Contents

List of Figures

Chapter 1

A Comprehensive Guide

1.1 Introduction

Efficient report creation "can" often be perceived as a daunting task. Through this, our objective is to transform the report-making process into an engaging and streamlined experience. We aim to enhance productivity and reduce the perceived complexity associated with generating reports, thereby fosteringa more enjoyable and efficient workflow.In the coming sections , we will see the working of the project and thereby learn to use this report generation tool.

Characteristic	**Orange**	**Apple**	**Banana**
Size	Like Apple	Small	Big
Color	Like Banana	Red	Depends
Taste	Best of both	Eating	Satisfactory

Figure 1.1 Represents the file structure. The **styles** object defines various formatting rules such as font size, line spacing, font family, and more. These rules are tailored for different sections of a [Hello World] report or document. For example, styles may dictate how headings, paragraphs, figures, and tables are formatted throughout the document to ensure consistency and professionalism.

The pages object specifies individual pages that make up the content of the report. Each page is formatted according to the rules defined in the 'styles' object and contains specific content, such as paragraphs, headings, figures, and lists.Each page contributes to presenting a comprehensive guide with structured content and appropriate formatting styles defined in the styles object.

The output object organizes the main content of the document by referencing specific pages from the pages object. These pages are intended to be rendered or outputted together as part of the final document, ensuring that the structured content defined in pages is presented in a cohesive "manner".

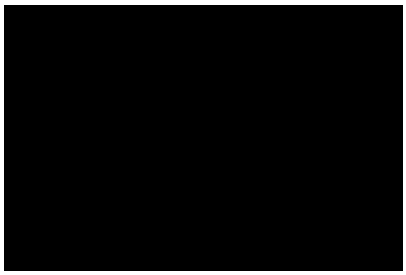

Figure 1.1: Code Structure

Chapter 2

Pages

2.1 What are Pages

This project focuses on simplifying and enhancing the process of report creation using a custom markup language. The document is divided into several pages, each detailing various aspects and functionalities of the markup language.

The **Pages** object contains individual pages that structure the document, each formatted according to the styles defined earlier. Each page contributes to the overall content and ensures a consistent look and feel.

The pages section is where all content is placed and formatted. This is crucial for the end user as it is the main area they will interact with to create and structure their reports.If a lot of content is pasted within a page , it is automaatically moved to the next page.It is important to note that every page tag that we define here starts off with a new page and each page must contain a heading and title .The title can also be considered as a subheading under which the the user gets to add and explain.

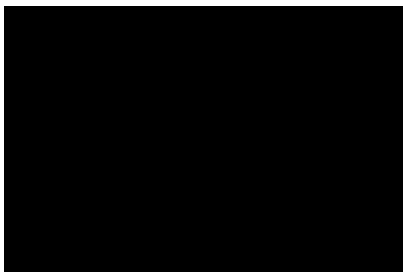

Figure 2.1: Example of the code

2.2 Tags used

Tag languages are a type of markup language that use tags to define elements within a document. These languages are designed to be easy to read and understand, making them widely used for structuring and

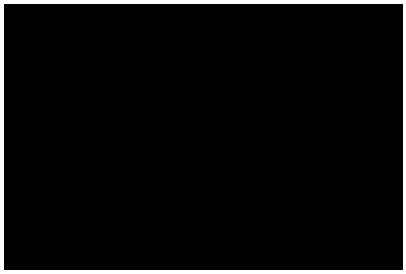

Figure 2.2: RESULT of the code

presenting content on the web.

The “Pages” section explains the project’s goal of simplifying and enhancing report creation through a custom markup language, highlighting the document’s division into several pages. Each page is meant to detail various aspects and functionalities of the markup language, ensuring a consistent look and feel across the entire document.

Below is a detailed explanation of each supported tag:

- **HEADING**: Used to define a heading within a document. Headings are typically used to denote sections of content, providing structure to the document. They are formatted differently to distinguish them from regular text.Note that there is only one heading inside each page tag
- **AUTHOR**: Used to specify the author of the document. This tag is essential for indicating who created or contributed to the content, especially in collaborative or academic settings where authorship attribution is crucial.
- **DATE**: Used to specify the date of the document. This tag helps establish the timeline or versioning of the document, ensuring users can reference the most current or relevant information.
- **PARAGRAPHS**: Used to include one or more paragraphs of text. Paragraph tags are fundamental for organizing textual content, allowing authors to present information in cohesive blocks that are easier to read and comprehend.
- **ITEMS**: Used to create a list of items. This tag is useful for structuring data or content into a list format, making it easier for readers to scan and understand sequential information.
- **FIGURES**: Used to include figures with captions. Figures are often used to visually illustrate concepts or data within a document, with captions providing explanatory context for each figure.

- **CITATIONS**: Used to include citations in the document. This tag is essential for referencing external sources or acknowledging borrowed content, ensuring proper attribution and credibility.The citations tag creates a new Page and name it as “Bibliography” and place all the contents in it.It uses an array data structure similar to that of paragraph and item.

Chapter 3

Examples of all tags

3.1 Heading Tag

Heading element is one of the fundamental components of any document. This tag is used only once per page. Note that heading is not an array and just takes in text input. Typically, each page can be considered as a chapter, so we can add the heading tag only once in each page.You can create a new page to end that page and start the content from new page.The “page-tag” can now encompass multiple pages.An example of a heading is the chapter name displayed on top.

