May 2022

Command-line Interface with Improved Screen Reader Accessibility

David J. Murphy

Szu Yu Huang

Lucas Radaelli

Amy Hu

Command-line Interface with Improved Screen Reader Accessibility

ABSTRACT

The output of a command-line interface (CLI) is typically unstructured text. When such CLI output is piped to an assistive technology such as a screen-reader, the text is vocalized in a character-by-character manner. The result is often a vocalization that includes irrelevant or repetitive content; vocalization of content in a confusing order; vocalization of tabular or graph-based content in a non-contextual manner; etc. This disclosure describes techniques to provide a structured-data endpoint on a CLI and to enable a screen-reader to interface with and vocalize such structured data via a standard, web-based client. The described techniques make command-line tooling more inclusive and lower barriers to and improve usability by users of assistive technology. The techniques also lower barriers to contribution by those developers of assistive technology who are also users of assistive technology.

KEYWORDS

- Command-line interface (CLI)
- Accessibility
- Accessible rich internet applications (ARIA)
- Hypertext markup language (HTML)
- Structured data
- Screen reader
- Assistive technology
- Vision impairment
- Command prompt
- Command-line tool
- Web rotors
- User interface (UI)
- JSON
- Tongue-controlled device
- Voice command

BACKGROUND

A screen-reader assists vision-impaired users (or users with learning/reading disabilities) by rendering as speech text and image content on a screen. The output of a command-line interface (CLI) or other text input/output environment is typically unstructured. This leads to screen-readers simply reading text in a character-by-character manner from the terminal and providing vocalized output that includes content in a confusing order, tabular or graph-based content in a non-contextual manner, or irrelevant content.

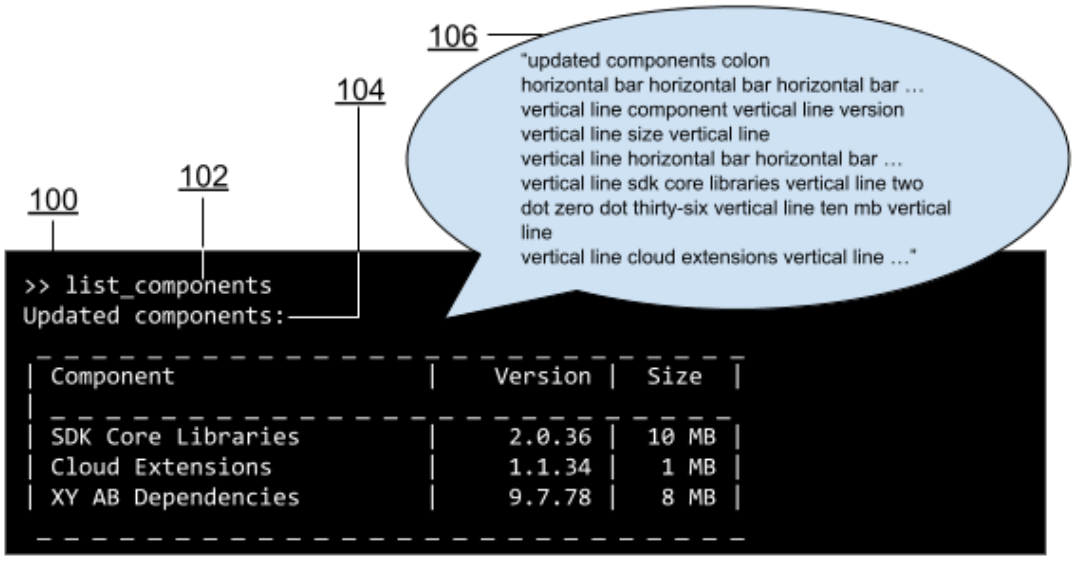
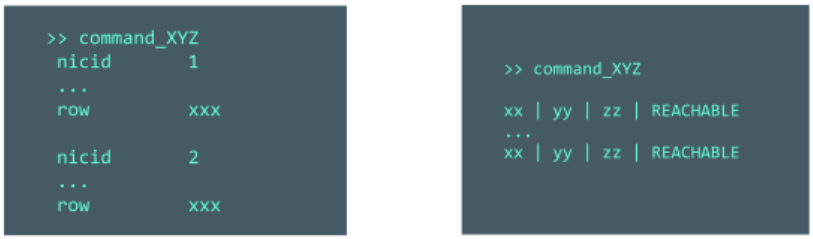


Fig. 1: Command-line interface and associated screen-reader vocalization

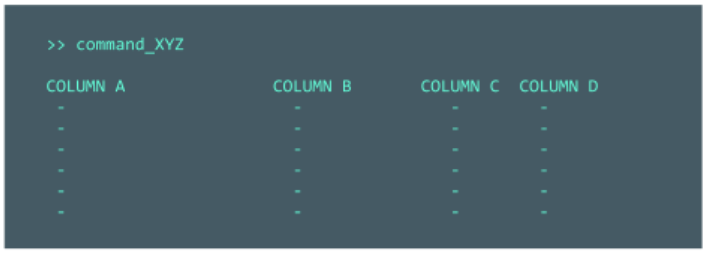
Fig. 1 illustrates a command-line interface (100). In response to a command (102, '*list_components*') provided at a command-line, output (104) is generated that is visually a table due to the use of dashes and the | character - formatted as unstructured text. Rather than read it out as a table, a screen-reader vocalizes the output character by character (106), giving rise to repeated, irrelevant invocations ('*horizontal bar*,' '*vertical line*') of characters that are included as mere visual separators. A user who hears the screen-reader does not know the row and column identities currently being vocalized. For example, a straightforward recitation '*sdk core libraries*

vertical line two dot zero dot thirty-six vertical line ten mb' gives no clue that '*sdk core libraries*' is the first row of type component; that '*two dot zero dot thirty-six*' indicates version; that '*ten mb*' indicates the size of the component '*sdk core libraries*.' Further, it is infeasible for such a user to navigate between table entries or browse command histories based on lines vocalized by the screen-reader.

