FOR STUDENTS WITH DISABILITIES
Journal of Science Education
ISLAND 2021 ARTICLE

Using Markup Languages for Accessible Scientific, Technical, and Scholarly Document Creation

Jason J.G. White[1]

[1]Educational Testing Service

Abstract

In using software to write a scientific, technical, or other scholarly document, authors have essentially two options. They can either write it in a 'what you see is what you get' (WYSIWYG) editor such as a word processor, or write it in a text editor using a markup language such as HTML, LATEX, Markdown, or AsciiDoc. This paper gives an overview of the latter approach, focusing on both the non-visual accessibility of the writing process, and that of the documents produced. Currently popular markup languages and established tools associated with them are introduced. Support for mathematical notation is considered. In addition, domain-specific programming languages for constructing various types of diagrams can be well integrated into the document production process. These languages offer interesting potential to facilitate the non-visual creation of graphical content, while raising insufficiently explored research questions. The flexibility with which documents written in current markup languages can be converted to different output formats is emphasized. These formats include HTML, EPUB, and PDF, as well as file formats used by contemporary word processors. Such conversion facilities can serve as means of enhancing the accessibility of a document both for the author (during the editing and proofreading process) and for those among the document's recipients who use assistive technologies, such as screen readers and screen magnifiers. Current developments associated with markup languages and the accessibility of scientific or technical documents are described. The paper concludes with general commentary, together with a summary of opportunities for further research and software development.

***Corresponding Author, Jason J.G. White** jason@jasonjgw.net
Submitted March 2, 2022
Accepted April 25, 2022
Published online October 27, 2022
DOI: 10.14448/jsesd.14.0005

Introduction

Two alternative methods of preparing electronic documents are in widespread use today. The first approach is to use a 'what you see is what you get' (WYSIWYG) editor—usually a word processor—for writing and revision. In this case, the presentation of the document in the editing environment somewhat resembles its final form as displayed or printed, including layout and choice of fonts. Control over presentation is exercised entirely through the graphical user interface of the application. In currently popular word processors, the underlying markup codes are largely hidden from the user. There is no 'reveal markup' mode, such as that provided by the once popular WordPerfect word processor. The second option is to write the document in a plain text file, annotating the text with markup language code that influences its later processing, including its layout and presentation. To generate a rendering of the document, a separate program is run, typically via editor commands or using a command line interface.

Although it is not the purpose of this paper to compare word processing with markup-based document authoring, differences between the two approaches are noted as they arise. An empirical study by Knauff and Nejasmic (2014) found that writing text in a word processor was less error prone and more efficient than writing in LATEX for document transcription tasks, for both more and less experienced LATEX users. The performance of LATEX users was greater, however, in transcribing equations. Moorhead (2020) acknowledges the significance of this study as perhaps the only peer-reviewed investigation of its kind, while noting that it has been criticized on methodological grounds. See the discussion in Moorhead (2020) and the sources there cited. The considerations in favor of LATEX tend to be founded on qualitative sources of evidence (see e.g., Bahls Wray, 2015; Sotomayor-Beltran, Barriales, Lara-Herrera, 2021; Wright, 2010). There have also been anecdotal reports of the experience of its use by students who are blind or vision-impaired (Ahmetovic et al., 2021; Zu Bexten Jung, 2002).

The markup language-based alternative to creating documents in a WYSIWYG editor, which is to be considered here, remains particularly prominent in mathematical, scientific and other broadly technical disciplines. It is not, of course, exclusive to these disciplines, as it offers a generally applicable paradigm for document development suitable for a wide variety of applications. Nevertheless, the relevance of using markup languages to write and revise documents in scientific and technical fields provides an important justification for considering the potential accessibility-related benefits of this approach. It is also reasonable to expect that students and professionals working in such fields (especially at the undergraduate level and above) are more likely to possess technical skills that facilitate the practical use of text editors and tools related to markup languages, such as familiarity with elementary programming concepts and command line interfaces. Knowledge of the UNIX command line interface, including programming concepts, as introduced in texts such as Shotts (2019), constitutes valuable, though not indispensable, background to using many of the tools and strategies described here.

This paper is motivated by two commitments. First, the accessibility-relevant characteristics of markup language use are considered both from the perspective of the document's author in writing and editing the material, and in relation to the quality of what is ultimately made available to readers (i.e., the accessibility of the formats that can be produced by processing the marked up input text). Second, the scope of

DOI: 10.14448/jsesd.13.0005

the discussion is limited to issues of non-visual access as encountered by authors and readers who use speech or braille output as their only, or at least principal means of interaction. There are two reasons for this restriction: pragmatically, it confines the subject-matter of the paper within reasonable bounds, and, more importantly, it focuses the exposition on issues with which the author has had experience. Hence, the discussion is more a reflection of the author's experience of working with markup languages in an entirely non-visual setting, than a comment on the relatively small body of relevant scholarly literature. Nevertheless, attention is devoted to recent developments of significance in this field, and to identifying insufficiently explored research questions which, in the author's view, merit further attention.

As is implied by the contrast with the WYSIWYG approach to editing, the markup languages to be considered here are those which can conveniently be written and manipulated in a text editor. These languages include LATEX,[1] Hypertext Markup Language (HTML), (Web Hypertext Application Technology Working Group, 2021), various XML-based formats, such as DocBook (Walsh Hamilton, 2010), and 'light-weight' markup languages, for example, Markdown,[2] AsciiDoc (Allen, White, individual Asci- iDoctor contributors, 2021), and ReStructuredText (Jones, 2021). The file formats used by word processors, presentation tools, and graphical office applications more generally tend to be poorly suited to direct, markup-based editing. Although the most important of these formats are applications of XML,[3] their complexity and syntactic verbosity preclude creating and editing document instances in a text editor. For practical purposes, documents in such formats are best produced in the graphical office applications which support them, or by conversion from another markup language that is more amenable to manual editing.

In the sections that follow, an admittedly artificial distinction is drawn between the author's and the reader's perspectives, while acknowledging that, in the process of writing, reviewing and revising a document, the same person alternately performs the functions of author of the marked up text, and reader of a rendered product. Nevertheless, to organize the discussion conveniently, issues of editing are addressed first, followed by consideration of the accessibility of the document upon conversion to formats in which it is ultimately read, not only by the author but by its intended audience.

Writing and Editing Documents in Markup Languages

A fundamentally important consequence of writing a document in a markup language using a text editor, and then using appropriate tools to convert it to desired output formats for reading and distribution, is that authors have the freedom to choose their preferred editing software and accompanying tools. As automatic citation and bibliography generation programs used by markup language processors retrieve bibliographical entries from text files in specialized formats such as BibTeX, the author is also free to avoid using graphical reference manage-

[1]For recent introductions, see Flynn (2021), Grätzer (2016), and Kottwitz (2021).

[2]There are multiple versions of the Markdown format, among the most useful of which is that supported by the Pandoc document conversion tool. See Mailund (2019) and MacFarlane (2021) for further details.

[3]The formats that provide the basis of Microsoft's office suite are standardized as Office Open XML (OOXML) (International Organization for Standardization and International Electrotechnical Commission, 2016). The file formats used in applications such as the open-source LibreOffice office suite are standardized as Open Document Format (ODF) (International Organization for Standardization and International Electrotechnical Commission, 2015).

