@turbodocx/html-to-docx

1.20.1 • Public • Published 22 days ago

Readme

Code Beta

12 Dependencies

4 Dependents

28 Versions

@turbodocx/html-to-docx

Convert HTML to Word, Google Docs, and DOCX files with the fastest, most reliable JavaScript library available. Built for modern applications that demand speed and precision—from AI-powered document generation to enterprise reporting systems.

Based on the original work and assisted by the original contributors of **privateOmega/html-to-docx**, this library is now actively maintained and enhanced by TurboDocx, ensuring continuous improvements and long-term support for production environments.

🌐 Explore the TurboDocx Ecosystem

Package	Links	Description
API & SDK	Website TurboDocx	Production-ready SDK for e-signature, document generation, template processing, and more
TurboSign	Website TurboDocx	Digital signature API for seamless e-signature workflows
sdk	Stars 2	Official TurboDocx SDK for seamless API integration
next-plugin-llms	Stars 3	Next.js plugin for automatic llms.txt generation
n8n-nodes-turbodocx	npm v1.0.2 Stars 4	n8n community node for TurboDocx API & TurboSign

Why @turbodocx/html-to-docx?

🚀 **Lightning Fast Performance** - Pure JavaScript implementation with no dependencies on headless browsers or external binaries. Perfect for AI applications that need rapid document generation.

🔄 **Active Maintenance & Support** - Backed by TurboDocx with regular updates, bug fixes, and feature enhancements. Not another abandoned open-source project.

🤖 **AI-Ready Architecture** - Designed for modern AI workflows where speed matters. Generate thousands of documents without the overhead of browser automation.

⚡ **Zero External Dependencies** - No need for Puppeteer, Chrome, or LibreOffice. Pure Node.js implementation that works in any environment.

🎯 **Production Battle-Tested** - Used in production environments processing thousands of documents. Reliable, stable, and performant at scale.

🛠️ **Developer Experience** - Full TypeScript support, comprehensive documentation, and extensive examples to get you up and running in minutes.

Installation

Use the npm to install the project.

```
npm install @turbodocx/html-to-docx
```

TypeScript Support

This package includes TypeScript typings. No additional installation is required to use it with TypeScript projects.

TypeScript Example

```
import HtmlToDocx from "@turbodocx/html-to-docx";

const htmlString = `<!DOCTYPE html>
    <html lang="en">
        <head>
            <meta charset="UTF-8" />
            <title>Document</title>
        </head>
        <body>
            <h1>Hello world</h1>
        </body>
    </html>`;

// Basic usage
```

```typescript
async function basicExample() {
  const docx = await HtmlToDocx(htmlString);
  // docx is ArrayBuffer in Node.js or Blob in browser environme
}

// With header
async function withHeader() {
  const headerHtml = "<p>Document Header</p>";
  const docx = await HtmlToDocx(htmlString, headerHtml);
}

// With document options
async function withOptions() {
  const docx = await HtmlToDocx(htmlString, null, {
    orientation: "landscape",
    title: "TypeScript Example",
    creator: "TurboDocx",
    table: {
      row: {
        cantSplit: true,
      },
      borderOptions: {
        size: 1,
        color: "000000"
      }
    },
    pageNumber: true,
    footer: true
  });
}

// With image processing options
async function withImageOptions() {
  const htmlWithImages = `<div>
```

```
    <img src="https://example.com/image.jpg" alt="Example">
  </div>`;

  const docx = await HtmlToDocx(htmlWithImages, null, {
    imageProcessing: {
      maxRetries: 3,          // Retry failed image downloads up
      verboseLogging: true,   // Enable detailed logging for deb
      downloadTimeout: 10000, // 10 second timeout per download
      maxImageSize: 5242880   // 5MB max image size
    }
  });
}

// With all parameters
async function complete() {
  const headerHtml = "<p>Document Header</p>";
  const footerHtml = "<p>Page Footer</p>";

  const docx = await HtmlToDocx(
    htmlString,
    headerHtml,
    {
      orientation: "landscape",
      pageSize: {
        width: 12240,
        height: 15840
      },
      margins: {
        top: 1440,
        right: 1800,
        bottom: 1440,
        left: 1800
      },
      title: "Complete Example",
```

```
      creator: "TurboDocx",
    },
    footerHtml
  );
}
```

For more comprehensive TypeScript examples, check out the following files in the `example/typescript` directory:

- `typescript-example.ts` - A complete example showing how to generate and save DOCX files using TypeScript
- `type-test.ts` - Demonstrates the type checking capabilities provided by the TypeScript definitions

Running the TypeScript Examples

To run the TypeScript examples:

```
# Navigate to the example directory
cd example/typescript

# Install ts-node globally (if not already installed)
npm install -g ts-node typescript

# Ensure @turbodocx/html-to-docx is built and accessible
# From the root directory of the project:
# npm install
# npm run build

# Run the TypeScript example directly
ts-node typescript-example.ts
```

This will generate two DOCX files in the `example/typescript` directory:

- `basic-example.docx` - A simple document with minimal configuration

- `advanced-example.docx` - A document with headers, footers, and advanced formatting options

Browser Standalone Build

The library provides a standalone browser build that bundles all dependencies into a single file. This allows you to use the library directly in HTML pages without any build tools or module bundlers.

Build Outputs

When you run `npm run build`, three distribution files are generated:

File	Format	Size	Dependencies	Use Case
`dist/html-to-docx.esm.js`	ES Module	~1.6 MB	External	Modern bundlers (Webpack, Vite, Rollup)
`dist/html-to-docx.umd.js`	UMD	~1.6 MB	External	Node.js, AMD, or manual dependency management
`dist/html-to-docx.browser.js`	IIFE	~2.4 MB	**All bundled**	Direct browser usage, CDN, quick prototypes

Build Commands

```
# Build all versions (ESM + UMD + Browser)
npm run build

# Build only the browser standalone version (development)
npm run build:browser
```

```
# Build only the browser standalone version (production, minifie
npm run build:browser:prod
```

Browser Usage

Include the standalone browser build directly in your HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML to DOCX Demo</title>
</head>
<body>
  <!--
    Polyfills for Node.js globals (required)
    Note: While rollup-plugin-polyfill-node bundles most Node.js
    these runtime globals must be set before the library loads b
    some dependencies check for them synchronously during initia
  -->
  <script>
    if (typeof global === 'undefined') window.global = window;
    if (typeof process === 'undefined') window.process = { env:
    if (typeof Buffer === 'undefined') {
      window.Buffer = {
        from: function(data, encoding) {
          if (typeof data === 'string') {
            // Handle base64 and utf-8 encoding
            if (encoding === 'base64') {
              var binary = atob(data);
              var bytes = new Uint8Array(binary.length);
              for (var i = 0; i < binary.length; i++) bytes[i] =
              return bytes;
            }
            return new TextEncoder().encode(data);
```

```
        }
        return new Uint8Array(data);
      },
      isBuffer: function() { return false; }
    };
  }
</script>

<!-- Include the standalone browser build -->
<script src="path/to/html-to-docx.browser.js"></script>

<script>
  async function generateDocument() {
    const htmlContent = `
      <h1>Hello World</h1>
      <p>This is a <strong>test document</strong> generated ir
    `;

    try {
      const result = await HTMLToDOCX(htmlContent, null, {
        title: 'My Document',
        creator: 'Browser App'
      });

      // Convert result to Blob for download
      let blob;
      if (result instanceof Blob) {
        blob = result;
      } else if (result instanceof ArrayBuffer || result insta
        blob = new Blob([result], {
          type: 'application/vnd.openxmlformats-officedocument
        });
      }
```

```
        // Trigger download
        const url = URL.createObjectURL(blob);
        const a = document.createElement('a');
        a.href = url;
        a.download = 'document.docx';
        a.click();
        URL.revokeObjectURL(url);
      } catch (error) {
        console.error('Error generating DOCX:', error);
      }
    }
  </script>

  <button onclick="generateDocument()">Generate DOCX</button>
</body>
</html>
```

Testing the Browser Build

A test page is included to verify the browser build works correctly:

```
# Build and start the test server
npm run test:browser
```

Then open **http://localhost:8080/tests/test_browser.html** in your browser.

The test page allows you to:

- Edit HTML content in a textarea
- Click "Generate DOCX" to create and download a Word document
- See status messages for success or errors

CDN Usage

You can also host the browser build on a CDN for easy inclusion:

```
<!-- Example: Self-hosted or CDN -->
<script src="https://your-cdn.com/html-to-docx/1.18.1/html-to-dc
```

Limitations in Browser Environment

- **Sharp (SVG conversion)**: The `sharp` image processing library is not available in browsers. SVG images will be embedded natively (requires Office 2019+).
- **File System**: No direct file system access. Documents are returned as Blob/ArrayBuffer for download.
- **Image URLs**: Remote images must be CORS-enabled or converted to base64 data URLs.

Usage

