Ein Preprocessor für AsciiDoc

Als PDF lesen

In AsciiDoc gibt es keine Möglichkeit, eigene Makros zu definieren. Makros erzeugen Testblöcke, die über ein *target* und *attributes* parametrisiert werden können.

Beispiele für vorhandene Makros sind

```
include::Example.cs[tag=ExampleConstructor,indent=0]
image::image.jpg[250,150,float=left,scaledwidth=25%]
```

Allgemein haben Makros den Aufbau `<name>::<target>[<attrlist>]`. Makros auf Blockebene werden mit `::` definiert, inline Makros mit `:`. Es gibt allerdings keine Möglichkeit, eigene Makros zu definieren (mit Ausnahme von in Roby geschriebenen Modulen). Da AsciiDoc ein textbasierendes Format ist, kann jedoch ein Programm vorgeschalten werden, das folgendes macht:

1. Einlesen der Textdatei
2. Suchen nach Strings mit dem Aufbau
   `^(?<name>[a-zA-Z0-9-_])::(?<target>)?\[(?<attributes>[^\]]*)\]`
   Dieser Ausdruck erkennt alle Blockmakros, die am Zeilenanfang stehen.
3. Ersetzen durch anderen, im Programm generierten, Inhalt.

Verwenden des bereitgestellten C# Preprocessors

Im Verzeichnis *Preprocessor* ist eine allgemeine C# Implementierung, die die obigen Schritte schon umsetzt. Über eine übergebene Funktion kannst du Methoden für einzelne Makros definieren und das Programm so erweitern. Die Datei *Proram.cs* ist so aufgebaut:

```
public static async Task<int> Main(string[] args)
{
    try
    {
        var preprocessor = AsciidocPreprocessor.FromFile(args[0]);                ①
        preprocessor.AddMacroProcessor("example_macro", ExampleMacroProcessor);   ②
        var newContent = await preprocessor.Process();                             ③
        Console.WriteLine(newContent);
        return 0;
    }
    catch (Exception e)
    {
        Logger.LogError(e.InnerException?.Message ?? e.Message);
        return 1;
    }
```

```
}

static string ExampleMacroProcessor
    (string target, Attributes attributes, Dictionary<string, string> globalVariables)
{
    return @$"
[source,html]
----
<img src=""{target}"" width=""{attributes[0]}"" height=""{attributes[1]}"" style=
""width: {attributes["css_width"]}"">
----
";
}
```

① Die zu lesende AsciiDoc Datei wird als Argument übergeben.

② Die Methode `AddMacroProcessor` erwartet 2 Parameter. Der erste Parameter ist der Name des Makros, wie er 1:1 in der AsciiDoc Datei verwendet wird. Die 2 Funktion ist eine Funktion, die 3 Parameter bekommt (siehe unten).

③ Hier wird die Verarbeitung gestartet.

Es gibt auch eine Methode `AddAsyncMacroProcessor` für einen `async` Processor.

Schreiben eines eigenen Macro Processors

Die Methode `ExampleMacroProcessor` ist ein Beispiel für eine solche Erweiterung. Sie bekommt 3 Parameter:

target

Wert des Targets im Makro. Beispiel: `my_macro::Example.cs[tag=ExampleConstructor,indent=0]` liefert *Example.cs*

attributes

Attributklasse, die den einfachen Zugriff auf Attribute bereitstellt. Der Indexoperator ist überladen. Ist der Index vom Typ `int`, wird das Attribut an der entsprechenden Stelle zurückgegeben (0 basierend).

Ist der Index vom Typ `string`, wird der Wert eines Attributes mit Namen zurückgegeben.

Beispiel `my_macro::imagefile.jpg[250,150,float=left,scaledwidth=25%]`

- **attributes[0]** 250
- **attributes[1]** 150
- **attributes[2]** float=left
- **attributes[3]** scaledwidth=25%
- **attributes["float"]** left
- **attributes["scaledwidth"]** 25%

Du kannst über das Property `AttributesArray` und `NamedAttributes` direkt auf das darunterliegende Array bzw. Dictionary zugreifen.

globalVariables

Ein Dictionary, in dem alle globalen Variablen (z. B. `:author: value`) abrufbar sind.

Aufruf

Erstelle in einem Verzeichnis eine AsciiDoc Datei. Kopiere dann das Verzeichnis *Preprocessor* in dieses Verzeichnis. In der Konsole kannst du dann mit 2 Varianten arbeiten:

Schreiben der Zwischendatei

```
dotnet run --project Preprocessor my_file.adoc > my_file_processed.adoc
convert_adoc.cmd my_file_processed.adoc my_file_processed.pdf
```

Diese Variante hat den Vorteil, dass du die erzeugte Datei *my_file_processed.adoc* kontrollieren und ggf. händisch anpassen kannst.