DOI: 10.14448/jsesd.13.0005

ment tools and hence to avoid their accessibility-related limitations. The ability to choose a text editor, and to manipulate the document with any software capable of operating on text files, constitutes a fundamental departure from the approach taken by word processors, in which the editing and formatting functions are inseparably integrated into a single application. In principle, any text editor capable of manipulating plain text files will suffice. This flexibility enables a well informed author to make choices that satisfy her or his accessibility-related needs, as well as other software-related preferences, which may vary according to the demands of the task and the situation. For example, commercially available braille displays and braille note-taking devices commonly support the creation and editing of text files. These portable systems can thus be used for writing and editing markup-based documents, which may later be transferred elsewhere for conversion to intended output formats or for additional processing. A user may rely on the text editing functionality of a braille device primarily for writing notes, or it may also be the preferred means of composing longer documents. Much depends on the demands of the task, and on the extent to which these can be satisfied by the editing functionality of the device which is available to the user.

A Brief Survey of Text Editors for Desktop Operating Systems

Under desktop operating systems such as Linux, Apple Mac OS, and Microsoft Windows, there is a broad choice of available text editing applications. In these software environments, it is again a question of selecting tools that correspond to the skills, the willingness to learn, and the needs of the individual. Text editors vary greatly in the support they provide for editing and manipulating different markup languages. While they can all achieve it, at least in principle, some editors offer features specific to the markup language in use that can enable the author to work more efficiently and to avoid or correct syntax errors. Editing environments that offer more features and which provide keyboard-based functionality that promotes efficiency also necessitate more learning on the user's part, but the skills thus acquired, as is true of proficiency in programming concepts and command line interfaces more widely, can be of long-lasting value from the perspective of an entire career in a scientific or technical profession.

At the most sophisticated end of the scale are editors popular among software developers and professional system administrators, which typically offer functionality or software extensions tailored to various markup languages, including Markdown, HTML, XML, and LATEX. The Emacs editor[4] is particularly accessible to speech users, thanks to the Emacspeak software (Raman, 1997, 2021), which provides a highly customized and efficient spoken interface not only to the central functions of the editor itself, but also to a wide variety of its extension packages. Emacs is also somewhat accessible using a screen reader in the Microsoft Windows environment, but in this case, one loses the advantages of the Emacspeak spoken interface. Indeed, the author of this paper found Emacs, together with Emacspeak and a braille display, under the Linux operating system, to be a preferable environment for writing his doctoral thesis in con- temporary Philosophy of Language. More specifically, AUCTeX mode—an extension of Emacs for editing LATEX documents—was used; see "AUCTeX" (2020) for documentation. To facilitate the editing of technical

[4]See the official Emacs documentation (Free Software Foundation, Inc., 2021), and Cameron, Elliott, Raymond, and Rosenblatt (2009) for an introductory text.

DOI: 10.14448/jsesd.13.0005

content in LATEX documents, Emacspeak can provide a spoken rendering of mathematical expressions (i.e., notation occurring in math mode), while indicating incidents of markup errors in the mathematics (Sorge, 2016b). Structural navigation of the notation is also possible (Sorge, 2016b). For a description and an empirical evaluation of somewhat similar functionality implemented as an extension to the Eclipse software development environment under Microsoft Windows, see Manzoor et al. (2019).

Modern descendants of the Vi editor—long regarded as the principal rival to Emacs—also allow for the development of software extensions that facilitate markup editing. For example, the VimTeX extension ("VimTeX", n.d.) facilitates the editing of LATEX documents. In contemporary usage, Vim, and a relatively recent derivative, Neovim, continue to attract software development effort. See Robbins and Hannah (2021) and McDonnell (2014) for introductions to the Vim editor. Vim is relatively accessible with a screen reader as a terminal application under Linux, Mac OS or Windows operating systems. (The user interface of the Neovim editor is essentially similar, and the two projects need not therefore be distinguished for purposes of this discussion.) However, the extent to which text is spoken automatically as the user performs navigational and editing commands varies depending on the screen reader in use. A satisfying experience is more likely for braille display users, or for users of Linux-based screen readers generally, which are designed to be effective in a text-based terminal environment.

Unlike text editors designed primarily to offer graphical user interfaces, Emacs and Vim are both centered on the use of keyboard commands. Each editor offers a rich repertoire of such commands, which can be further enhanced by the installation of extension packages (e.g., for use with particular markup languages). Either editor thus offers the non-visual user an opportunity to work with software in which the keyboard is intended as the primary means of control, rather than as secondary to mouse or touch input as in a graphical interface. Consequently, the available keyboard commands are more extensive than typically found in other environments, and the documentation describing the use of the editors often refers to the keyboard interface rather than to menus or graphical operations. It is not clear whether the distinctive, modal approach taken by the Vi and Vim editors, in which many of the standard keys on the keyboard are assigned by default to editor commands, and may be used for text entry only in insert mode, has any particular advantages or drawbacks with respect to accessibility with screen readers. It is more likely to be, as for the user population more broadly, a question of personal preference.

Among the more feature-rich graphical editors, Microsoft's Visual Studio Code is especially relevant, in that its accessibility-related features, including support for screen readers (Microsoft Corporation, 2021) continue to be enhanced. The editor includes facilities by default for editing Markdown documents, and extension packages are available, for example for editing and processing LATEX documents. Visual Studio Code is designed to be accessible with screen readers under Linux, Mac OS and Windows environments, as the application is built on Web technologies using the Electron framework. As of the time of writing, there remain accessibility-related issues that limit its effectiveness in editing markup-language documents. In particular, word wrap is unavailable if a screen reader is in use, and hence each logical line of text (terminated by a new-line character), however long,

DOI: 10.14448/jsesd.13.0005

is treated as a single line for purposes of cursor navigation. This problem can be mitigated to some extent by installing an extension that introduces line breaks into the file as text is entered;[5] however, wrapping the lines of the text file may not be desirable, for instance if one is working with collaborators who have different window size preferences or who require magnification. Another alternative is to adopt the convention of using 'semantic line breaks' in the source text (Matt, n.d.), in which each clause or sentence in a paragraph is terminated by a new line.

There are also specialized text editors designed for working with LATEX documents. Of these, TeXShop— designed for the Mac OS environment—appears to be relatively accessible with a screen reader, except for its PDF viewing functionality. TeXShop is included in the MacTeX software distribution (TeX Users Group, n.d.). The features provided by such custom-designed editors are similar, in important respects, to those of the extensions available for the editors already described that provide specialized support for LATEX. However, the LATEX-specific editors do not include the wealth of keyboard commands distinctive of Emacs or Vim.

As indicated earlier, any text editor suitable for manipulating text files may be used for writing and revising markup-language documents. General-purpose, graphical editors of significance in this connection include TextMate (MacroMates Ltd., 2021), which runs in the Mac OS environment, and Notepad++ (Ho, n.d.) under Microsoft Windows. TextMate notably provides extensions for working with markup languages, including Markdown, LATEX, and HTML documents. These features, combined with its good screen reader compatibility, ensure that it is a promising option for a variety of editing tasks.

