XTerm.js

The coolest terminal since velocity?

Contents

Introduction to XTerm.js

- Terminal emulator developed using web technologies that brings a fully-featured terminal to browsers / embedded browser engines
- Based on the standard xterm terminal, released in 1984 and still used today
- Renders text using the Canvas API and various low-level techniques for very good performance
- Used in a lot of popular products, including Visual Studio Code and the Hyper terminal application
- Deliberately thin out of the box, to allow for maximum flexibility when building on top of the API
- Growing ecosystem of third-party libraries

ANSI Codes

- The terminal ”interface” is controlled by entering special codes, known as ANSI codes:

Cursor left	`\u001B[G]`
Backspace	`\b \b`
Red text	`\ESC[31m`

- There are libraries that include all these, so they are simple to use

What do we want?

- Basic features like entering and running commands, copy and paste, command history, etc.
- Context-aware styling of text
- Timestamp support
- Autocompletion of commands
- Additional context / documentation on hover
- Sending commands to attached Nordic board and printing results

Plugin System

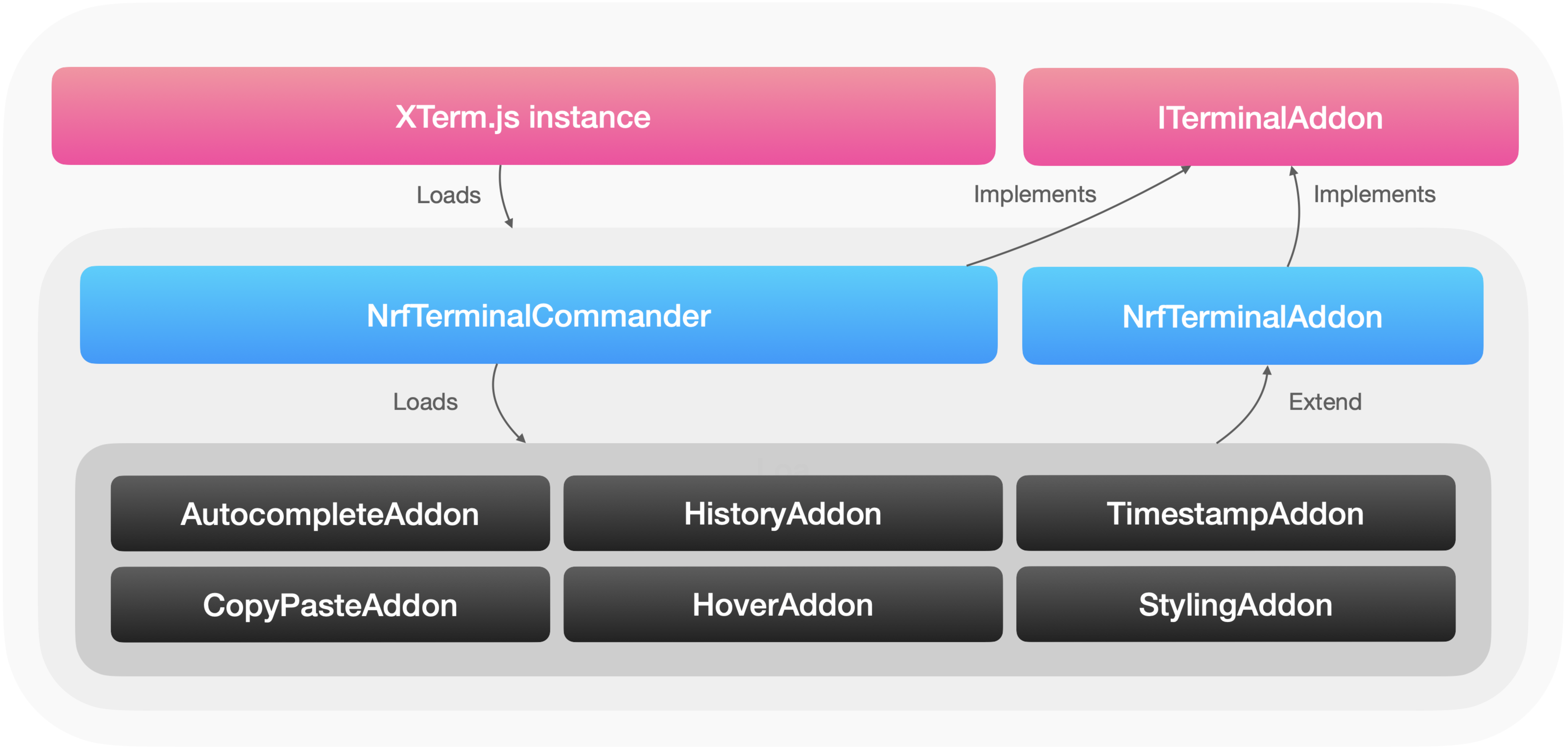
- All plugins must implement the `ITerminalAddon` interface, the definition of which is very simple:

```
interface ITerminalAddon {
    activate(terminal: Terminal): void;
    dispose(): void;
}
```