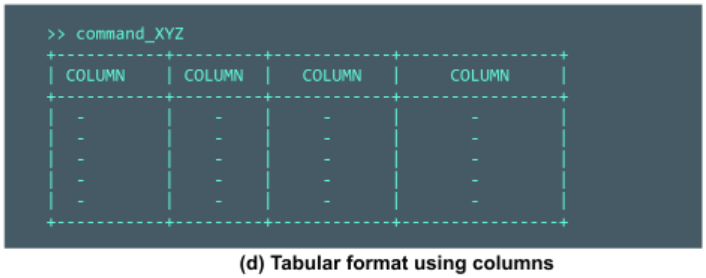
(a) Tabular format using key-value pairs



(b) Tabular format using rows

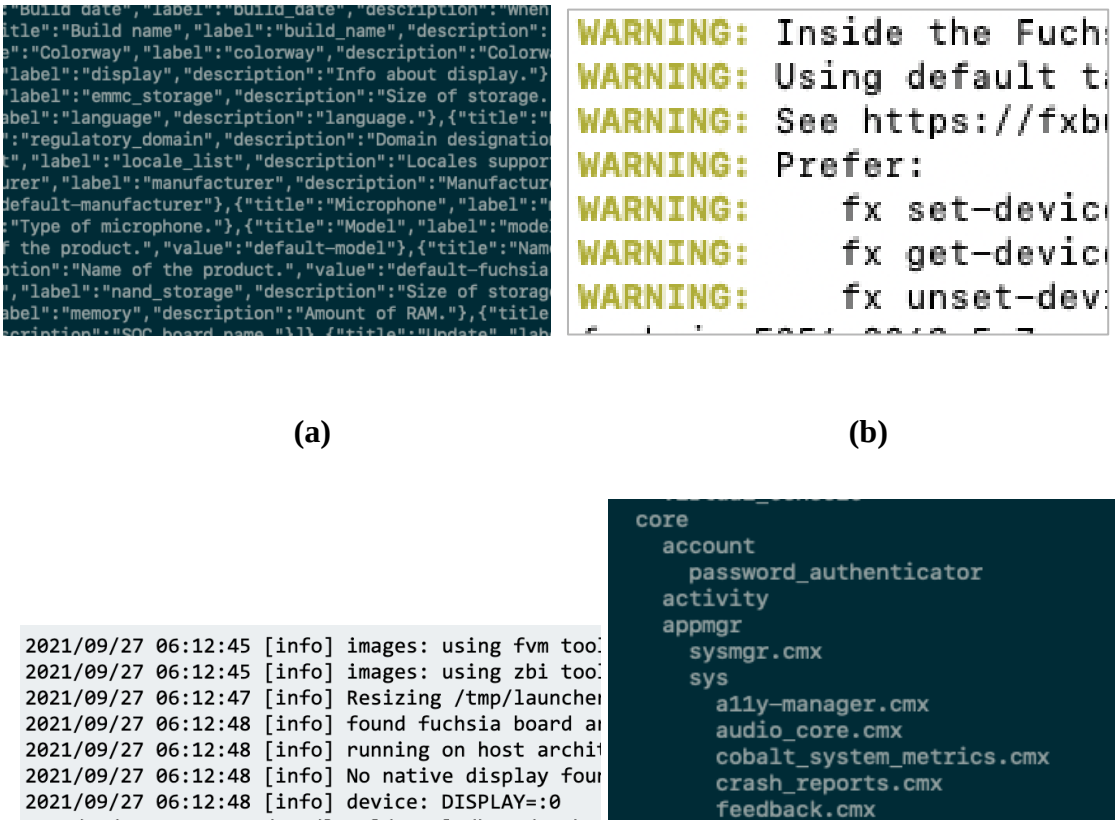


(c) Tabular format using columns



(d) Tabular format using columns

Fig. 2: Tabular data presented in different formats



(a) **(b)**

(c) **(d)**

Fig. 3: Features of CLIs that make them inaccessible

Additional challenges with making command-line interfaces accessible include:

- *Lack of semantics*: The terminal has nowhere to put semantics that a screen-reader can use.
- *Long outputs*: Illustrated in Fig. 3(a), CLI outputs can be too long for a screen-reader to be useful.
- *Syntax coloring*: Illustrated in Fig. 3(b), CLI outputs often use color, which assists sighted users, but is uninterpretable or cumbersome to be interpreted by screen-readers.
- *Redundant information*: Illustrated in Fig. 3(c), CLIs can output redundant information such as timestamp, Unicode, version, etc., which sighted users know to ignore but screen-reader users have to endure, adding to the screen-reader user's cognitive load.

- *Meaning embedded in whitespace*: Illustrated in Fig. 3(d), CLI output can embed meaning in whitespace, e.g., via indentation, which is uninterpretable by a screen-reader.
- *Command history is line-by-line*: Sighted users can easily obtain command history, e.g., by pressing the up-arrow key, but the lack of structure in command history doesn't lend itself for use by screen-reader users.