[5]See the discussion of the issue and its work-around in "Word wrap should not be disabled when accessibility is turned on 95428" (2021).

Markup-Related Features of Text Editors

The markup language-specific features of text editors vary greatly, according to the markup language used, the text editor, and any installed extension packages. Although these features are designed for a general user population, they can especially benefit screen reader users by improving efficiency and assisting in the prevention or correction of syntax errors in the use of the markup language. Thus, there are typically menus and keyboard commands for inserting frequently needed document structures, such as headings and lists. Keyboard commands can also typically be used (e.g., the tab key or the escape key) to complete a partially typed markup language code, such as an HTML element or a LATEX command or environment, thus enhancing typing efficiency while contributing to the avoidance of errors. Similarly, keyboard operations can be used quickly to close markup elements by inserting a closing tag in an HTML or XML document. The text editor may also enable sections of a document to be selectively expanded or collapsed, via a code folding feature, thus creating an outline. Some editing environments also provide navigational commands for moving the cursor by structural components of the document, such as section headings, and commands for manipulating these objects may be available as well, for example in the Vim editor. Some editors, including TextMate, Emacs and Vim also permit multiple place markers to be set at specific cursor locations, to which the user can quickly return by issuing a keyboard command. Since screen readers present text primarily in response to cursor navigation, the strategy available to sighted users of scrolling

DOI: 10.14448/jsesd.13.0005

the display to read text elsewhere in the file while leaving the cursor in place is either unavailable or inconvenient in a non-visual setting (Mealin Murphy-Hill, 2012)—a limitation that place markers overcome.

The potential value of this functionality to users who depend on braille or spoken interaction is suggested by the fact that some of these features, particularly document navigation commands, are implemented by screen readers themselves for use in Web browsers and word processors that lack them. In contrast, the most advanced text editors implement the keyboard-based document navigation and editing functions directly, and for all users.

Editor commands are often also available for processing markup-language documents by running external tools, such as TEX engines or conversion programs (e.g., Pandoc). Error messages produced by these tools, reflecting syntax errors in the markup, are typically displayed in a window to which the user can navigate. In some implementations, editor commands are provided to move the cursor to the location in the document corresponding to each error, thereby improving efficiency for keyboard users generally, and for screen reader users especially. The accessibility of editor features designed to assist in the identification and diagnosis of markup errors is crucially important to the user's overall productivity in creating and revising marked up documents non-visually. Thus, it has become an important focus of attention in the development of customized speech-based interfaces for markup editing (Manzoor et al., 2019; Sorge, 2016b).

Interestingly, in the case of LATEX documents, the warnings issued by the TEX engine can also provide insight into typographical problems that could otherwise be discovered only by visual means. For example, if a paragraph cannot be typeset optimally without exceeding the width of the text block and thus, in languages such as English that are written left to right, intruding into the right margin, a warning is issued in the TEX log file to alert the user. Corrective changes can then be made, for example by inserting a discretionary hyphen into a word. Without the availability of the log file, only a discerning sighted reader would be able to report the issue.

Choosing an Appropriate Markup Language

Markup languages differ considerably in their syntax and capabilities. Choosing a tool that is appropriate to a given task and to the needs of the particular user is a clear necessity. The 'light-weight' markup languages, such as Markdown, AsciiDoc, RestructuredText and Emacs Org-mode ("Org Mode: your life in plain text", n.d.), all have the advantage of a concise syntax that makes extensive use of punctuation and symbols, thus enhancing the readability of the source text not only to visual users, but to braille and speech users as well. However, the light-weight languages do not allow for the extensive features and control over presentation that can be gained from a typesetting language such as LATEX, or from the combination of HTML, Cascading Style Sheets (CSS) and JavaScript. Of course, developing proficiency in using a more complex markup language demands a greater investment in learning, whereas the light-weight languages have the advantage that they can be mastered relatively quickly.

This is among the principal reasons for Seo, McCurry, and Team (2019) to propose the use of R Markdown, combined with a custom-developed, simple Web-based editing and format conversion application, for screen reader users who lack the background in programming concepts

DOI: 10.14448/jsesd.13.0005

that would equip them to use more complex markup-based languages and tools. A project with similar features, implemented in Python and packaged for distribution by its authors, is documented in Godfrey and James (2016). A Markdown-based authoring tool intended to assist screen reader users in writing presentation slides while avoiding a WYSIWYG solution is described in Oelen and Auer (2019).

Despite the ease of learning characteristic of light-weight markup languages, it should be noted that tools which transform documents marked up in these languages to presentational formats often do so via a conversion to HTML or LATEX. Hence, knowledge of either of the latter languages is necessary to anyone who wishes to customize the templates or procedures used in such conversions. It follows that learning HTML, LATEX, or both, together with a light-weight markup language, would be advantageous, especially to those whose work involves extensive editing of technical documents.

From the non-visual author's perspective, working with a markup language is quite different from writing a document in a contemporary word processor. In the former case, the entire text and structure of the document, together with any presentational controls, are included as part of the text. None of these components is hidden. In the latter case, presentational attributes such as fonts, spacing, and word processor styles, typically need to be queried via screen reader commands, although a screen reader may provide for a mode in which formatting changes are announced proactively as the document is read. A further beneficial characteristic of the markup-based approach is that mathematical notation is simply included as part of the marked up text, thus avoiding the use of and the potential accessibility issues associated with graphical equation editors. The TEX notation for mathematics is widely supported. It may be used, for example, in Markdown documents, or in HTML documents that invoke MathJax to render mathematical expressions. A second text-based notation for mathematical content is AsciiMath, which is supported, for instance, in the AsciiDoc format.

Although graphical content can be integrated into marked up documents by making reference to vector or rasterized image files created by graphics editors, it is also possible to construct certain types of diagrams by writing code in a domain-specific programming language. The source code gives a precise description of the diagram, which is then rendered as an image when the document is processed. In the LATEX environment, for example, diagrams can be programmed in languages such as TikZ (Tantau, n.d.), Asymptote ("Asymptote: the Vector Graphics Language", n.d.), and Graphviz ("Graphviz", 2021). Specialized packages support specific types of graphics, for example chemical diagrams, flow charts, and electrical circuits. Plots of mathematical functions can be generated by symbolic algebra systems for inclusion in marked up documents. Likewise, the R statistics package, and its accompanying markup language—R Markdown—offer rich graphing capabilities and the ability to embed the output directly in a document. See Baumer and Udwin (2015) for an overview, together with the more detailed expositions in Xie, Allaire, and Grolemund (2018) and Xie (2016).

The availability of such domain-specific languages opens interesting opportunities for the independent, non-visual creation of graphical material. For example, in preparing a recent publication, the author developed conceptual diagrams as directed graphs, implemented in the Graphviz language and refined under the guidance of colleagues. This was inspired by Raman (1997, ac-

DOI: 10.14448/jsesd.13.0005

knowledgements), who acknowledges using the PSTricks graphics package to prepare the figures for his book. The design of methods and tools for enhancing the non-visual construction of diagrams in domain-specific languages remains a largely unexplored area of potential research. Which languages can be most effectively used in a non-visual setting for creating different types of graphical content, and what language features are most desirable in non-visual authoring scenarios, remain important questions that has so far not received sustained attention in the scholarly literature.

