

```

1 // @ts-nocheck
2
3 // MIT License
4
5 // Copyright (c) 2021 Pine Wu
6
7 // Permission is hereby granted, free of charge, to any person obtaining a co
py
8 // of this software and associated documentation files (the "Software"), to d
eal
9 // in the Software without restriction, including without limitation the righ
ts
10 // to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
11 // copies of the Software, and to permit persons to whom the Software is
12 // furnished to do so, subject to the following conditions:
13
14 // The above copyright notice and this permission notice shall be included in
all
15 // copies or substantial portions of the Software.
16
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
18 // IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
19 // FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL TH
E
20 // AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
21 // LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FR
OM,
22 // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE
23 // SOFTWARE.
24
25 // @filename: renderer.ts
26 // source: https://github.com/shikijs/shiki/blob/ca5a6c60a255fb390776ada849e0
1e1744a916fc/packages/shiki/src/renderer.ts
27 import { FontStyle } from './stackElementMetadata'
28 import { IThemedToken } from './themedTokenizer'
29
30 export interface HtmlRendererOptions {
31   langId?: string
32   fg?: string
33   bg?: string
34 }
35
36 export function renderToHtml(
37   lines: IThemedToken[][],
38   options: HtmlRendererOptions = {}
39 ) {
40   const bg = options.bg || '#fff'
41
42   let html = ''
43
44   html += `

```
${options.langId}</div>`
47 }
48 html += `>`
49
50 lines.forEach((l: IThemedToken[]) => {
51 html += `

```


```

```

63     }
64     html += ``
67     })
68     html += `</span>\n`
69 })
70 html = html.replace(/\n*$/, '') // Get rid of final new lines
71 html += `</code></pre>`
72
73 return html
74 }
75
76 const htmlEscapes = {
77     '&': '&amp;',
78     '<': '&lt;',
79     '>': '&gt;',
80     '"': '&quot;',
81     "'": '&#39;',
82 }
83
84 function escapeHtml(html: string) {
85     return html.replace(/([<>"']/g, (chr) => htmlEscapes[chr])
86 }
87
88 // @filename: highlighter.ts
89 // source: https://github.com/shikijs/shiki/blob/ca5a6c60a255fb390776ada849e01e1744a916fc/packages/shiki/src/highlighter.ts
90 import type {
91     Highlighter,
92     HighlighterOptions,
93     ILanguageRegistration,
94     IShikiTheme,
95     IThemeRegistration,
96     StringLiteralUnion,
97 } from './types'
98 import { Resolver } from './resolver'
99 import { tokenizeWithTheme } from './themedTokenizer'
100 import { renderToHtml } from './renderer'
101
102 import { getOnigasm } from './loader'
103 import { Lang, languages as BUNDLED_LANGUAGES } from './languages'
104 import { Registry } from './registry'
105 import { Theme } from './themes'
106
107 function resolveLang(lang: ILanguageRegistration | Lang) {
108     return typeof lang === 'string'
109         ? BUNDLED_LANGUAGES.find((l) => l.id === lang || l.aliases?.includes(lang))
110         : lang
111 }
112
113 function resolveOptions(options: HighlighterOptions) {
114     let _languages: ILanguageRegistration[] = BUNDLED_LANGUAGES
115     let _themes: IThemeRegistration[] = options.themes || []
116
117     if (options.langs?.length) {
118         _languages = options.langs.map(resolveLang)
119     }
120     if (options.theme) {
121         _themes.unshift(options.theme)
122     }
123     if (!_themes.length) {
124         _themes = ['nord']
125     }
126
127     return { _languages, _themes }
128 }
129
130 export async function getHighlighter(