DESCRIPTION

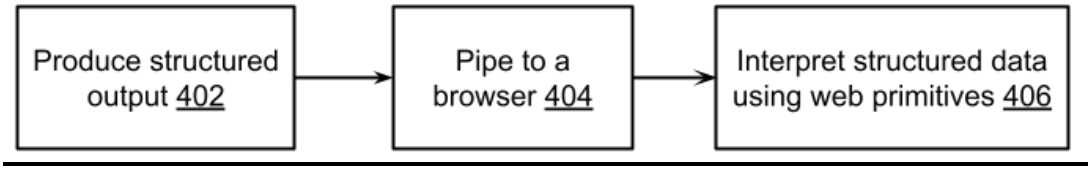


Fig. 4: Structuring command-line interfaces for improved accessibility

This disclosure describes techniques, as illustrated in Fig. 4, to provide a structured data endpoint on a command-line interface (or tool) and to enable a screen-reader to interface with such structured data via a standard client, e.g., a web-based client. Developers are required to author commands that produce consistent, structured output, e.g., in JavaScript Object Notation (JSON) format (402). JSON advantageously is a human-readable standard that enables web applications, e.g., it is interpretable over a browser interface.

The structured data, e.g., in JSON format, is piped to a browser (404). The structured data is interpreted and presented using HTML-ARIA (accessible rich internet applications) format, thus leveraging existing accessibility web primitives and standards (406). By structuring CLI output as tables (or trees or other structures) in an existing HTML accessibility standard, users experience consistent, familiar, web navigation rather than as undifferentiated text. Effectively, the user can interact with a browser-based command-line interface and use the same, popular,

web-accessibility tools, e.g., screen-readers, web rotors, switch access, tongue-controlled devices, up-down keyboard arrows, voice commands, etc., that they are familiar with.

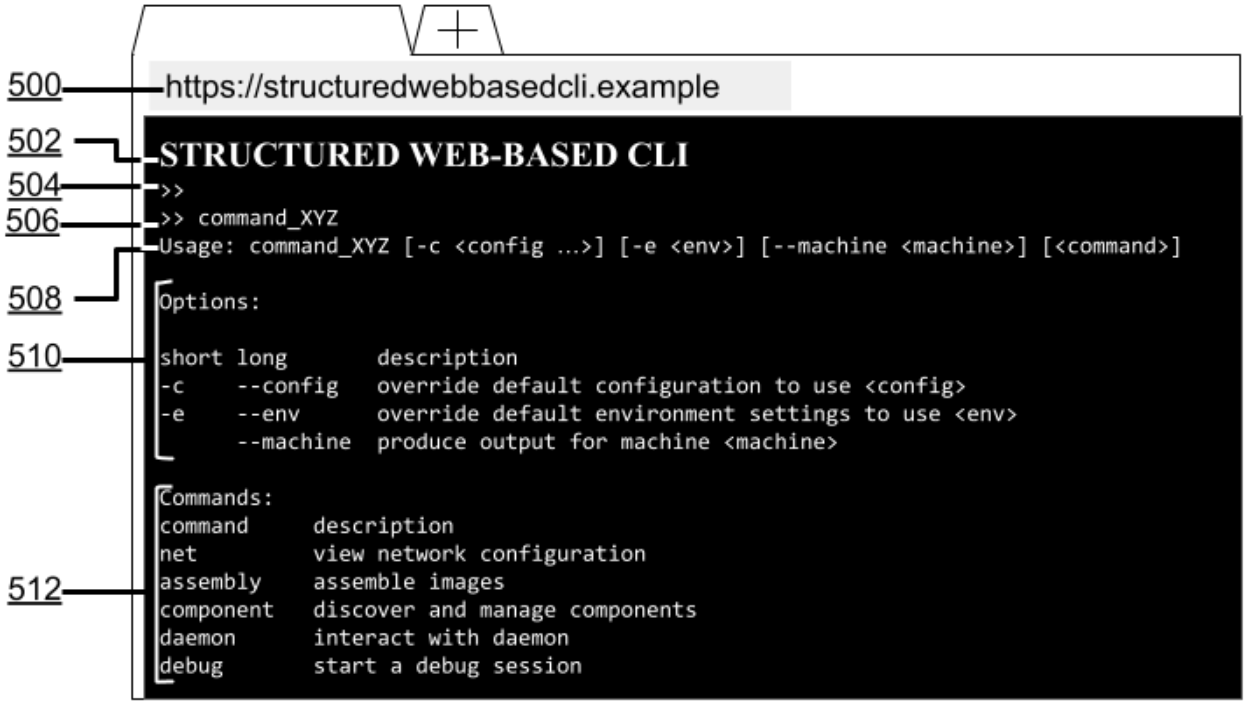


Fig. 5: Structured, web-based command-line interface

Fig. 5 illustrates a structured, web-based command-line interface. Although looking no different from an ordinary CLI, its data, e.g., the CLI input and output, is actually a JSON object accessed via a browser (500). Retaining data in the form of a JSON object provides several affordances; in particular, the structure in the data, e.g., its tabular form, is preserved. Thus, although the command input 506 (`command_XYZ`) and the considerably complex command output-sections 508 (`Usage …`), 510 (`Options`), 512 (`Commands`) look like undifferentiated text (as in a terminal), they are actually cells in a table.

As in any JSON table, cells in the table can have sub-tables, sub-sub-tables, etc. The session title (502) can be placed in the top-level, or most important (H1) heading of the HTML page. The command prompt (504) can be placed consistently in the second-level (H2) heading of

the HTML page. Textual properties in the output, e.g., color (red for erroneous, green for correct output), font (italics for indicating importance), etc., are retained in the JSON object, e.g., marked semantically to be interpretable by a screen-reader. Based on the semantics, a red-green color-blind user can, for example, change the color scheme to orange for erroneous and blue for correct output, or use an alert sound for erroneous output. The developer of the command doesn't make particular adaptations to the command output (except to make it JSON-structured); accessibility adaptations occur when the command output is provided to the user via a screen-reader.