An exception is Takagi, Suzuki, and Araki (2020), in which a domain-specific language is described that has been designed to support authors who are blind by reducing the need for the computation of coordinates in specifying diagrams composed of elementary geometric constructs such as lines and plane figures. A dynamic tactile graphics display was introduced to make a graphical rendering of the diagrams thus produced accessible to the user. The authors note the value of domain-specific languages in enabling educators or other professionals who are blind to create precise diagrams for a sighted readership. Practical initiatives to advance this approach further could include the development of libraries for existing graphics languages that facilitate working with diagrams non-visually, as well as tools for producing an accessible rendering of the graphical material, for example as a description or for tactile display. Thus, it is important not only to choose an appropriate markup language for a particular purpose, but also to select, if needed, a language in which to construct the graphical components of the document. Once these choices have been made, the author is well placed to use her or his preferred text editor to create and revise scientific, technical and scholarly documents of arbitrary length and complexity.

Brief Comments on Revision Control and Collaborative Writing

Whether for purposes of individual or collaborative writing projects, the use of revision control tools to keep track of changes to marked up documents is invaluable. Making experimental changes, recovering from mistakes, and comparing different versions of a document are among the tasks that can be completed more easily and efficiently if the text is placed under revision control. If one is working with collaborators, the approval and merging of changes into the primary version of the text can be carried out in a precise and well organized manner by maintaining distinct branches of development. For these reasons, revision control systems created for use in programming projects can be and in practice have been applied to the maintenance of marked up documents as well. In particular, Git is currently a popular tool which is well suited to this application. A useful introduction to Git appears in Straub and Chacon (2014).

Some text editors, such as Emacs, TextMate, and Visual Studio Code implement direct support for working with version control systems, including Git. In the author's experience, these tools generally prove to be accessible, at least for basic functions such as committing changes to a repository. Likewise, the Git command line tool itself is very accessible with a screen reader. The main difficulty, from a non-visual perspective, lies in making sense of the diff patches used to represent the differences between revisions of a file. This task is especially challenging if the changes occur within paragraphs of text, each paragraph is represented as a single line in the source file, and the differences are shown on a line-by-line basis, as is the default. To solve this

DOI: 10.14448/jsesd.13.0005

problem, the author has found it convenient to use the Git –word-diff option, which displays the differences word-by-word, with insertions shown as {+inserted text+}, and deletions as {-deleted text-}. If each paragraph is represented as a single line in the file, the –unified=0 option is often desirable to omit context lines from the output. Git can also be configured to respect the syntax of LATEX markup in determining what constitutes a 'word' in computing the differences between revisions.

In a collaborative setting, it is also typical for comments on the document to be maintained separately from the text, using an issue tracking tool such as that provided by the GitHub or GitLab Web-based repository hosting service. Thus, issues can be created and discussed without modifying the document itself, and thus without introducing potentially distracting comments and responses to them into the source text. This solution may be contrasted with what is offered by contemporary word processors, in which screen reader announcements of comments and replies thereto can be confusing and may distract the user's attention from writing and editing tasks (Das, Piper, Gergle, 2022, § 4).[6]

Although all authors can make changes to the document simultaneously, each participant's modifications are not available to collaborators until they are uploaded to a repository, and conflicts are resolved in a discrete step with the merging of branches. The approaches to collaborative work engendered by revision control systems are thus different from those associated with real-time, collaborative editing systems, in which changes introduced by co-authors are immediately reflected in each contributor's editing environment as they occur. From a non-visual perspective, the cognitive demands of attempting to monitor the changes made by collaborators while concurrently writing and editing a document, which are inherent in co-editing systems, are avoided entirely. The established conventions of committing change sets to a repository as discrete tasks (e.g., adding a section to a document or correcting typographical errors), and of writing an appropriate log message for each commit, simplify the task of identifying and understanding the work of collaborators, often without having to review the exact differences between revisions. It is the author's experience that, together, the characteristics and practices surrounding revision control tools can enhance collaboration without imposing a cognitive burden on a non-visual contributor's editing activity, as co-editing systems do.

Research investigating the accessibility issues raised by real-time co-editing systems has focused largely on word processors rather than on markup-based text editors. To date, the complications introduced by scientific and technical writing in this connection do not appear to have been explored. It is nevertheless clear that, for a multiplicity of practical reasons, real-time co-editing is often challenging and can impose significant cognitive demands upon authors who are blind, notwithstanding the accessibility-related features of current screen readers and word processors (Das, Gergle, Piper, 2019; Das, Piper, Gergle, 2022, § 4; Das, McHugh, Piper, Gergle, 2022, § 4). As the use of online, collaborative editors becomes increasingly widespread in scientific, technical and scholarly writing projects, this lack of accessibility threatens to pose growing barriers to participation in tasks arising in education and in the workplace. The accessibility of real-time, collaborative, markup editors is thus a topic to which further research and de-

[6] In Das, Piper, and Gergle (2022), the authors investigate alternative techniques for enhancing the non-visual user interface to reduce these difficulties in the word processor environment.

DOI: 10.14448/jsesd.13.0005

velopment effort could valuably be directed. An interesting potential approach is that of heterogeneous editing, in which different editors—for example, Emacs and Vim—are used by different authors in the real-time, cooperative interaction (Cho, Sun, Ng, 2019). It is reasonable to predict that authors—whether or not they have accessibility-related needs—will continue to differ in their editor preferences, and for a multiplicity of reasons, only some of which are related to disability and support for assistive technologies. Given this condition, it would be advantageous for accessible, collaborative editing solutions to be developed that do not require the same editor to be used by all parties in order for them to cooperate in an interactive authoring session.

Converting Marked Up Documents for Presentation to Readers

Having considered the use of markup languages exclusively from the author's perspective in the preceding section, the discussion now switches to the reader's perspective by focusing on the non-visual accessibility of marked up documents as they are ultimately presented. As has been made clear, what is delivered to readers is typically produced by applying document processing or conversion tools, for example a typesetting program such as TEX, or a file format converter such as Pandoc or AsciiDoctor. Indeed, it is customary for the author to review such output repeatedly in the course of creating and revising a document. Hence, the accessibility of the output is important to the author as well as to readers in general. In an alternative scenario, the markup language can be used to prepare study materials or other documents for reading by individuals with print disabilities, for example by an educational institution for its students (Murillo-Morales, Miesenberger, Ruemer, 2016; Voegler, Bornschein, Weber, 2014). The accessibility-related characteristics of the output produced obviously depend on both the relevant features of the markup language and the nature of the conversion process employed. Although the various combinations of languages and tools cannot be reviewed in detail here, general observations can be made that may nonetheless prove useful.