```

```

131 options: HighlighterOptions
132 ): Promise<Highlighter> {
133     const { _languages, _themes } = resolveOptions(options)
134     const _resolver = new Resolver(getOnigasm(), 'onigasm')
135     const _registry = new Registry(_resolver)
136
137     if (options.paths?.themes) {
138         _registry.themesPath = options.paths.themes
139     }
140
141     if (options.paths?.languages) {
142         _resolver.languagesPath = options.paths.languages
143     }
144
145     const themes = await _registry.loadThemes(_themes)
146     const _defaultTheme = themes[0]
147     let _currentTheme: IShikiTheme | undefined
148     await _registry.loadLanguages(_languages)
149
150     /**
151      * Shiki was designed for VS Code, so CSS variables are not currently supported.
152      * See: https://github.com/shikijs/shiki/pull/212#issuecomment-906924986
153      *
154      * Instead, we work around this by using valid hex color codes as lookups in a
155      * final "repair" step which translates those codes to the correct CSS variables.
156      */
157     const COLOR_REPLACEMENTS: Record<string, string> = {
158         '#000001': 'var(--shiki-color-text)',
159         '#000002': 'var(--shiki-color-background)',
160         '#000004': 'var(--shiki-token-constant)',
161         '#000005': 'var(--shiki-token-string)',
162         '#000006': 'var(--shiki-token-comment)',
163         '#000007': 'var(--shiki-token-keyword)',
164         '#000008': 'var(--shiki-token-parameter)',
165         '#000009': 'var(--shiki-token-function)',
166         '#000010': 'var(--shiki-token-string-expression)',
167         '#000011': 'var(--shiki-token-punctuation)',
168         '#000012': 'var(--shiki-token-link)',
169     }
170
171     function fixCssVariablesTheme(theme: IShikiTheme, colorMap: string[]) {
172         theme.bg = COLOR_REPLACEMENTS[theme.bg] || theme.bg
173         theme.fg = COLOR_REPLACEMENTS[theme.fg] || theme.fg
174         colorMap.forEach((val, i) => {
175             colorMap[i] = COLOR_REPLACEMENTS[val] || val
176         })
177     }
178
179     function getTheme(theme?: IThemeRegistration) {
180         const _theme = theme ? _registry.getTheme(theme) : _defaultTheme
181         if (!_theme) {
182             throw Error(`No theme registration for ${theme}`)
183         }
184         if (!_currentTheme || _currentTheme.name !== _theme.name) {
185             _registry.setTheme(_theme)
186             _currentTheme = _theme
187         }
188         const _colorMap = _registry.getColorMap()
189         if (_theme.name === 'css-variables') {
190             fixCssVariablesTheme(_theme, _colorMap)
191         }
192         return { _theme, _colorMap }
193     }
194
195     function getGrammar(lang: string) {
196         const _grammar = _registry.getGrammar(lang)
197         if (!_grammar) {

```

```

198     throw Error(`No language registration for ${lang}`)
199   }
200   return { _grammar }
201 }
202
203 function codeToThemedTokens(
204   code: string,
205   lang = 'text',
206   theme?: StringLiteralUnion<Theme>,
207   options = { includeExplanation: true }
208 ) {
209   if (isPlaintext(lang)) {
210     return [{ content: code }]
211   }
212   const { _grammar } = getGrammar(lang)
213   const { _theme, _colorMap } = getTheme(theme)
214   return tokenizeWithTheme(_theme, _colorMap, code, _grammar, options)
215 }
216
217 function codeToHtml(
218   code: string,
219   lang = 'text',
220   theme?: StringLiteralUnion<Theme>
221 ) {
222   const tokens = codeToThemedTokens(code, lang, theme, {
223     includeExplanation: false,
224   })
225   const { _theme } = getTheme(theme)
226   return renderToHtml(tokens, {
227     fg: _theme.fg,
228     bg: _theme.bg,
229   })
230 }
231
232 async function loadTheme(theme: IShikiTheme | Theme) {
233   await _registry.loadTheme(theme)
234 }
235
236 async function loadLanguage(lang: ILanguageRegistration | Lang) {
237   const _lang = resolveLang(lang)
238   _resolver.addLanguage(_lang)
239   await _registry.loadLanguage(_lang)
240 }
241
242 function getLoadedThemes() {
243   return _registry.getLoadedThemes()
244 }
245
246 function getLoadedLanguages() {
247   return _registry.getLoadedLanguages()
248 }
249
250 function getBackgroundColor(theme?: StringLiteralUnion<Theme>) {
251   const { _theme } = getTheme(theme)
252   return _theme.bg
253 }
254
255 function getForegroundColor(theme?: StringLiteralUnion<Theme>) {
256   const { _theme } = getTheme(theme)
257   return _theme.fg
258 }
259
260 return {
261   codeToThemedTokens,
262   codeToHtml,
263   getTheme: (theme: IThemeRegistration) => {
264     return getTheme(theme)._theme
265   },
266   loadTheme,
267   loadLanguage,

```

[illegible]