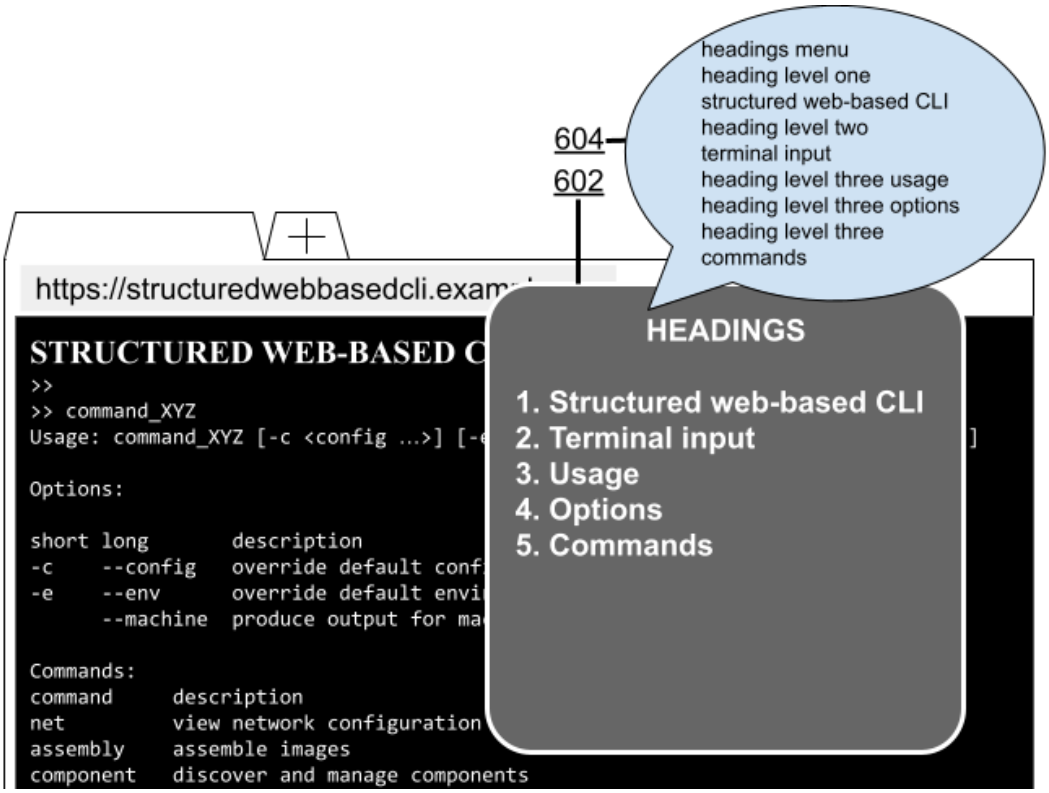


Fig. 6: Navigation between table entries, terminal input, command history, etc.

Leveraging the capabilities of HTML-ARIA, which is cognizant of the semantics of structured HTML data, presenting and interpreting a CLI as a structured JSON object enables easy navigation by screen-reader users, as illustrated in the example of Fig. 6. A navigator panel

(602) can optionally be popped up that lists headings as links. Rather than read the entire command-line input/output as undifferentiated text, the screen-reader (604) only reads out the contents of the navigator panel (e.g., '*usage*,' '*options*,' '*commands*,' etc.).

Leveraging the tabular structure, the screen-reader also prefaces each header with its position (e.g., '*heading level three usage*'), thus providing context to the screen-reader user. Based on screen-reader vocalization, the user can navigate through the navigator links using any of the standard HTML-ARIA techniques and familiar accessibility techniques, e.g., web rotors, up-down keyboard arrows, switch access, tongue-controlled devices, voice commands, etc. Vocalizing headings as opposed to undifferentiated CLI text and drilling down to the contents of a cell of a table only when its link is activated presents a substantially lower cognitive load on the user as compared to understanding and acting upon undifferentiated text.