Of the two most common output formats—HTML, and Portable Document Format (PDF)—it is the author's experience that the former is considerably more accessible in practice than the latter. As the primary format of the World Wide Web, HTML is well supported by browsers and assistive technologies across all widely used desktop and mobile operating systems, ensuring a high degree of accessibility, as long as appropriate practices and standards are followed. The principal Web standard specifying accessibility-related requirements for HTML documents, including those generated by markup language conversion tools, is presently Web Content Accessibility Guidelines (WCAG) 2.1 (World Wide Web Consortium, 2018b). Even fully automated document processing tools could enhance support for accessibility by implementing relevant aspects of Authoring Tool Accessibility Guidelines (ATAG) 2.0 (World Wide Web Consortium, 2015). Although PDF standards have long provided for tagging, via the use of a structure tree that captures structural components of the document,7 this and associated accessibility-supportive features have not been fully and widely implemented, in a variety of operating systems, either in PDF producing applications, or in document reading software. The effect of these limitations on screen reader users varies depending on the complexity of the document and on the software involved in both production and reading. It can entail a lack of support for structural navigation in PDF documents, an in-

DOI: 10.14448/jsesd.13.0005

correct reading order of text, or the inability to obtain alternative text for images or comprehensible mathematical notation altogether. The cumulative consequences may readily be disastrous for non-visual access to scientific and technical documents presented in PDF format (Polsley, Lacy, Hammond, 2021). Although some PDF reading applications attempt to recognize headings or other structural aspects of the document while processing an untagged PDF file, the results of this analysis may be quite inaccurate. In the absence of alternative text for images and without an adequate encoding of the structure and content of mathematical notation, untagged PDF files have substantial accessibility-related limitations that typically render diagrams and equations incomprehensible to the braille or speech user.

By contrast, the accessibility-related features of HTML and related standards find support not only in Web browsers and screen readers, but also in the conversion tools used to process documents written in various markup languages. Although the extent of this support varies according to the implementation, one can typically produce alternative text for images, and convert mathematical notation to the Mathematical Markup Language (MathML), either directly or via the inclusion of a MathJax script in the resulting HTML document. As has been elaborated in greater detail elsewhere (Soiffer Noble, 2019; White, 2020), MathML is not only the standard for representing mathematical notation on the Web; it is also the only representation that has been implemented by screen readers, allowing for braille and spoken rendering of the notation as well as for interactive, structural navigation and reading. In addition, HTML content can be processed by proprietary or open-source braille translation software to produce embossed braille versions of a document, although the extent of support for mathematical content varies among the available translation tools. These considerations place HTML as the best supported output format to which documents can be converted in a manner that preserves their accessibility-related characteristics. Since the EPUB digital publishing format (World Wide Web Consortium, 2018a) is based on HTML and CSS, and permits the inclusion of mathematics in Presentation MathML form, it shares the accessibility-related advantages of HTML described here.

Many of the features needed for non-visual accessibility can also be preserved by a conversion to a word processor format—in particular, Office Open XML, or Open Document Format. However, the extent to which structural navigation of the document and braille or speech-based reading of mathematical expressions are available is contingent on the word processor and the screen reader used. For example, Microsoft Word in the Windows environment, together with a suitable screen reader, offers all of these capabilities, but other word processor and screen reader combinations exhibit limitations that differ among implementations, including versions of the same word processor built for different operating systems. Thus, word processor formats are best suited to circumstances in which further editing of the document by its recipients is expected, rather than as a delivery format for general reading.

In general, document conversion tools differ not just in their capabilities, but also in the nature and extent of the customizations that can be introduced to tailor the output to meet specific needs and preferences. The Pandoc processor, for example, which can convert between a range of file formats, allows for the creation of user-supplied document templates, and for the writing of filters—programs that manipulate the in-

DOI: 10.14448/jsesd.13.0005

termediate, abstract representation of the document which it uses internally. The accessibility of the output is thus a product of the content of the source document (e.g., whether it is well structured through proper use of markup, or whether alternative text for images is supplied by the author), the capabilities of the conversion tool, and the influence of any customizations which are in effect. In the hands of the ultimate recipient, of course, the capabilities of the Web browser or other reading software, the assistive technology, and the user's knowledge and skills, all contribute to the level of understanding which is achieved and hence to the performance of tasks associated with using the document.

The conversion of LATEX documents to other markup languages such as HTML is particularly complex, as the underlying TEX typesetting system amounts to a specialized programming language. Modern TEX engines produce output in PDF format by default, although the older device-independent (DVI) format remains supported. Conversion of LATEX documents to HTML can be achieved by a range of tools. A particularly successful approach is to use a TEX engine as part of the process, enabling the tool to execute TEX code and more easily to support a larger subset of the many LATEX packages that may be used by authors. Current examples of this approach are TeX4ht (TEX Users Group, 2021) and Lwarp (Dunn, 2021). An alternative strategy is to implement desired aspects of the TEXmacro language in an interpreter designed for document conversion, as in LaTeXML (Miller Ginev, n.d.), which is written in Perl 5. Together, these tools presently offer the best freely and publicly available routes for producing HTML or word processor files from LATEX source documents. Two qualifications should be noted, however. First, the results of the conversion are improved if the source document relies only on LATEX packages which are supported by the conversion tool. Second, enhancements in the accessibility of the output may be achieved by customizing the conversion process, using features of the chosen tool as detailed in its documentation. Since the PDF files produced by LATEX by default are not tagged, and hence provide limited accessibility, non-visual writing and revision of LATEX documents can be greatly facilitated by converting them to an HTML format for review using one of the tools already noted. Indeed, the process of generating HTML and PDF output in parallel can be easily automated by means of a simple 'make' file or shell script.

Recent Developments in Document Conversion

Three encouraging developments related to the conversion of marked up documents to accessible formats suggest areas in which further progress can be expected in the coming years. Together, these initiatives promise considerable improvement in the non-visual accessibility obtainable, by fully automated means, from properly marked up source files.

Currently, it is not feasible to generate tagged PDF from LATEX documents without making use of experimental packages that sometimes require considerable, manual interventions in the source text. See Moore (2020) for an overview of the interventions needed to tag a technical report written in LATEX using a current experimental package. Hagen (2010) describes an implementation of basic support for tagged PDF in ConTEXT, another TEX-based markup language that is less widely used than LATEX. However, the core developers of LATEX have recently commenced a funded project with the purpose of progressively implementing support for tagged PDF in LATEX itself (Mittelbach

DOI: 10.14448/jsesd.13.0005

Rowley, 2020). Ultimately, it is hoped that this functionality will enable the automated creation of PDF files that satisfy accessibility standards, including coverage of the images and mathematical notation occurring in scientific and technical texts.[7] By integrating tagging into the core of LATEX, it should also be possible to extend this functionality to many of the LATEX packages that have been developed by the open-source community, and which are available in software distributions, such as TEXLive, for use by authors. This project also has the potential to lead to the creation of common software infrastructure for converting LATEX to other markup languages, principally HTML, yielding more reliable conversion processes as well as more effective and maintainable support for a larger set of LATEX packages than current tools accommodate. Indeed, an algorithm for converting tagged PDF to HTML has already been standardized (PDF Association, 2019).[8] It could be adopted, perhaps with further refinements to enhance accessibility, as a common approach to producing HTML or EPUB documents from LATEX sources. A further benefit of this work noted by Mittelbach and Rowley (2020) is that a large class of existing documents could be made considerably more accessible by simply rebuilding them with updated tools to produce tagged PDF and HTML versions. If this strategy were to become practicable, it would greatly enhance access to a potentially large academic and scholarly literature written in LATEX. Meanwhile, the best pragmatic response by students and researchers who are blind who wish to access existing LATEX documents is to obtain the original source files from the authors or publishers, and then to read them directly in a text editor or using any of the available document conversion tools.

