while also offering the benefits of consistent character widths for layout. TrueType fonts, rendering as outlines, might appear less integrated or could prove harder to read for dense blocks of code. This decision represents a crucial trade-off between aesthetic preference and practical usability.

Effectively, a successful implementation of this strategy would involve creating a dedicated script or tool that functions as a "syntax highlighter to drawj2d compiler." This tool would encapsulate the complexities of parsing various programming languages, mapping syntactic elements to a color scheme, and meticulously generating the .hcl instructions to reconstruct the code's visual layout with colors. The most challenging aspect of this "compiler" will be the precise management of token positioning to accurately replicate the original code's indentation and spacing, for which a monospaced font is highly advantageous.

5. Comparative Analysis: rmscene vs. drawj2d for Generating Syntax-Highlighted Text

Choosing the right tool is paramount for efficiently achieving the goal of generating syntax-highlighted text on the reMarkable Pro.

5.1. rmscene Assessment

- **Strengths**:
  - Excellent for reading and parsing existing .rm v6 files, offering deep insights into their structure, including color and text elements.[11]
  - Capable of understanding and navigating the complex CRDT (Conflict-free Replicated Data Type) structure that underpins version 6 files.[8]
  - Highly useful for analysis, data extraction, or verification of .rm files generated by other means.
- **Weaknesses for Generation**:
  - Writing capabilities are explicitly experimental and primarily designed for round-trip testing or emulating different reMarkable software versions for compatibility purposes.[11]
  - No high-level API is documented for creating new, complex colored text layouts from scratch.
  - Directly constructing valid CRDT sequences for arbitrarily styled and positioned text elements would be an exceptionally complex and error-prone task for this specific use case.

5.2. drawj2d Assessment

- **Strengths**:
  - Specifically designed for generating vector graphics, including text, from a human-readable descriptive script (.hcl files).[3]
  - Provides explicit and robust support for outputting to native reMarkable formats (.rm, .rmn, .rmdoc) that result in pages editable on the device.[3]
  - Offers a clear and straightforward mechanism for defining pen colors (including highlighter colors for the Pro model) and applying them to text elements via .hcl commands.[3]
  - Its ability to convert PDF vector data into editable lines on the reMarkable demonstrates robust line and stroke generation capabilities.[3]
- **Weaknesses for Generation (or, more accurately, aspects requiring external logic)**:
  - Requires an external process to parse the source code and generate the intermediate .hcl script. drawj2d itself is a rendering engine, not a syntax analyzer.
  - Font limitations (the font Lines family for a native ink feel, or TrueType fonts rendered as outlines only) might impact the final aesthetic and readability.[3]
  - Manual layout control through coordinates in the .hcl script can become verbose and complex for intricate text arrangements.

5.3. Conclusion of Comparison

For the specific objective of *generating* new reMarkable documents featuring color syntax highlighting, drawj2d emerges as the significantly more direct, capable, and practical tool. This is due to its descriptive language approach, its well-defined color handling for the reMarkable Pro, and its native support for producing editable reMarkable output formats.

rmscene's role in such a project would be complementary rather than primary. It could be employed, for instance, to analyze the .rm files produced by drawj2d for verification or debugging, or for other tasks that involve reading and understanding existing reMarkable files. The two tools fundamentally address different aspects of the problem: rmscene is akin to a decompiler, allowing one to understand the reMarkable's native, "compiled" file format. In contrast, drawj2d provides a "source language" (.hcl) that can be used to generate content in that native format. This distinction is crucial: the user's project will effectively involve building a "compiler" that translates tokenized and styled source code into drawj2d's .hcl language. The main complexity will lie in this translation process, rather than in drawj2d's own capabilities once it receives a valid .hcl script.