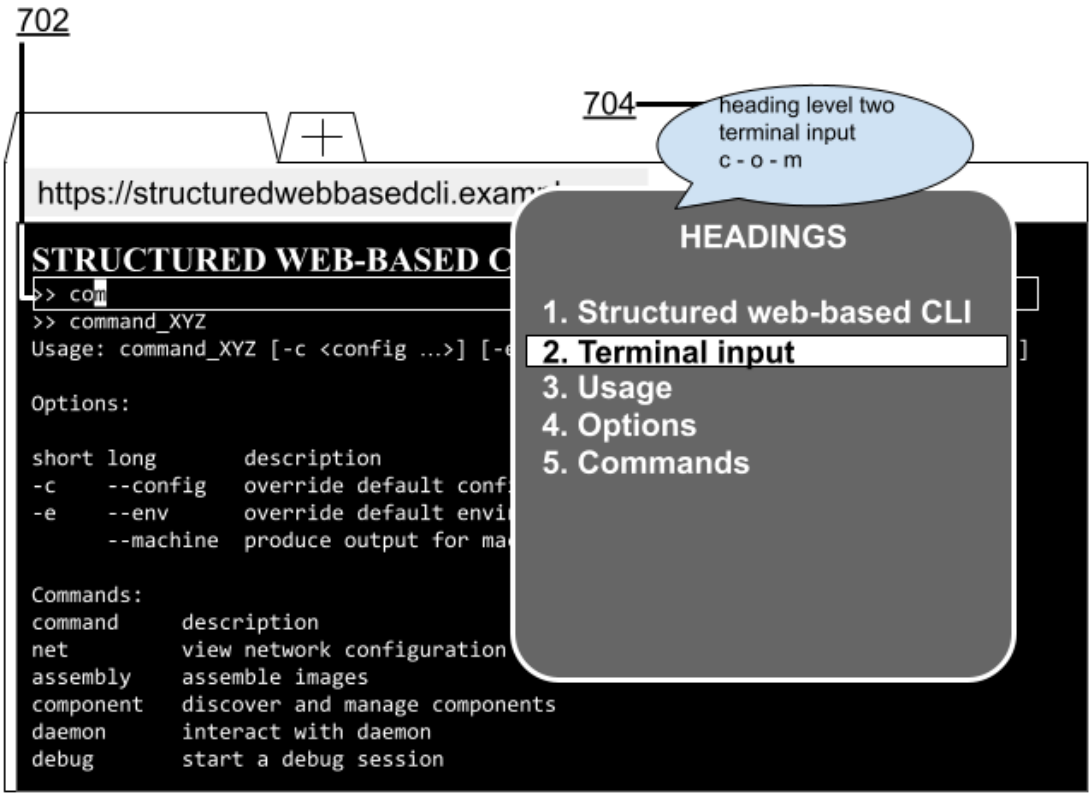


Fig. 7: Navigating to the terminal input

Fig. 7 illustrates navigating to the terminal input, also known as command prompt. As the user moves through the items of the navigator panel, the screen-reader (704) voices out the current link (‘*terminal input*’) and its position (‘*heading level two*’). Upon activating the current link (using a familiar assistive technology, e.g., web rotors, up-down keyboard arrows, switch access, tongue-controlled devices, voice commands, etc.), user control is transferred to the command prompt (702), where, as the user types out a new command, the screen-reader voices it out (‘*c-o-m …*’). Rather than wade through undifferentiated text, leveraging the tabular structure of the command-line data, the screen-reader user is enabled, with only a few steps, to navigate quickly to the command prompt to issue a new command.

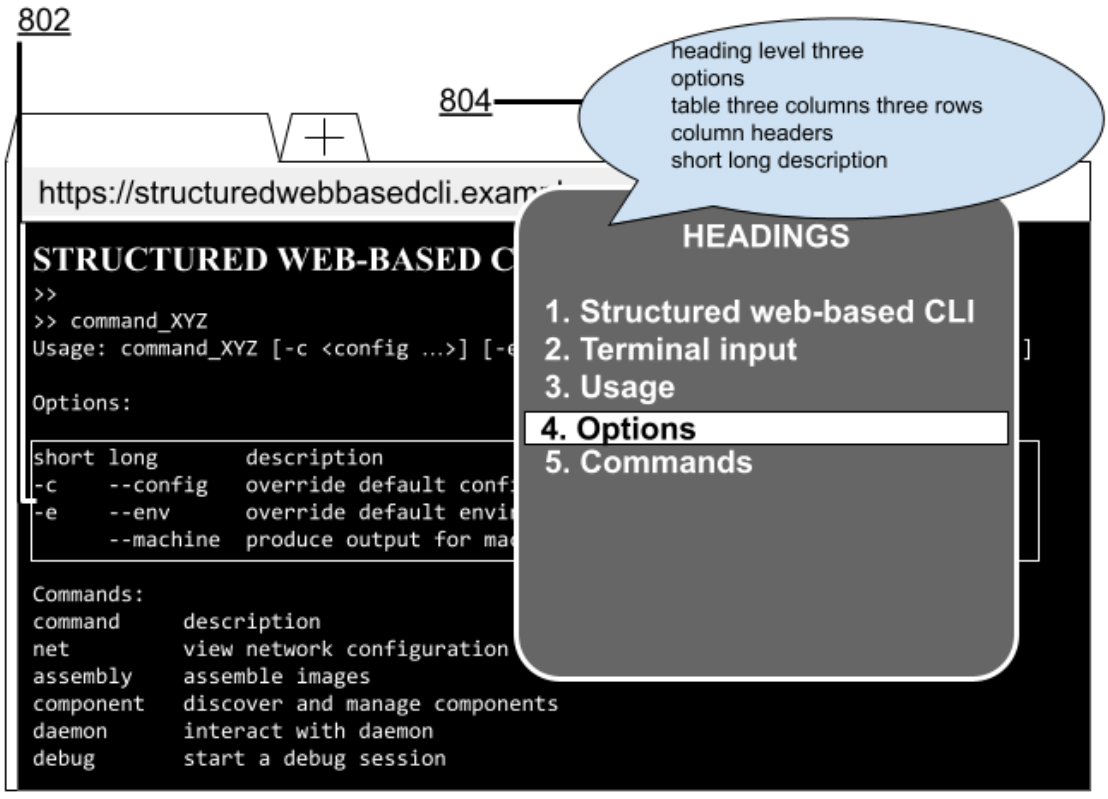


Fig. 8: Navigating to a section of command output

Fig. 8 illustrates navigating to a section of the command output, the `options` section in this example. As the user moves through the items of the navigator panel, the screen-reader (804)

voices out the current link (‘*options*’) and its position (‘*heading level three*’). Upon activating the current link using a familiar assistive technology, user control is transferred to the corresponding section of the command output (802), where the screen-reader voices out a table description (‘*table three columns three rows*’) and declares the column headers (‘*column headers short long description*’). Leveraging the tabular structure of the command-line data, the screen-reader user is enabled, with only a few steps, to navigate to a section of the command output, to hear a description of the table, and to hear not merely an unrelated sequence of words but their significance as column headers.