A second, recent initiative concerns the addition of support for braille translation and formatting to conversion tools developed for the PreTeXt ("PreTeXt", n.d.) markup language. PreTeXt is an XML-based markup language optimized for writing and editing directly by authors, and originally created for the production of mathematical texts. The accompanying software can convert PreText documents to output formats including HTML, and PDF via LATEX. Using a process based on XSLT (World Wide Web Consortium, 2017), and the Liblouis braille translator and formatter (Liblouis, n.d.), code for converting PreTeXt documents to embossed braille has been integrated as a generally available feature of the software. Mathematical notation is translated to Nemeth Code braille via Speech Rule Engine (Sorge, 2022; Sorge, Chen, Raman, Tseng, 2014), a specialized library for the production of accessible spoken, and now also braille, mathematics.

A third encouraging development signals a move beyond hand-crafted alternative text as the primary means of making graphical content accessible on the Web and in electronic documents generally. Packages developed for the R statistical programming environment automate the creation of data representations which are non-visually accessible. These formats are of three kinds (Seo, 2021): first, algorithmically generated, static or interactively readable and

[7] Alternative methods of representing mathematical notation for purposes of enabling non-visual access in PDF files generated from LATEX sources are explored in Moore (2014). Ahmetovic et al. (2018) describe a LATEX package that inserts the TEX source of mathematical expressions as alternative text in the generated PDF file, but without tagging document structures. A dictionary is provided to improve the spoken presentation of the notation.

[8] The author gratefully acknowledges Ross Moore, in a discussion on the TEX Users Group's 'accessibility' mailing list, for referring to this specification and pointing out its potential.

DOI: 10.14448/jsesd.13.0005

navigable descriptions of basic chart types;[9] second, sonification of charts (Siegert Williams, 2017); and third, creation of PDF files suitable for the production of tactile graphics using swell touch paper (Seo, 2020). These solutions are founded on the understanding that, though useful, plain text descriptions of graphics are not sufficient for independent, non-visual analysis and exploration of data (Fitzpatrick et al., 2017); they need to be complemented by efficient auditory and tactile representations, or by representations of data that can be traversed and read by interactive means. Fully automatic generation of these accessible forms from the data themselves support the independence of students and professionals in engaging in statistics-related work—an advantage that would not be achieved by approaches to accessibility requiring manual intervention by a sighted person, such as writing descriptions or drawing tactile graphics by hand. Since the original data can be preserved in R Markdown documents, it becomes possible to produce graphical and non-visual representations of the data in parallel, leading to the construction of highly accessible scientific texts. Examples are provided in Godfrey (2021, chapter 7), and in Seo (2021).

Conclusion

Markup languages, and the software available for editing and processing them, have features that can make them convenient for the non-visual creation and revision of scientific, technical, or scholarly documents. The newer, lightweight markup languages have advantages in source text readability and ease of learning. However, they typically lack the features, extensibility or flexibility of their more complex alternatives. Similarly, text editors that offer greater efficiency and more features also tend to demand a larger investment in learning, for example in understanding relevant concepts and in acquiring proficiency in the application of an extensive set of keyboard commands. The tradition, established most prominently by the UNIX operating system and its derivatives, that separates the choice of a text editor from the selection of a markup language, and which prioritizes using plain text files to store content, has proven to be of lasting value in enabling the matching of available tools with the needs of the individual user.

The capacity to enhance and extend certain text editors by independently created software packages has opened opportunities to achieve a quality of non-visual access that screen readers have often not afforded. Emacspeak is the most enduring, and appears to be the earliest example of this approach. However, it is the author's observation, based on discussions that have taken place via Internet mailing lists during almost three decades, that few, if any, of the editor-related research projects intended to improve non-visual access have gained a sustainable community of users or developers. Instead, the software seldom progresses beyond the confines of the project in which it was envisioned. This may be a reasonable outcome if the intent is purely to conduct research and to publish findings, which are worthy aims in themselves, but it also limits the benefits of the work to the people whose needs it is supposed to meet. Projects with ambitions beyond the publication of research, in the author's view, need a strategy for sustaining development, for example by publication as an extension package in distribution repositories associated with an established

[9] Use of the BrailleR package (Godfrey, n.d.), which provides this capability, is documented in detail by its author in Godfrey (2021). Work undertaken to extend the BrailleR package to build interactive, Web-based diagrams supporting non-visual reading and exploration with a screen reader is reported in Fitzpatrick, Godfrey, and Sorge (2017).

DOI: 10.14448/jsesd.13.0005

text editor, or by integration as new features or accessibility-related enhancements of the editor itself that can be maintained over the long term. Encouraging emergence of a community of users is also desirable. Further text editor-related research is justified, particularly in regard to the accessibility of real-time, collaborative editing systems.

Recent efforts further to improve the non-visual accessibility of the output producible from marked up documents are encouraging. The inclusion of support for tagged PDF in the core code of LATEX promises significantly to enhance the accessibility of new and existing documents prepared in this format, not only, or perhaps even primarily, via the direct consumption of the PDF output, but by improving the quality and reliability of conversions to other formats, especially those based on HTML. The integration of mechanisms for braille production directly into tools supporting a mathematically-oriented markup language is useful in itself, while suggesting a strategy of development that could profitably be pursued further.

Generation of non-visually accessible charts via the R statistics environment also illustrates the significant benefits obtainable from small but strategic software projects, engaging developers who are also users of these accessibility-related tools. Further research and development work could usefully explore the potential of other graphics-oriented programming languages, especially those used in the construction of diagrams for inclusion in marked up documents, not only to produce accessible non-visual output, but also to be applied by authors who are blind to the task of producing graphics for reading by educators and colleagues. Here also, a case can be made for extending existing languages and tools to improve non-visual access, rather than for designing new graphics languages intended primarily to be used by people with disabilities, which would be isolated from the evolution of more widely used alternatives and thus unable to benefit from their larger communities of users and developers.

It is here hypothesized that domain-specific languages which describe diagrams in terms of the objects, properties and relations represented, rather than in purely geometric terms, can capture semantic distinctions that should prove valuable in automatically generating non-visual representations adapted to be comprehensible to the person interpreting them. A further illustration of this strategy for making diagrams accessible appears in Sorge (2016a) and Sorge, Lee, and Wilkinson (2015), in which Chemical Markup Language (CML) is used as an intermediate representation of diagrams which are subsequently rendered as interactive, graphical objects that can be magnified, or read and navigated via a screen reader. More generally therefore, domain-specific languages have potential both as a medium of expression for authors in preparing marked up documents non-visually, and as source formats from which highly usable representations may be derived by means of sonification, tactile graphics, or text-based reading and interaction in speech or braille.