Table 2: Feature Comparison for Syntax Highlighting Generation (rmscene vs. drawj2d)

Feature	rmscene	drawj2d
Primary Purpose	Reading/Parsing .rm files	Generating vector graphics from script
.rm/.rmdoc Writing Capability	Experimental, limited (primarily for testing)	Yes, native & robust support
Text Element Control (for generation)	Low-level (direct CRDT manipulation required)	High-level (via .hcl text/label commands)
Color Application Granularity (for generation)	Potentially fine-grained (if writing CRDTs directly, but highly complex)	Per text/shape element via pen command in .hcl script
Ease of Generating Complex Layouts from Scratch	Very difficult	Moderate (requires scripting .hcl, layout logic can be complex)
reMarkable Pro Color Support (for generation)	Not directly for generation; reads Pro colors	Yes, explicit color names & mappings defined
Suitability for Syntax Highlighting Generation	Low	High (when used with an external pre-processing script for .hcl generation)

6. Recommendations for Implementing Color Syntax Highlighting on reMarkable Pro

Based on the analysis of available tools and reMarkable Pro capabilities, the following strategy is recommended for implementing color syntax highlighting.

6.1. Recommended Tool: drawj2d

As established in the comparative analysis (Section 5), drawj2d is the preferred tool for generating the final .rm or .rmdoc files. Its descriptive .hcl language, support for reMarkable Pro colors, and ability to produce editable native files make it uniquely suited for this task.

6.2. Core Implementation Strategy: The "Compiler" Approach

The most effective method involves creating a system that translates source code into a .hcl script, which drawj2d then renders. This "compiler" approach can be broken down into the following steps:

1. **Source Code Lexical Analysis (Lexing)**:
    - Employ a lexer appropriate for the programming language(s) to be supported. Libraries like Python's Pygments offer robust lexers for a wide array of languages. Alternatively, a simpler, custom lexer could be developed for more limited or specific needs.
    - The lexer's role is to break down the input source code into a stream of tokens, where each token has an associated type (e.g., KEYWORD, COMMENT, STRING, IDENTIFIER, OPERATOR, NUMBER, WHITESPACE).
2. **Token Styling and Color Mapping**:
    - Define a style scheme that maps each relevant token type to a specific drawj2d-supported color for the reMarkable Pro (referencing Table 1 for available color names).
    - Select a font. font LinesMono [3] is highly recommended for code due to consistent character widths, which greatly simplifies layout calculations.
    - Determine a suitable font size (e.g., font LinesMono 3.0).
3. **.hcl Script Generation**:
    - This is the core of the "compiler." Programmatically construct a .hcl text file by iterating through the token stream generated in Step 1.
    - For each token:
        - If the token's type requires a color change, emit a pen <color_name> command.
        - Render the token's textual content using the text {token_string} command.
        - Crucially, manage the cursor position using moveto (absolute move, e.g., m x_coord y_coord), linerel (relative line), or similar drawj2d positioning commands. This is essential for handling line breaks, indentation, and spacing between tokens to accurately reconstruct the visual layout of the original source code. This part requires careful calculation of character widths and line heights.
        - Ensure font properties (e.g., font LinesMono 3.0) are set at the beginning of the .hcl script or as needed.
4. **drawj2d Execution**:
    - Invoke the drawj2d Java program via the command line. Pass the generated .hcl file as input, and specify the desired reMarkable output format (-Trmdoc for a full notebook document or -Trm for raw page data) and the output file name.[3]
    - Example command: java -jar drawj2d.jar -Trmdoc input_generated.hcl -o output_syntax_highlighted.rmdoc

6.3. Managing the reMarkable Pro Color Palette

To ensure predictable and visually appealing results, strict adherence to drawj2d's supported color names for the reMarkable Pro is necessary.[3] For enhanced usability, consider making the color scheme (the mapping of token types to reMarkable colors) user-configurable within the .hcl generation script.

6.4. Font Considerations

The choice of font significantly impacts readability and aesthetic.

- **font LinesMono**: As mentioned, this built-in monospaced vector font [3] is likely the best choice for rendering source code. Monospaced characters simplify the complex task of calculating text widths and managing alignment for indentation and spacing.
- **TrueType Fonts**: The "outlines only" rendering of TrueType fonts on reMarkable via drawj2d [3] might be aesthetically undesirable and could reduce readability for dense code.
- **Font Size**: Experimentation with different font sizes (e.g., font LinesMono 3.0 or font LinesMono 2.5) will be necessary to find an optimal balance between information density and legibility on the reMarkable Pro's screen.