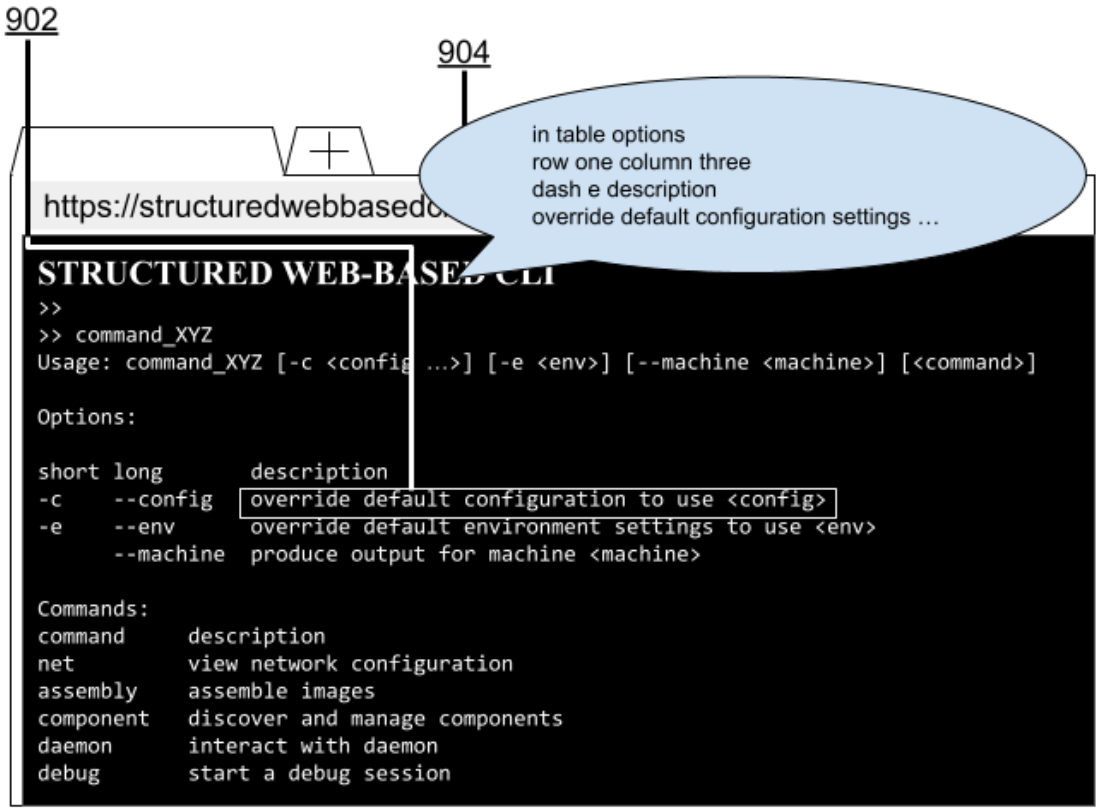


Fig. 9: Navigating to a subsection of the command output

Fig. 9 illustrates navigating to a subsection of the command output, the description of the `--config` parameter of the `options` section (902) in this example. Once in the table, as the user moves through cells within the table, the screen-reader (904) voices out the table name (‘*in*

table options'); the coordinates of the current cell ('*row one column three*'); the row header ('*dash e*'); the column header ('*description*'); the contents of the cell ('*override default configuration …*'). Leveraging the tabular structure of the command-line data, the screen-reader user is enabled to not only navigate to a command-output subsection in only a few steps, but also to hear a description of the present table and of the present cell while hearing the contents of the cell. This is in contrast to screen reading undifferentiated text, where the user does not hear the significance (e.g., name of the table, name of the row and of the column) of the words being read out.

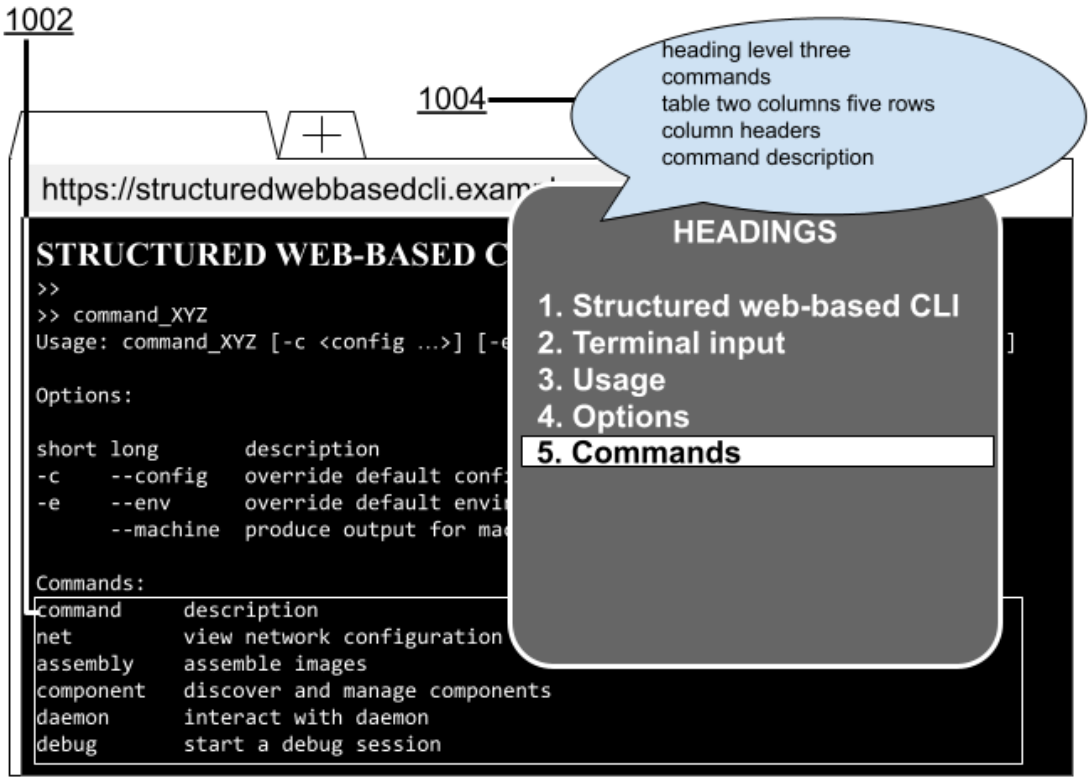


Fig. 10: Navigating to another section of the command output

Fig. 10 illustrates navigating to another section of the command output, the `commands` section in this example. As the user moves through the items of the navigator panel, the screen-reader (1004) voices out the current link ('*commands*') and its position ('*heading level three*').