Acknowledgments

The author is indebted to colleagues, friends, e-mail correspondents, and participants in online fora who have influenced his views over the years on the issues presented here. Naturally, full responsibility for the opinions developed in the paper rests entirely with him. Particular gratitude is owed to T.V. Raman for his insights, and for valuable discussion of Emacspeak as well as pointing out the significance of domain-specific languages for non-visual drawing of diagrams. Volker Sorge helpfully noted

the relevance of his work on chemical diagrams to the topic of the paper. Participants in the 'BlindMath' mailing list have discussed LATEX, Markdown, R, the accessibility benefits of converting documents to HTML format, and related topics on various occasions, providing valuable information and ideas. Mark Hakkinen and Heather Buzick (Educational Testing Service), and Clayton Lewis (University of Colorado Boulder) reviewed the manuscript and provided insightful comments.

DOI: 10.14448/jsesd.13.0005

References

Ahmetovic, D., Armano, T., Bernareggi, C., Berra, M., Capietto, A., Coriasco, S., ... Taranto, E. (2018). Axessibility: A latex package for mathematical formulae accessibility in pdf documents. In Proceedings of the 20th international acm sigaccess conference on computers and accessibility (pp. 352–354).

Ahmetovic, D., Bernareggi, C., Bracco, M., Murru, N., Armano, T., Capietto, A. (2021). Latex as an inclusive accessibility instrument for highschool mathematical education. In Proceedings of the 18th international web for all conference (pp. 1–9).

Allen, D., White, S., individual AsciiDoctor contributors. (2021). Asciidoc language documentation. Retrieved December 8, 2021, from https://docs.asciidoctor.org/asciidoc/latest/

Asymptote: the Vector Graphics Language. (n.d.). Retrieved January 4, 2022, from https : / / asymptote . sourceforge.io

AUCTeX. (2020). Retrieved December 8, 2021, from https://www.gnu.org/software/auctex/manual/auctex/

Bahls, P., Wray, A. (2015). Latexnics: The effect of specialized typesetting software on stem students' composition processes. Computers and Composition, 37, 104–116.

Baumer, B., Udwin, D. (2015). R markdown. Wiley Interdisciplinary Reviews: Computational Statistics, 7 (3), 167–177.

Cameron, D., Elliott, J., Raymond, M. L. E. S., Rosenblatt, B. (2009). Learning gnu emacs, 3rd edition. O'Reilly Media, Inc.

Cho, B., Sun, C., Ng, A. (2019). Issues and experiences in building heterogeneous co-editing systems. Proceedings of the ACM on Human-Computer Interaction, 3, 1–28.

Das, M., Gergle, D., Piper, A. M. (2019). " it doesn't win you friends" understanding accessibility in collaborative writing for people with vision impairments. Proceedings of the ACM on Human-Computer Interaction, 3(CSCW), 1–26.

Das, M., McHugh, T. B., Piper, A. M., Gergle, D. (2022). Co11ab: Augmenting accessibility in synchronous collaborative writing for people with vision impairments. Retrieved January 14, 2022, from https: //maitraye.github.io/files/papers/BVI$_C$W$_C$HI22.*pdf*

Das, M., Piper, A. M., Gergle, D. (2022). Design and evaluation of accessible collaborative writing techniques for people with vision impairments. ACM Transactions on Computer-Human Interaction, 29 (2), 1–42.

Dunn, B. (2021). The lwarp package: Latex to html. Retrieved from http://mirrors.ctan.org /macros/latex/ contrib/lwarp/lwarp.pdf

Fitzpatrick, D., Godfrey, A. J. R., Sorge, V. (2017). Producing accessible statistics diagrams in r. In Proceedings of the 14th international web for all conference (pp. 1–4).

Flynn, P. (2021). Formatting information: A beginner's introduction to typesetting with latex. Retrieved December 8, 2021, from http://latex.silmaril.ie/formattinginformation/

Free Software Foundation, Inc. (2021). The emacs editor. Retrieved December 8, 2021, from https://www. gnu.org/software/emacs/manual/html$_n$*ode*/*emacs*/

Godfrey, A. J. R. [A Jonathan R], James, M. C. (2016). Simple authoring of statistical analyses by and for blind people. In Proceedings of the international workshop on digitization and e-inclusion in mathematics

DOI: 10.14448/jsesd.13.0005

and science 2016 (pp. 47–54).

Godfrey, A. J. R. [A. Jonathan R.]. (n.d.). Brailler: Improved access for blind users. massey university. r package version 0.32.1. Retrieved January 24, 2022, from https://cran.r-project.org/package=BrailleR

Godfrey, A. J. R. [A. Jonathan R.]. (2021). Brailler in action. Retrieved January 12, 2022, from https://r-resources.massey.ac.nz/BrailleRInAction/

Graphviz. (2021). Retrieved January 4, 2022, from https://graphviz.org

Grätzer, G. (2016). More math into latex (5th). Springer, CHAM.

Hagen, H. (2010). Tagged pdf in context. TUGboat, 31(3), 197–202.

Ho, D. (n.d.). Notepad++. Retrieved January 13, 2021, from https://notepad-plus-plus.org/

International Organization for Standardization. (2008). Iso 32000-1:2008—document management—portable document format—part 1: Pdf 1.7.

International Organization for Standardization and International Electrotechnical Commission. (2015). Iso/iec 26300-1:2015, information technology—open document format for office applications (open- document) v1.2.

International Organization for Standardization and International Electrotechnical Commission. (2016). Iso/iec 29500-1:2016: Information technology—document description and processing languages— office open xml file formats—part 1: Fundamentals and markup language reference.

Jones, R. (2021). A restructured text primer. Retrieved December 8, 2021, from https://docutils.sourceforge.io/docs/user/rst/quickstart.html

Knauff, M., Nejasmic, J. (2014). An efficiency comparison of document preparation systems used in academic research and development. PloS one, 9 (12), e115069.

Kottwitz, S. (2021). Latex beginner's guide (2nd). Packt Publishing Ltd.

Liblouis. (n.d.). Liblouis—an open-source braille translator and back-translator. Retrieved January 11, 2021, from http://liblouis.org

MacFarlane, J. (2021). Pandoc user's guide. Retrieved December 8, 2021, from https://pandoc.org/MANUAL. html

MacroMates Ltd. (2021). Textmate for macos. Retrieved January 13, 2022, from https://macromates.com/

Mailund, T. (2019). Introducing markdown and pandoc: Using markup language and document converter. Apress.

Manzoor, A., Arooj, S., Zulfiqar, S., Parvez, M., Shahid, S., Karim, A. (2019). Alap: Accessible latex based mathematical document authoring and presentation. In Proceedings of the 2019 chi conference on human factors in computing systems (pp. 1–12).

Matt. (n.d.). Semantic line breaks. Retrieved December 29, 2021, from https://sembr.org/

McDonnell, M. (2014). Pro vim. Apress.

Mealin, S., Murphy-Hill, E. (2012). An exploratory study of blind software developers. In 2012 ieee symposium on visual languages and human-centric computing (vl/hcc) (pp. 71–74). IEEE.

Microsoft Corporation. (2021). Accessibility in visual studio code. Retrieved January 18, 2021, from https://code.visualstudio.com/docs/editor/accessibility

DOI: 10.14448/jsesd.13.0005

Miller, B., Ginev, D. (n.d.). Latexml a latex to xml/html/mathml converter. Retrieved April 22, 2022, from https://math.nist.gov/BMiller/LaTeXML/

Mittelbach, F., Rowley, C. (2020). Latex tagged pdf—a blueprint for a large project. TUGboat, 41(3), 292–298.