6.5. Potential Challenges and Mitigation Strategies

Several challenges may arise during implementation:

- **Layout Complexity**: Accurately replicating the indentation and spacing of source code requires meticulous coordinate management within the .hcl script.
    - *Mitigation*: Consistently use a monospaced font. Develop robust logic within the .hcl generator to calculate character widths and line heights, processing the source code line by line and token by token to position elements precisely.
- **Performance for Large Files**: Generating very large .hcl files (for extensive source code) or the subsequent processing of these files by drawj2d might become slow.
    - *Mitigation*: Optimize the .hcl generation script to produce concise commands where possible. If drawj2d struggles with extremely long single pages, consider strategies for paginating the output into multiple .rm files within a single .rmdoc (though reMarkable's native continuous scroll feature for long pages helps alleviate this [22]).
- **drawj2d Limitations**: drawj2d provides basic colored text. It does not directly support richer text features like underlining specific parts of a text block or easily mixing different fonts or styles (e.g., bold, italic) on a per-token basis within a single text command.
    - *Mitigation*: Work within the existing capabilities of drawj2d. The primary goal is colored text for syntax differentiation. Additional styling might require more complex .hcl structures (e.g., breaking text into smaller pieces for style changes).
- **Error Handling**: The .hcl generation script must include robust error handling to manage unexpected input or issues during tokenization and layout calculation.

The success of this project largely depends on the quality and sophistication of the "Source Code to .hcl" translator. This component will house the bulk of the development effort. Its ability to correctly handle various programming language constructs, manage layout effectively, and produce optimized .hcl will determine the final quality of the syntax-highlighted documents. The choice of lexer/parser in the initial step will also directly influence the accuracy and granularity of the syntax highlighting; a more powerful, language-specific lexer will enable richer and more precise styling possibilities compared to simpler, regex-based approaches.
It is also worth considering the desired aesthetic: should the output aim to precisely match the appearance of syntax highlighting from a desktop Integrated Development Environment (IDE), or should it strive for a more "reMarkable-native" look and feel, perhaps by embracing the characteristics of the font Lines family? This decision will guide font and styling choices.
The described approach is not limited to source code; it could be generalized to render other types of structured, colored text on the reMarkable, such as formatted logs, comparison diffs, or other domain-specific textual data.

7. Conclusion

This investigation has explored the potential of rmscene and drawj2d for rendering text as ink, with a focus on achieving color syntax highlighting on the reMarkable Pro.

- **Summary of Findings**:
    - rmscene is a vital Python library for reading, parsing, and understanding the reMarkable's version 6 .rm file format, including its complex CRDT structure and embedded color and text information. Its current writing capabilities are experimental and primarily geared towards testing and format emulation, making it less suitable for the de novo generation of complex, syntax-highlighted documents.
    - drawj2d is the recommended tool for *generating* custom reMarkable documents (.rm or .rmdoc) that feature colored text for the reMarkable Pro. Its descriptive .hcl language provides the necessary commands for setting colors and rendering text elements, which can be effectively leveraged to produce syntax-highlighted output that is editable on the device.
    - The reMarkable Pro offers a distinct palette of pen and highlighter colors, which drawj2d supports through a defined set of color names and internal mappings, simplifying the process for developers.
- **Feasibility of Color Syntax Highlighting**: The creation of color syntax-highlighted documents on the reMarkable Pro is indeed feasible. The most promising pathway involves developing an external script that parses source code, maps syntactic tokens to drawj2d-compatible colors, and then generates a .hcl script for drawj2d to process into a native reMarkable file.
- **Key to Success**: The central development effort and the ultimate success of such a project lie in the creation of a robust and intelligent "source code to .hcl" translation logic. This translator must accurately tokenize the input code, apply the desired color

scheme, and meticulously manage the layout and positioning of text elements.

- **Outlook**: This approach empowers users to generate highly customized, syntax-highlighted documents, thereby enhancing the reMarkable Pro's utility for developers, students, and anyone working with structured textual information that benefits from color differentiation. This endeavor exemplifies the creative use of existing community-developed tools to extend the functionality of the reMarkable platform beyond its out-of-the-box features. Success in this area could inspire further development of tools aimed at generating other types of rich, structured content for e-ink devices, potentially fostering a broader ecosystem of "e-ink document compilers."