```
await HTMLtoDOCX(htmlString, headerHTMLString, documentOptions,
```

full fledged examples can be found under `example/`

Parameters

- `htmlString` <String> clean html string equivalent of document content.
- `headerHTMLString` <String> clean html string equivalent of header. Defaults to `<p></p>` if header flag is `true` .
- `documentOptions` <?Object>
  - `orientation` <"portrait"|"landscape"> defines the general orientation of the document. Defaults to portrait.
  - `pageSize` <?Object> Defaults to U.S. letter portrait orientation.
    - `width` <Number> width of the page for all pages in this section in TWIP. Defaults to 12240. Maximum 31680. Supports equivalent measurement in pixel, cm or inch.
    - `height` <Number> height of the page for all pages in this section in TWIP. Defaults to 15840. Maximum 31680. Supports equivalent measurement in pixel, cm or inch.
  - `margins` <?Object>

        - `top` <Number> distance between the top of the text margins for the main document and the top of the page for all pages in this section in TWIP. Defaults to 1440. Supports equivalent measurement in pixel, cm or inch.
        - `right` <Number> distance between the right edge of the page and the right edge of the text extents for this document in TWIP. Defaults to 1800. Supports equivalent measurement in pixel, cm or inch.
        - `bottom` <Number> distance between the bottom of text margins for the document and the bottom of the page in TWIP. Defaults to 1440. Supports equivalent measurement in pixel, cm or inch.
        - `left` <Number> distance between the left edge of the page and the left edge of the text extents for this document in TWIP. Defaults to 1800. Supports equivalent measurement in pixel, cm or inch.
        - `header` <Number> distance from the top edge of the page to the top edge of the header in TWIP. Defaults to 720. Supports equivalent measurement in pixel, cm or inch.
        - `footer` <Number> distance from the bottom edge of the page to the bottom edge of the footer in TWIP. Defaults to 720. Supports equivalent measurement in pixel, cm or inch.
        - `gutter` <Number> amount of extra space added to the specified margin, above any existing margin values. This setting is typically used when a document is being created for binding in TWIP. Defaults to 0. Supports equivalent measurement in pixel, cm or inch.
    - `title` <?String> title of the document.
    - `subject` <?String> subject of the document.
    - `creator` <?String> creator of the document. Defaults to `html-to-docx`
    - `keywords` <?Array<String>> keywords associated with the document. Defaults to ['html-to-docx'].
    - `description` <?String> description of the document.
    - `lastModifiedBy` <?String> last modifier of the document. Defaults to `html-to-docx` .
    - `revision` <?Number> revision of the document. Defaults to `1` .
    - `createdAt` <?Date> time of creation of the document. Defaults to current time.
    - `modifiedAt` <?Date> time of last modification of the document. Defaults to current time.
    - `headerType` <"default"|"first"|"even"> type of header. Defaults to `default` .

- `header` <?Boolean> flag to enable header. Defaults to `false`.
- `footerType` <"default"|"first"|"even"> type of footer. Defaults to `default`.
- `footer` <?Boolean> flag to enable footer. Defaults to `false`.
- `font` <?String> font name to be used. Defaults to `Times New Roman`.
- `fontSize` <?Number> size of font in HIP(Half of point). Defaults to `22`. Supports equivalent measure in pt.
- `complexScriptFontSize` <?Number> size of complex script font in HIP(Half of point). Defaults to `22`. Supports equivalent measure in pt.
- `table` <?Object>
  - `row` <?Object>
    - `cantSplit` <?Boolean> flag to allow table row to split across pages. Defaults to `false`.
  - `borderOptions` <?Object>
    - `size` <?Number> denotes the border size. Defaults to `0`.
    - `stroke` <?String> denotes the style of the borderStrike. Defaults to `nil`.
    - `color` <?String> determines the border color. Defaults to `000000`.