Moore, R. (2014). Pdf/a-3u as an archival format for accessible mathematics. In International conference on intelligent computer mathematics (pp. 184–199). Springer.

Moore, R. (2020). Tagging with latex—part 1: Author considerations. TUGboat, 41(2), 223–242.

Moorhead, A. (2020). Is latex use correlated with the number of equations in a manuscript? In Aas/division of dynamical astronomy meeting (Vol. 52, pp. 103–07).

Murillo-Morales, T., Miesenberger, K., Ruemer, R. (2016). A latex to braille conversion tool for creating accessible schoolbooks in austria. In International conference on computers helping people with special needs (pp. 397–400). Springer.

Oelen, A., Auer, S. (2019). Content authoring with markdown for visually impaired and blind users. In 2019 ieee international symposium on multimedia (ism) (pp. 285–290). IEEE.

Org Mode: your life in plain text. (n.d.). Retrieved January 13, 2022, from https://orgmode.org/

PDF Association. (2019). Deriving html from pdf: A usage specification for tagged iso 32000-2 files. Retrieved January 6, 2022, from https://www.pdfa.org/wp-content/uploads/2019/06/Deriving$_H$TML$_f$rom$_P$DF.pdf

Polsley, S., Lacy, A., Hammond, T. (2021). Are best practices best? making technical pdfs more accessible. Retrieved January 4, 2021, from https://www.w3.org/WAI/about/projects/wai-coop/paper102.html

PreTeXt. (n.d.). Retrieved January 6, 2022, from https://pretextbook.org

Raman, T. (1997). Auditory user interfaces: Toward the speaking computer. Kluwer Academic Publishers.

Raman, T. (2021). Emacspeak: The complete audio desktop. Retrieved January 18, 2022, from https://github.com/tvraman/emacspeak

Robbins, A., Hannah, E. (2021). Learning the vi and vim editors (8th). O'Reilly Media, Inc.

Seo, J. (2020). Tactiler: R package for creating tactile graphics for users with visual impairments. Retrieved January 24, 2022, from https://github.com/jooyoungseo/tactileRSeo, J. (2021). Accessible data science for the blind using r. Retrieved January 10, 2022, from https : / /jooyoungseo.com/post/ds4blind/

Seo, J., McCurry, S., Team, A. (2019). Latex is not easy: Creating accessible scientific documents with r markdown. Journal on Technology and Persons with Disabilities, 7, 157–171.

Shotts, W. (2019). The linux command line: A complete introduction (2nd). No Starch Press.

Siegert, S., Williams, R. (2017). Sonify: Data sonification - turning data into sound. r package version 0.0-1. Retrieved January 24, 2022, from https://cran.r-project.org/web/packages/sonify/

Soiffer, N., Noble, S. (2019). Mathematics

DOI: 10.14448/jsesd.13.0005

and statistics. In S. Harper Y. Yesilada (Eds.), Web accessibility—a foundation for research (pp. 417–443). Springer.

Sorge, V. (2016a). Polyfilling accessible chemistry diagrams. In International conference on computers helping people with special needs (pp. 43–50). Springer.

Sorge, V. (2016b). Supporting visual impaired learners in editing mathematics. In Proceedings of the 18th international acm sigaccess conference on computers and accessibility (pp. 323–324). Sorge, V. (2022).

Speech rule engine. Retrieved January 25, 2022, from https://github.com/Speech-Rule-Engine/speech-rule-engine

Sorge, V., Chen, C., Raman, T., Tseng, D. (2014). Towards making mathematics a first class citizen in general screen readers. In Proceedings of the 11th web for all conference (p. 40). ACM. Sorge, V., Lee, M., Wilkinson, S. (2015).

End-to-end solution for accessible chemical diagrams. In Proceedings of the 12th international web for all conference (pp. 1–10).

Sotomayor-Beltran, C., Barriales, A. L. F., Lara-Herrera, J. (2021). Work in progress: The impact of using latex for academic writing: A peruvian engineering students' perspective. In 2021 ieee world conferenceon engineering education (edunine) (pp. 1–4). IEEE.

Straub, B., Chacon, S. (2014). Pro git (2nd). Apress.

Takagi, N., Suzuki, T., Araki, T. (2020). Development of a drawing assistant system for blind users using an object-oriented graphic description language. In 2020 international conference on machine learning and cybernetics (icmlc) (pp. 88–93). IEEE.

Tantau, T. (n.d.). Tikz pdf manual for version 3.1.8b. Retrieved January 4, 2022, from https://mirrors.rit.edu/CTAN/graphics/pgf/base/doc/pgfmanual.pdf

TeX Users Group. (n.d.). Mactex. Retrieved January 13, 2022, from https://www.tug.org/mactex/

TEX Users Group. (2021). Tex4ht. Retrieved January 6, 2022, from https://tug.org/tex4ht/

VimTeX. (n.d.). Retrieved December 30, 2021, from https://github.com/lervag/vimtex

Voegler, J., Bornschein, J., Weber, G. (2014). Markdown–a simple syntax for transcription of accessible study materials. In International conference on computers for handicapped persons (pp. 545–548). Springer.

Walsh, N., Hamilton, R. L. (2010). Docbook 5: The definitive guide. O'Reilly Media, Inc.

Web Hypertext Application Technology Working Group. (2021). Html: Living standard. Retrieved December 8, 2021, from https://html.spec.whatwg.org/

White, J. (2020). The accessibility of mathematical notation on the web and beyond. Journal of science Education for Students with disabilities, 23(1).

Word wrap should not be disabled when accessibility is turned on 95428. (2021). Retrieved January 24, 2022, from https://github.com/microsoft/vscode/issues/95428

World Wide Web Consortium. (2015). Authoring tool accessibility guidelines (atag) 2.0. Retrieved from https://www.w3.org/TR/ATAG20/

World Wide Web Consortium. (2017). Xsl transformations (xslt) version 3.0. Retrieved April 20, 2020, from https://www.w3.org/TR/

DOI: 10.14448/jsesd.13.0005

xslt-30/

World Wide Web Consortium. (2018a). Epub 3.2. Retrieved April 20, 2020, from https://www.w3.org/ publishing/epub3/epub-spec.html

World Wide Web Consortium. (2018b). Web content accessibility guidelines (wcag) 2.1. Retrieved from https://www.w3.org/TR/ 2018/REC-WCAG21-20180605/

Wright, C. H. (2010). Technical writing tools for engineers and scientists. Computing in Science Engineering, 12(5), 98–103.

Xie, Y. (2016). Bookdown: Authoring books and technical documents with r markdown. CRC Press.

Xie, Y., Allaire, J. J., Grolemund, G. (2018). R markdown: The definitive guide. CRC Press.

Zu Bexten, E. M., Jung, M. (2002). Latex at the university of applied sciences giessen-friedberg—experiences at the institute for visually impaired students. In International conference on computers for handicapped persons (pp. 508–509). Springer.

DOI: 10.14448/jsesd.13.0005