Upon activating the current link using a familiar assistive technology, user control is transferred to the corresponding section of the command output (1002), where the screen-reader voices out a table description (‘*table two columns five rows*’) and declares the column headers (‘*column headers command description*’). Leveraging the tabular structure of the command-line data, the screen-reader user is enabled, with only a few steps, to navigate to a section of the command output, to hear a description of the table, and to hear not merely an unrelated sequence of words but their significance as column headers.

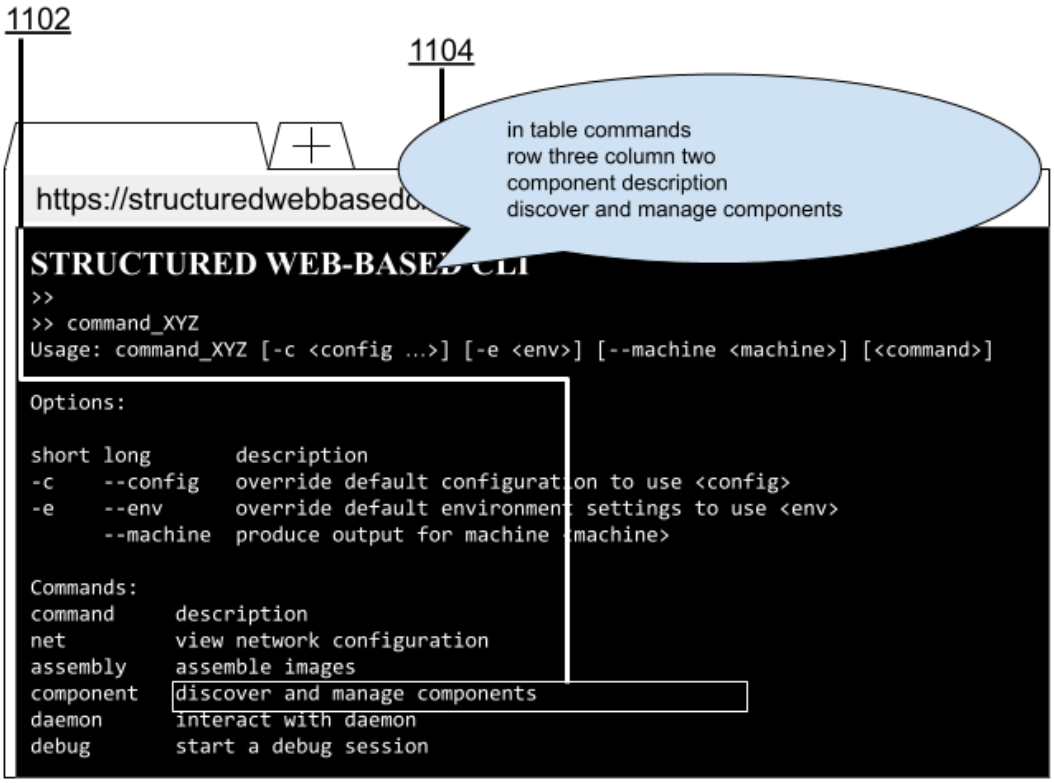


Fig. 11: Navigating to a subsection of the command output

Fig. 11 illustrates navigating to a subsection of command output, the description of the `component` parameter of the `commands` section (1102), in this example. Once in the table, as the user moves through cells within the table, the screen-reader (1104) voices out the table name (‘*in table commands*’); the coordinates of the current cell (‘*row three column two*’); the row

header (‘*component*’); the column header (‘*description*’); the contents of the cell (‘*discover and manage components*’).

Leveraging the tabular structure of the command-line data, the screen-reader user is enabled to not only navigate to a command-output subsection in only a few steps, but also to hear a description of the present table and of the present cell while hearing the contents of the cell. This is in contrast to reading out undifferentiated text, where the user does not hear the significance (e.g., name of table, name of the row and of the column) of the words being read out.

