domhandler 

The DOM handler creates a tree containing all nodes of a page. The tree can be manipulated using the domutils or cheerio libraries and rendered using dom-serializer .

Usage

```
const handler = new DomHandler([ <func>
        callback(err, dom), ] [ <obj>
        options ]);
// const parser = new Parser(handler[,
        options]);
```

Available options are described below.

Example

```
const { Parser } =
        require("htmlparser2");
const { DomHandler } =
        require("domhandler");
const rawHtml =
```

```
    "Xyz <script language=
        javascript>var foo =
        '<<bar>>';</script><!--<!--
        Waah! -- -->";
const handler = new DomHandler((error,
        dom) => {
    if (error) {
        // Handle error
    } else {
        // Parsing completed, do
        something
        console.log(dom);
    }
});
const parser = new Parser(handler);
parser.write(rawHtml);
parser.end();
```

Output:

```
[
    {
        data: "Xyz ",
        type: "text",
    },
    {
        type: "script",
        name: "script",
        attribs: {
            language: "javascript",
        },
        children: [
            {
```

```
                name: "font",
            },
        ],
    },
];
```

License: BSD-2-Clause

Security contact information

To report a security vulnerability, please use the Tidelift security contact. Tidelift will coordinate the fix and disclosure.

`domhandler` for enterprise

Available as part of the Tidelift Subscription

The maintainers of `domhandler` and thousands of other packages are working with Tidelift to deliver commercial support and maintenance for the open source dependencies you use to build your applications. Save time, reduce risk, and improve code health, while paying the maintainers of the exact dependencies you use. Learn more.


```
                data: "var foo =
        '<bar>';<",
                type: "text",
            },
        ],
    },
    {
        data: "<!-- Waah! -- ",
        type: "comment",
    },
];
```

Option: `withStartIndices`

Add a `startIndex` property to nodes. When the parser is used in a non-streaming fashion, `startIndex` is an integer indicating the position of the start of the node in the document. The default value is `false`.

Option: `withEndIndices`

Add an `endIndex` property to nodes. When the parser is used in a non-streaming fashion, `endIndex` is an integer indicating the position of the end of the node in the document. The default value is `false`.

Option: `normalizeWhitespace` *(deprecated)*

Replace all whitespace with single spaces. The default value is `false`.

Note: Enabling this might break your markup.

For the following examples, this HTML will be used:

```
<font> <br />this is the text <font></
        font></font>
```

Example: `normalizeWhitespace: true`

```
[
    {
        type: "tag",
        name: "font",
        children: [
            {
                data: " ",
                type: "text",
            },
            {
                type: "tag",
                name: "br",
            },
            {
                data: "this is the text
        ",
                type: "text",
            },
            {
                type: "tag",
                name: "font",
            },
        ],
    },
];
```


Example: `normalizeWhitespace: false`

```
[
    {
        type: "tag",
        name: "font",
        children: [
            {
                data: "\n\t",
                type: "text",
            },
            {
                type: "tag",
                name: "br",
            },
            {
                data: "this is the
        text\n",
                type: "text",
            },
            {
                type: "tag",
```