- The plugin can then be given to the terminal's `loadAddon()` method, which will register it and call the `activate()` method, passing itself as an argument

- The `activate()` method can be used to register event listeners on the terminal, and save it for later use

Architecture



NrfTerminalCommander

- The “master” plugin in the tree, responsible for creating and loading the additional NrfTerminal plugin objects
- Takes a configuration object for customising the terminal for a particular use case or context
- Provides essential functionality like a prompt, cursor movement, writing text, and running commands
- Exposes helper methods and data to plugins, such as the current line output, line number, and additional event listeners

NrfTerminalAddon

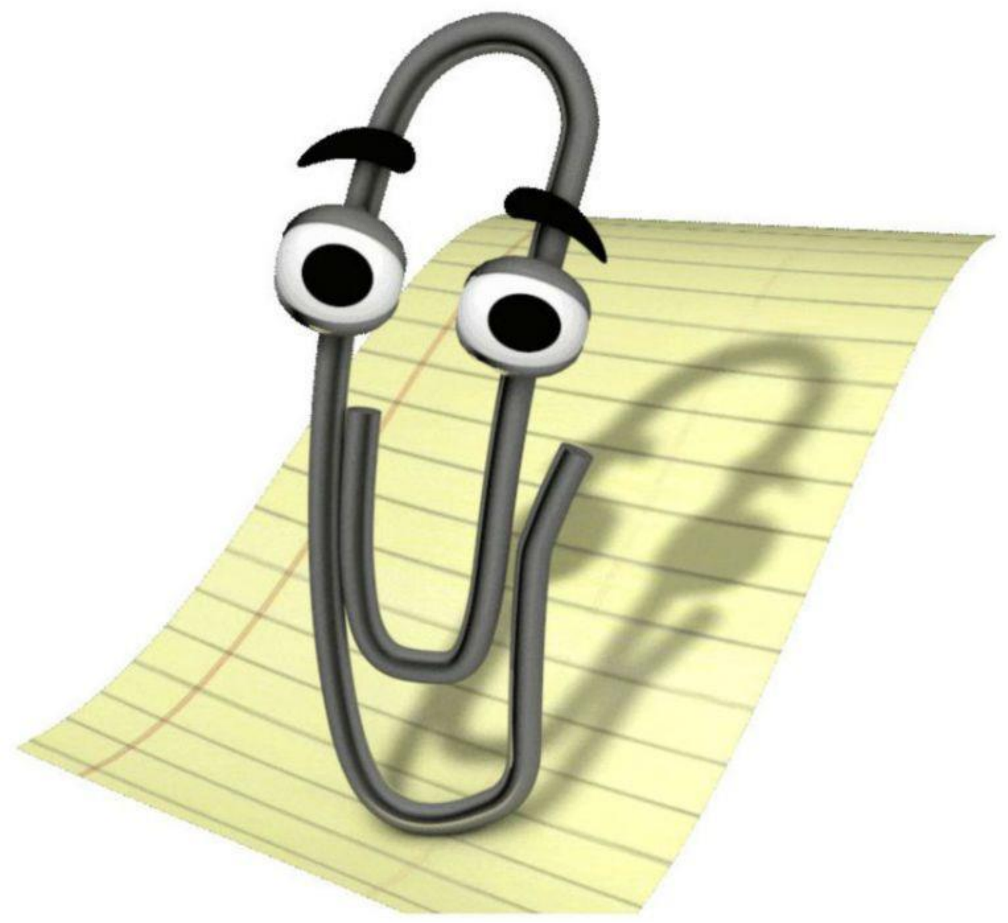
- Abstract class that implements `ITerminalAddon`:

```
abstract class NrfTerminalAddon implements ITerminalAddon {
    public abstract name: string;

    protected terminal!: Terminal;
    protected commander: NrfTerminalCommander;

    protected abstract onActivate(): void;
}
```