  - `addSpacingAfter` <?Boolean> flag to add an empty paragraph after tables for spacing. Defaults to `true`.
- `pageNumber` <?Boolean> flag to enable page number in footer. Defaults to `false`. Page number works only if footer flag is set as `true`.
- `skipFirstHeaderFooter` <?Boolean> flag to skip first page header and footer. Defaults to `false`.
- `lineNumber` <?Boolean> flag to enable line numbering. Defaults to `false`.
- `lineNumberOptions` <?Object>
  - `start` <Number> start of the numbering - 1. Defaults to `0`.
  - `countBy` <Number> skip numbering in how many lines in between + 1. Defaults to `1`.
  - `restart` <"continuous"|"newPage"|"newSection"> numbering restart strategy. Defaults to `continuous`.
- `numbering` <?Object>
  - `defaultOrderedListStyleType` <?String> default ordered list style type. Defaults to `decimal`.
- `heading` <?Object> custom heading styles configuration
- `heading1` - `heading6` <?Object> heading style configuration

        - `font` <?String> font family
        - `fontSize` <?Number> font size in half-points
        - `bold` <?Boolean> whether text is bold. Defaults to `true`
        - `spacing` <?Object> paragraph spacing configuration
            - `before` <?Number> spacing before heading in twips
            - `after` <?Number> spacing after heading in twips
        - `keepLines` <?Boolean> keep lines together. Defaults to `true`
        - `keepNext` <?Boolean> keep with next paragraph. Defaults to `true`
        - `outlineLevel` <?Number> outline level (0-5)
    - `decodeUnicode` <?Boolean> flag to enable unicode decoding of header, body and footer. Defaults to `false`.
    - `lang` <?String> language localization code for spell checker to work properly. Defaults to `en-US`.
    - `direction` <?String> text direction for RTL (right-to-left) languages. Set to `'rtl'` for Arabic, Hebrew, etc. Defaults to `'ltr'`.
    - `preProcessing` <?Object>
        - `skipHTMLMinify` <?Boolean> flag to skip minification of HTML. Defaults to `false`.
    - `imageProcessing` <?Object>
        - `maxRetries` <?Number> maximum number of retry attempts for failed image downloads. Defaults to `2`.
        - `verboseLogging` <?Boolean> flag to enable detailed logging of image processing operations. Defaults to `false`.
        - `downloadTimeout` <?Number> timeout in milliseconds for each image download attempt. Defaults to `5000` (5 seconds).
        - `maxImageSize` <?Number> maximum allowed image size in bytes. Defaults to `10485760` (10MB).
        - `retryDelayBase` <?Number> base delay in milliseconds for exponential backoff between retries. Defaults to `500` (500ms).
        - `minTimeout` <?Number> minimum timeout in milliseconds. Defaults to `1000` (1 second).
        - `maxTimeout` <?Number> maximum timeout in milliseconds. Defaults to `30000` (30 seconds).
        - `minImageSize` <?Number> minimum image size in bytes. Defaults to `1024` (1KB).

    - `maxCacheSize` <?Number> maximum total cache size in bytes (LRU cache limit to prevent OOM). Defaults to `20971520` (20MB).
    - `maxCacheEntries` <?Number> maximum number of unique images in cache (LRU eviction). Defaults to `100`.
    - `svgHandling` <?String> strategy for handling SVG images. Defaults to `'convert'`. Options:
        - `'convert'` - Converts SVG to PNG for maximum compatibility with all Word versions (requires `sharp` package)
        - `'native'` - Embeds SVG natively for Office 2019+ (preserves vector quality)
- `footerHTMLString` <String> clean html string equivalent of footer. Defaults to `<p></p>` if footer flag is `true`.

Returns

<Promise<Buffer|Blob>>

Notes

Currently page break can be implemented by having div with classname "page-break" or style "page-break-after" despite the values of the "page-break-after", and contents inside the div element will be ignored. `<div class="page-break" style="page-break-after: always;"></div>`

CSS list-style-type for `<ol>` element are now supported. Just do something like this in the HTML:

```
<ol style="list-style-type:lower-alpha;">
  <li>List item</li>
  ...
</ol>
```

List of supported list-style-types:

- upper-alpha, will result in `A. List item`
- lower-alpha, will result in `a. List item`
- upper-roman, will result in `I. List item`

- lower-roman, will result in `i. List item`
- lower-alpha-bracket-end, will result in `a) List item`
- decimal-bracket-end, will result in `1) List item`
- decimal-bracket, will result in `(1) List item`
- decimal, **(the default)** will result in `1. List item`

Also you could add attribute `data-start="n"` to start the numbering from the n-th.

`<ol data-start="2">` will start the numbering from (B. b. II. ii. 2.)

SVG Image Support

The library provides comprehensive SVG image support with two strategies to fit your needs:

Installation & Package Size

The library supports SVG images with `sharp` as an optional peer dependency for high-quality SVG → PNG conversion.

Basic Installation (Lightweight):

```
npm install @turbodocx/html-to-docx
```

- **Package size**: ~2.8MB (sharp not included)
- **Compatibility**: SVGs embedded natively - requires Office 2019+ or Microsoft 365
- **Best for**: Modern-only environments or size-constrained deployments (Lambda, Edge)
- **Auto-fallback**: Library automatically uses native SVG mode when sharp is unavailable

Full Installation (Maximum Compatibility - Recommended):

```
npm install @turbodocx/html-to-docx sharp
```

- **Base package**: ~2.8MB
- **With sharp**: Additional ~34MB (platform-specific native binaries)
- **Compatibility**: Converts SVGs to PNG - works with all Word versions (2007+)
- **Best for**: Production applications requiring broad compatibility

Why is sharp optional?

Sharp is a native Node.js module that provides the best SVG to PNG conversion quality, but adds ~34MB of platform-specific native binaries to your `node_modules`. We've made it an optional peer dependency so you can choose:

Configuration	Install Command	Size	SVG Handling	Word Compatibil
Without sharp (default)	`npm install @turbodocx/html-to-docx`	2.8MB	Native SVG	Office 2019- only
With sharp (recommended)	`npm install @turbodocx/html-to-docx sharp`	2.8MB + 34MB binaries	PNG conversion	All versions (2007+)

The library **gracefully handles both scenarios** - if sharp is unavailable, SVGs are automatically embedded in native format.

1. Convert to PNG (Default - Maximum Compatibility)

By default, SVG images are automatically converted to PNG format for maximum compatibility with all Word versions (requires `sharp`):

```
const htmlWithSVG = `
  <div>
    <img src="
  </div>
`;

// Default behavior - SVG converted to PNG
const docx = await HTMLtoDOCX(htmlWithSVG);

// Or explicitly set to convert
const docx = await HTMLtoDOCX(htmlWithSVG, null, {
```

```
  imageProcessing: {
    svgHandling: 'convert'  // Converts SVG to PNG (default)
  }
});
```

Benefits:

- ✅ Works with all Word versions (2007+)
- ✅ Compatible with Word Online, Google Docs, LibreOffice
- ✅ No compatibility warnings or errors

2. Native SVG (Office 2019+ Only)

For modern Office environments, you can embed SVG images natively to preserve vector quality:

```
const htmlWithSVG = `
  <div>
    <img src="..." alt="Vector Graphic
  </div>
`;

const docx = await HTMLtoDOCX(htmlWithSVG, null, {
  imageProcessing: {
    svgHandling: 'native'  // Embed SVG natively (Office 2019+)
  }
});
```

Benefits:

- ✅ Perfect vector quality at any zoom level
- ✅ Smaller file sizes for complex graphics
- ✅ Editable in modern Office applications

Requirements:

- Microsoft Office 2019 or later
- Microsoft 365
- Word for Mac 2019+

Note: Older Word versions will show an "unreadable content" error with native SVG. Use `'convert'` mode for backwards compatibility.

Handling SVG Without Sharp

If `sharp` is not installed (e.g., using `--no-optional`), the library automatically falls back to native SVG embedding:

```
// Even with svgHandling: 'convert', if sharp unavailable → uses
const docx = await HTMLtoDOCX(htmlWithSVG, null, {
  imageProcessing: {
    svgHandling: 'convert',          // Tries to convert, falls b
    suppressSharpWarning: false,     // Set to true to suppress w
    verboseLogging: true             // Shows: "Sharp not availab
  }
});
```

No crashes, no errors - the library detects sharp availability at runtime and adjusts automatically:

```
# With sharp installed
✅ SVG → PNG conversion → Works in Word 2007+

# Without sharp (--no-optional)
ℹ️  SVG → Native embedding → Works in Office 2019+ only

# Suppress the warning (if intentionally using native SVG mode)
imageProcessing: { suppressSharpWarning: true }
```

Pro tip: For serverless/Lambda deployments with size constraints, install without sharp and set `suppressSharpWarning: true` to avoid console warnings. Document that generated files require Office 2019+.

RTL (Right-to-Left) Language Support

The library also supports RTL languages like Arabic and Hebrew. Use the `direction` option to enable RTL text flow:

```
const htmlString = `
  <h1>مرحبا بالعالم</h1>
  <p>هذا نص تجريبي باللغة العربية ليظهر من اليمين إلى اليسار</p>
`;

const docx = await HTMLtoDOCX(htmlString, null, {
  direction: 'rtl',         // Enable RTL text direction
  lang: 'ar-SA',           // Arabic locale (or 'he-IL' for Hebre
  font: 'Arial',           // Use a font that supports RTL charac
});
```

For more RTL examples, check `example/react-example/src/example-rtl.js` .

Font Compatibility

Font family doesnt work consistently for all word processor softwares

- Word Desktop work as intended
- LibreOffice ignores the fontTable.xml file, and finds a font by itself
- Word Online ignores the fontTable.xml file, and finds closest font in their font library

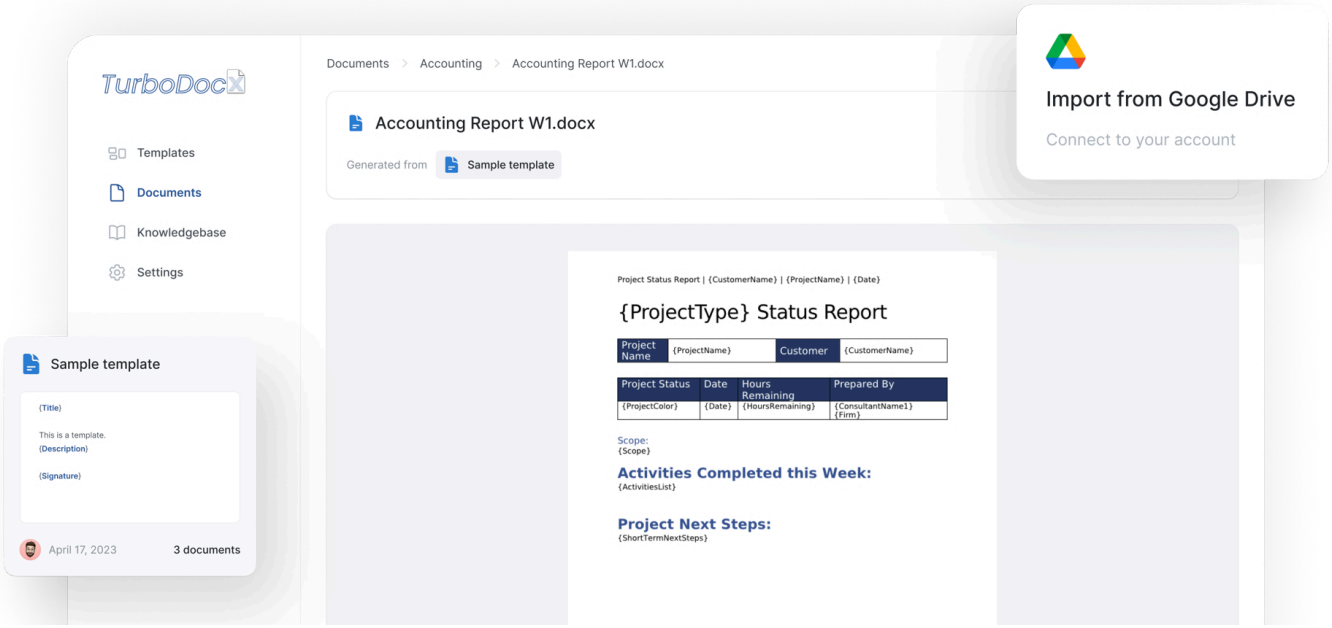
Contributing

Pull requests are welcome. For major changes, please open an issue first to discuss what you would like to change.

Please make sure to branch new branches off of develop for contribution.

Support

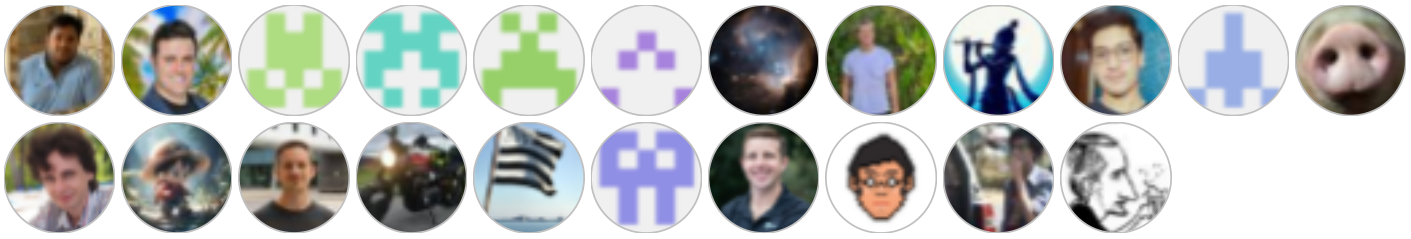
Proudly Sponsored by TurboDocx