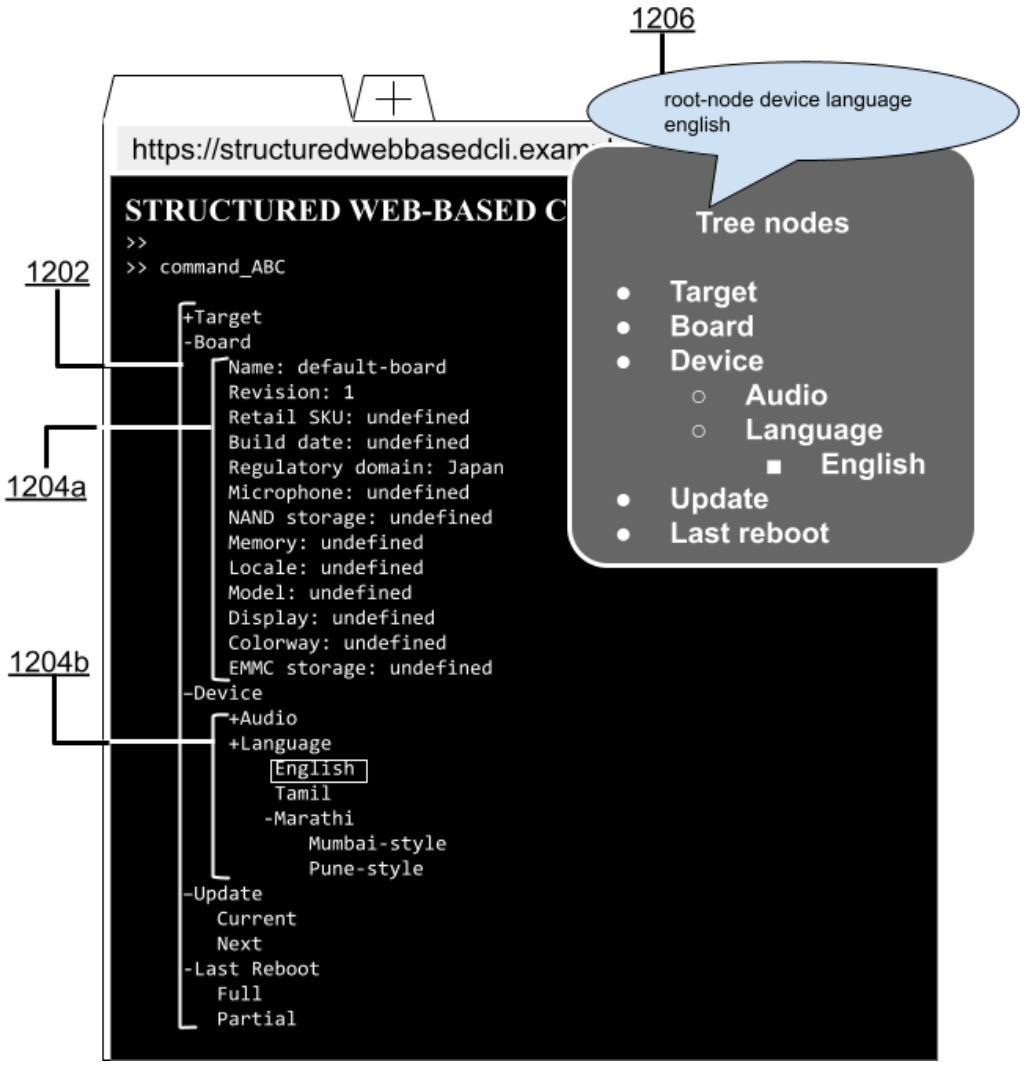


Fig. 12: Structured, web-based, display of a tree

The described techniques, e.g., of generating CLIs as structured JSON output rather than undifferentiated text, apply to various types of structured data, e.g., tables, lists, trees, messages, etc. For example, Fig. 12 illustrates a command that produces an output (1202) in the form of a tree of substantial breadth (1204a) and depth (1204b). Were the tree merely visually formatted as one while actually being undifferentiated text with no structure, a screen-reader user would have to endure a recitation of each node (many of whom, as illustrated, are repetitive and unilluminating, such as the descriptor '`undefined`'), all the way down to the leaves, before hearing the node of interest. With the CLI output formatted as a JSON tree, as described herein, a screen-reader (1206) can recite just the main nodes and, when the user activates a node (and its children) using a convenient assistive technology such as web rotors, up-down keyboard arrows, switch access, tongue-controlled devices, voice commands, etc., the screen-reader can quickly traverse down to just the leaf of interest to the user.

Fig. 13: Features of structured, web-based, CLI for improved accessibility

The HTML document structure can be further leveraged to enhance usability for screen-reader users as follows:

- As illustrated in Fig. 13, for easier discovery and navigation, the input prompt of the CLI (1302) can be made the top-level, or most important (H1), heading of the HTML page.
- Furthermore, the CLI presentation can be adapted for assistive technology users by optionally inverting scrolling from conventional bottom-up to top-down. Command line interfaces use bottom-up scrolling because they evolved from typewriters. Making the list scroll top-down makes it easier for assistive technology users to jump to the input at the top of the page without having to scroll through an increasingly long list of history to enter a new command. This is illustrated in Fig. 13, where, in contrast to conventional CLIs, the latest command (1304) and its output (1306) appear above the previous command and output (1308). As always, the latest command ('`command_XYZ --machine theMachine`') and its output remains structured and navigable, just as the previous command.
- As illustrated in Fig. 13, each command in the command history is added to an HTML list as a second-level (H2) heading (e.g., the previous command '`command_XYZ`' and the latest command '`command_XYZ --machine theMachine`' form an HTML list), which makes them immediately navigable, e.g., via web rotors, up-down keyboard arrows, switch access, tongue-controlled devices, voice commands, etc. The command response (output) is nested beneath the commands, enabling easier history browsing.

The described techniques are advantageous for non-assistive technology users too. For example, converting the CLI input/output to proper HTML rather than formatted but unstructured text enables accurate response to resizing. Enabling UI tools to present structured

output enables the adaptation of the presentation for different assistive technology needs. It also enables the writing of widgets that present structured data (e.g., tables, trees, information messages, errors, etc.) once and provide consistent presentation across the entire command surface. The UI inconsistency of plugins created by independent teams in how they print structured data is eliminated.

In this manner, by providing structured data in a format that enables screen-readers to read it as data rather than as undifferentiated lines of command-line text, the described techniques make command-line tooling more inclusive and lower barriers to and improve usability by users of assistive technology. The techniques also lower barriers to contribution by those developers of assistive technology who are also users of assistive technology.

CONCLUSION

This disclosure describes techniques to provide a structured-data endpoint on a CLI and to enable a screen-reader to interface with and vocalize such structured data via a standard, web-based client. The described techniques make command-line tooling more inclusive and lower barriers to and improve usability by users of assistive technology. The techniques also lower barriers to contribution by those developers of assistive technology who are also users of assistive technology.