Works cited

1. plasma.ninja, accessed May 15, 2025, https://plasma.ninja/blog/devices/remarkable/binary/format/2017/12/26/reMarkable-lines-file-format.html#:~:text=lines%20(nowadays%20.,annotated%20PDFs)%20on%20the%20tablet.
2. reMarkable .lines File Format, accessed May 15, 2025, https://plasma.ninja/blog/devices/remarkable/binary/format/2017/12/26/reMarkable-lines-file-format.html
3. Drawj2d / Wiki / reMarkable, accessed May 15, 2025, https://sourceforge.net/p/drawj2d/wiki/reMarkable/
4. chopikus/rm-exporter: Export large notes & folders from reMarkable - GitHub, accessed May 15, 2025, https://github.com/chopikus/rm-exporter
5. reMarkable Paper Pro | reMarkable, accessed May 15, 2025, https://remarkable.com/store/remarkable-paper/pro/details/features
6. Compare paper tablets - reMarkable Paper Pro | reMarkable, accessed May 15, 2025, https://remarkable.com/store/remarkable-paper/pro/details/compare
7. remarkable_lines — Rust parser // Lib.rs, accessed May 15, 2025, https://lib.rs/crates/remarkable_lines
8. Remarkable files specification : r/RemarkableTablet - Reddit, accessed May 15, 2025, https://www.reddit.com/r/RemarkableTablet/comments/1fcmfu9/remarkable_files_specification/
9. ax3l/lines-are-rusty: Rust File API for the reMarkable tablet - GitHub, accessed May 15, 2025, https://github.com/ax3l/lines-are-rusty
10. Packages - remarkable - NixOS Search, accessed May 15, 2025, https://search.nixos.org/packages?channel=unstable&size=50&sort=relevance&query=remarkable
11. ricklupton/rmscene: Read v6 .rm files from the reMarkable tablet - GitHub, accessed May 15, 2025, https://github.com/ricklupton/rmscene
12. MIT license - ricklupton/rmscene - GitHub, accessed May 15, 2025, https://github.com/ricklupton/rmscene/blob/main/LICENSE
13. ricklupton/rmc: Convert to/from v6 .rm files from the reMarkable tablet - GitHub, accessed May 15, 2025, https://github.com/ricklupton/rmc
14. Issues · ricklupton/rmscene - GitHub, accessed May 15, 2025,

https://github.com/ricklupton/rmscene/issues
15. Issues · ricklupton/rmc - GitHub, accessed May 15, 2025, https://github.com/ricklupton/rmc/issues
16. Minecraft - Change Hardcoded Colors! (Resource Pack Tutorial) - YouTube, accessed May 15, 2025, https://www.youtube.com/watch?v=CGPpi9kjmmk
17. How to Create Awesome COLOR-CHANGING TEXT in Python - Pygame Tutorial - YouTube, accessed May 15, 2025, https://www.youtube.com/watch?v=VIeDvD5Nqvc
18. How to use re module to parse color names in text file? - Stack Overflow, accessed May 15, 2025, https://stackoverflow.com/questions/64622547/how-to-use-re-module-to-parse-color-names-in-text-file
19. Drawj2d / Wiki / Home - SourceForge, accessed May 15, 2025, https://sourceforge.net/p/drawj2d/wiki/Home/
20. Geometric shapes in rM notebooks - Drawj2d updated to support ..., accessed May 15, 2025, https://www.reddit.com/r/RemarkableTablet/comments/1fsf0uo/geometric_shapes_in_rm_notebooks_drawj2d_updated/
21. Drawj2d 1.3.4 released - SourceForge, accessed May 15, 2025, https://sourceforge.net/p/drawj2d/blog/2024/09/drawj2d-134-released/
22. Drawj2d / Wiki / Examples_reMarkable - SourceForge, accessed May 15, 2025, https://sourceforge.net/p/drawj2d/wiki/Examples_reMarkable/
23. Drawj2d, accessed May 15, 2025, https://drawj2d.sourceforge.io/
24. Drawj2d download walkthrough : r/RemarkableTablet - Reddit, accessed May 15, 2025, https://www.reddit.com/r/RemarkableTablet/comments/1bfcs56/drawj2d_download_walkthrough/