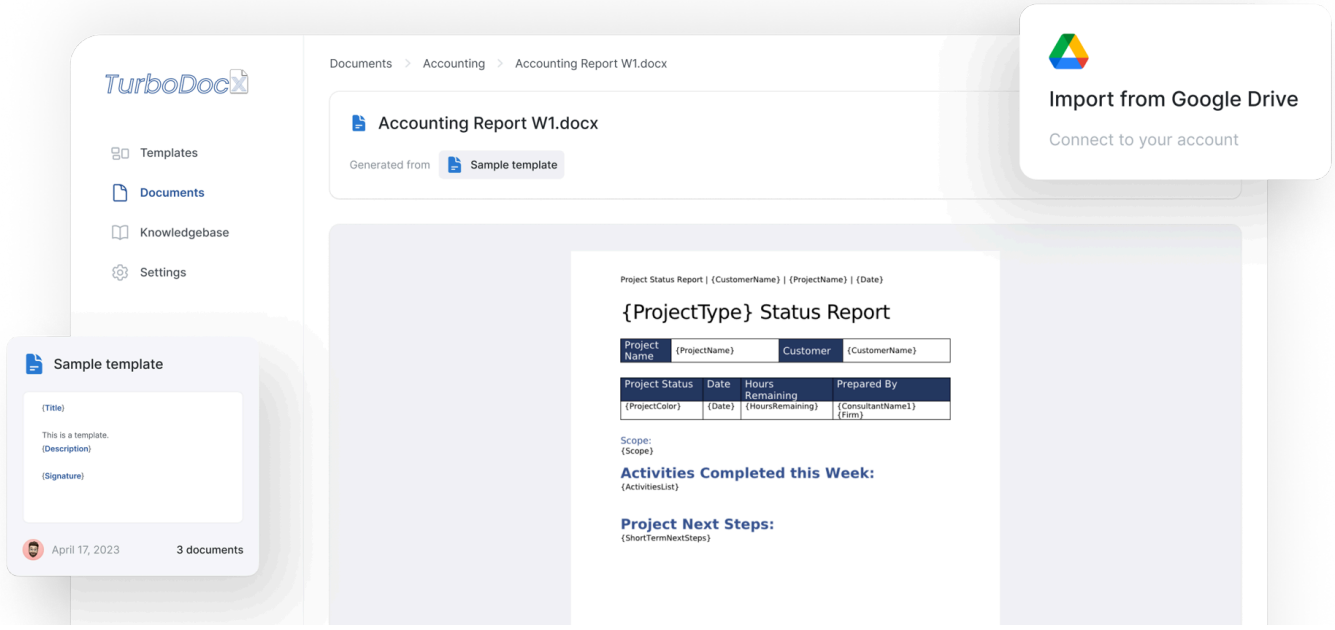
License

MIT

Contributors



Made with contrib.rocks.



Keywords

html-to-docx html to docx html docx office word pptx templates template templater templating report xlsx generation generate generator document generation document creator document automation dynamic document generation microsoft office microsoft word microsoft powerpoint microsoft excel create make Office Open XML OOXML document generation software automated document creation batch document generation document templating typescript ts hebrew arabic rtl right to left

Install

```
npm i @turbodocx/html-to-docx
```

Repository

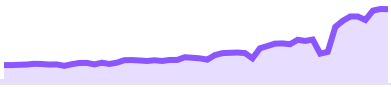
github.com/turbodocx/html-to-docx

Homepage

github.com/TurboDocx/html-to-docx#readme

Weekly Downloads

25,685



Version

1.20.1

License

MIT

Unpacked Size

5.04 MB

Total Files

9

Last publish

22 days ago

Collaborators

Support

Help

Advisories

Status

Contact npm

Company

About

Blog

Press

Terms & Policies

Policies

Terms of Use

Code of Conduct

Privacy