- All NrfTerminal plugins extend this class, ensuring they have access to an instance of the terminal and the `NrfTerminalCommander` without boilerplate

Copy and paste

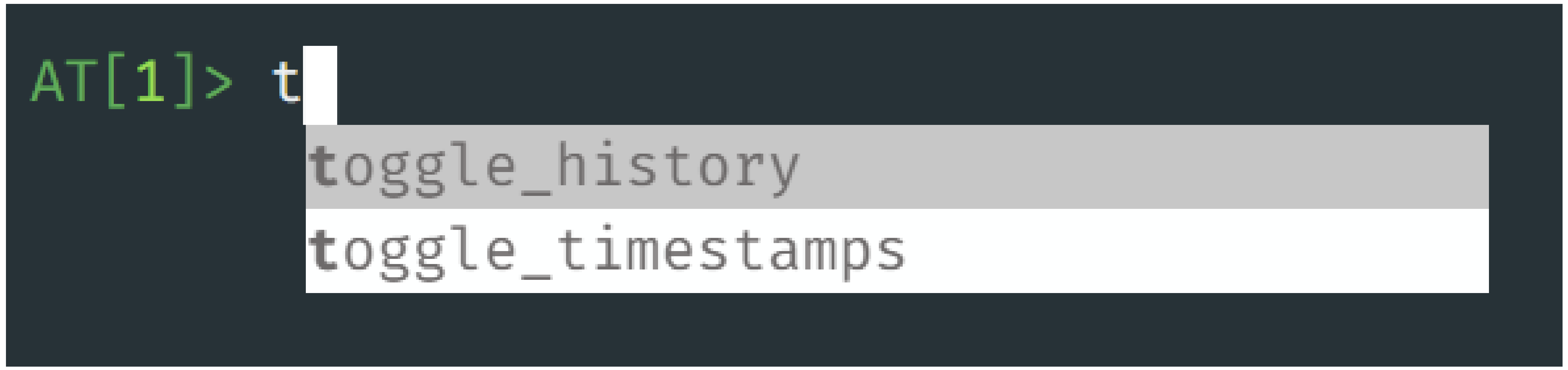
- Copy and paste works out-of-the-box on Macs (with ⌘-C / ⌘-V)
- Needs to be implemented manually on Windows and Linux
- Straightforward case of listening for the keyboard events and running the browser environment’s copy / paste commands



Autocompletion

- Though the terminal itself is a canvas element, that doesn't preclude us from rendering additional DOM elements in response to keyboard events

- This makes it possible to display interface elements like an autocomplete box:



```
AT[1]> t
        toggle_history
        toggle_timestamps
```

- For complicated interfaces, we could even render using React, passing the terminal and `NrfTerminalCommander` as props

- Raises some issues with conflicting keyboard events, which means plugins have to know about each other

Styling commands

- We might want to style known text, like AT commands, separately from other text, so that the user knows they’re on the right track
- XTerm.js includes support for highlighting URLs, with an option to override this using a custom regular expression
- Currently marked as deprecated in the TypeScript typings (but not the website), so this might not be a viable long term approach
- Potentially possible to insert ANSI escape codes retroactively, after a word has been typed

Showing additional data on hover

- Building on the previous add-on, we could hook into the link matcher API to insert a DOM element
- Gets quite tricky regarding how this element should be positioned, taking into account line width, scroll back position, etc.
- Discussion amongst maintainers on adding a proper “decorations” API, modelled on the one in VS Code

Questions?