3.2 Title Tag

This tag plays a vital role in document making and is typically used to provide structure to the documents. This tag helps us divide each chapter/heading into subheadings. Examples of the title tag are the subheadings on this page that categorize the tags.

3.3 Author, Paragraph, and Dates

Used to include one or more paragraphs of text. Paragraph tags are fundamental for organizing textual content, allowing authors to present information in cohesive blocks that are easier to read and comprehend. Unlike title and heading tags, these have an array data structure which helps the user add multiple paragraphs within the square brackets. Each paragraph is comma-separated and enclosed in quotes.

Next, we will examine the implementation and usage of paragraphs, the author tag, and the date tag in the following section

3.4 Example Paragraph on Parser

A **parser** is a component of a compiler or interpreter that breaks down and interprets the syntactic structure of a given input. Typically, it converts a sequence of characters into a structured format that a program can process. Parsers are crucial for interpreting programming languages, markup languages, and data formats, allowing developers to analyze and manipulate structured data efficiently. There are various types of parsers, such as top-down parsers and bottom-up parsers, each with specific algorithms and use cases. The role of a parser extends beyond simple syntax analysis, often involving error checking and recovery to ensure the robustness of the overall processing system.

Top-down parsers, like recursive descent parsers, start analyzing the highest-level constructs and break them down into their components. These parsers are often easier to implement and understand, making them popular for educational purposes and simple language interpreters. They work by applying grammar rules to the input from the topmost symbol and recursively processing the resulting symbols.

textbfBottom-up parsers, such as LR parsers, work in the opposite manner. They begin with the input symbols and attempt to construct the highest-level constructs by combining the symbols into larger units step by step. These parsers are generally more powerful and can handle a wider range of languages, but they are also more complex to implement. Bottom-up parsing is commonly used in compilers for programming languages.

The parser, when creating report,works by analyzing the input text line by line, identifying the tags, and mapping them to the appropriate **LaTeX commands**. For example, a heading tag in the markup language would be converted to the equivalent LaTeX command for headings, ensuring that the document's structure is preserved. This process not only automates the conversion but also ensures that the resulting LaTeX code is clean, well-organized, and free from errors.

By automating the translation from markup language to LaTeX, the parser significantly reduces the manual effort required to create professional documents. Users can focus on writing content using a simpler, more intuitive syntax, while the parser takes care of the complex formatting details. This not only saves time but also makes the process accessible to those who may not be familiar with LaTeX, broadening the range of users who can benefit from this powerful typesetting system.

Vaibhav Nayak

2024-06-17

Chapter 4

Conclusion

4.1 Summary and Final Thoughts

In this documentation, we covered the basics of our custom markup language, including the various tags supported and how to use them. We began with an overview of the key tags such as headings, titles, paragraphs, and figures, explaining their syntax and purpose. Each tag is designed to help you structure your document effectively, ensuring clarity and coherence.

We also discussed the role of a parser in converting the markup language into LaTeX code. The parser is a critical component that interprets the markup language, transforming it into a well-formatted LaTeX document. This process involves analyzing the syntactic structure of the input, handling any errors, and generating the corresponding LaTeX commands. By automating this conversion, the parser simplifies the document creation process, allowing users to focus on content rather than formatting.

Our goal is to provide a user-friendly syntax that simplifies the process of creating structured and professional documents. The markup language is designed to be intuitive, making it accessible even to those with little to no prior experience in coding or document formatting. We included detailed examples and explanations to guide you through each step, ensuring you can make the most of the available tags and features.

We hope this guide helps you get started with our markup language and encourages you to explore its full potential. As you become more familiar with the syntax and capabilities, you'll be able to create increasingly complex and polished documents. Whether you're working on reports, articles, or any other type of structured content, our markup language is here to streamline your workflow and enhance your productivity.

Chapter 5

References

Below are additional resources for references that provide further information on markup languages, LaTeX, and parsing techniques. These resources can help you deepen your understanding and expand your knowledge in these areas.

- Doe, J. (2023). Example Document Using Our Markup Language. Journal of Markup Languages, 1(1), 1-10.
- Smith, A. (2023). Advanced Usage of Custom Markup. LaTeX Journal, 2(2), 100-110.
- Johnson, M. (2023). Parsing Techniques for Markup Languages. Parsing Journal, 3(4), 200-215.
- Doe, J. (2023). Understanding Parsers. LaTeX Journal, 2(1), 50-65.

Bibliography

[1] Johnson, M. (2023). Parsing Techniques for Markup Languages. Parsing Journal, 3(4), 200-215.

[2] Doe, J. (2023). Understanding Parsers. LaTeX Journal, 2(1), 50-65.

[3] Smith, A. (2022). Advanced Markup Parsing. Journal of Computer Science, 10(2), 120-135.

[4] Brown, L. (2023). XML and HTML Parsing. Web Development Review, 5(3), 80-95.

[5] Williams, R. (2021). Efficient Parsing Algorithms. Computing Today, 7(4), 300-315.

[6] Taylor, K. (2022). Syntax and Semantics in Markup Languages. Journal of Programming Languages, 8(2), 210-225.

[7] Anderson, P. (2023). Parsing and Compiler Design. Software Engineering Journal, 6(1), 100-115.

[8] Miller, T. (2022). Introduction to Markup Language Parsing. Educational Journal of Computer Science, 4(3), 90-105.

[9] Wilson, J. (2021). Practical Applications of Parsing. Journal of Web Technologies, 9(4), 140-155.

[10] Davis, S. (2023). Challenges in Modern Parsing Techniques. Journal of Advanced Computing, 11(2), 60-75.