Direkte Verarbeitung über die Pipe

Möchtest du keine Zwischendatei anlegen, kann auch über die Pipe gearbeitet werden. Das Symbol `-` bedeutet, dass *convert_adoc.cmd* die Daten von stdin liest.

```
dotnet run --project Preprocessor -- my_file.adoc | convert_adoc.cmd -
my_file_processed.pdf
```

Das Makro `chatgpt_prompt`

Im Preprocessor ist auch ein Makro `chatgpt_prompt` integriert. In der Datei Program.cs wird dieser geladen, wenn eine Datei *chatgpt_key.txt* vorhanden ist.

Dieser Preprocessor braucht einen API Key von OpenAI, der allerdings kostenpflichtig ist. Kopiere diesen Key in die Datei *chatgpt_key.txt*. Die Datei muss ich im Ordner des AsciiDoc Dokumentes befinden.

Schließe die Datei über eine *.gitignore* Datei aus, wenn du das Dokument in einem Repository verwaltest.

Das Makro hat folgenden Aufbau

```
chatgpt_prompt::["Analysiere rhetorische Stilmittel zu folgendem Artikel:
https://www.diepresse.com/19123344/dividenden-wurde-ktm-ausgeraeumt
",max_tokens=8192,temperature=0,resolve_links=true,save_message=true]
```

Das Target ist leer. Das erste Attribut ist der Prompt. Zusätzlich können mehrere Attribute definiert

werden.

max_tokens

Die Abrechnung basiert über sogenannte Token. 1000 Token entsprechen ca. 1.5 Cent und umfasst 750 Worte. Es gibt pro Modell eine Maximalgrenze von rd. 10000 Token. Ist der Wert zu hoch, liefert die API *bad request*.

temperature

Steuert, wie "kreativ" das Modell ist.

- **Niedrige Werte (z. B. 0.0 bis 0.3):** Das Modell wird deterministischer und fokussierter. Es bevorzugt die wahrscheinlichste Antwort und reduziert die Variation. Ideal für Aufgaben, bei denen Genauigkeit und Konsistenz entscheidend sind, z. B.:
  - Mathematik
  - Faktenbasierte Antworten
  - Code-Generierung
- **Mittlere Werte (z. B. 0.4 bis 0.7):** Das Modell wird kreativer, bleibt aber noch weitgehend kontrolliert. Nützlich für Anwendungen, die sowohl Kreativität als auch Relevanz erfordern, z. B.:
  - Schreiben von E-Mails
  - Generierung von Blog-Artikeln
  - Content-Erstellung mit klaren Vorgaben
- **Hohe Werte (z. B. 0.8 bis 1.0 oder höher):** Das Modell wird viel kreativer und erzeugt originelle, oft überraschende Inhalte. Antworten können inkonsistenter sein oder weniger präzise, aber dafür unerwarteter. Gut geeignet für:
  - Brainstorming
  - Kreatives Schreiben
  - Anwendungen, bei denen Vielfalt bevorzugt wird

resolve_links=true

Über die API können keine Hyperlinks aufgelöst werden. Mit dieser Option fordert das C# Programm über die Klasse `HttpClient` die Inhalte an, löscht die HTML Syntaxelemente und ersetzt den Link durch den Text, bevor er an ChatGPT gesendet wird. Dies funktioniert nur für Links, die Textdokumente zurückgeben.

Schreibe die Links in eine eigene Zeile und ohne Punkt (.) am Ende. Alles von _http(s)?:// bis zur nächsten Leerstelle wird als Link interpretiert.

save_message=true

Die API ist *stateless*, das bedeutet dass der Prompt nicht das Ergebnis der vorigen Prompts zur Verarbeitung einschließt. Mit dieser Option wird die Antwort gespeichert und automatisch bei nachfolgenden Requests zum Prompt angefügt.

Verwende die Option nur bei Prompts, die nachfolgende Werte beeinflussen. Sie

> vergrößert den Prompt, die Anfragen kosten dann mehr Tokens.

Folgender Prompt wird immer mitgeliefert:

> Du spricht mit einem Computerprogramm, das nur AsciiDoc versteht. Es dürfen keine anderen Inhalte von dir übermittelt werden. Source code soll in einen [source] Block mit der entsprechenden Programmiersprache stehen und mit der Option linenums versehen werden. Achte darauf, dass Callouts immer in spitzen Klammern stehen. PlantUML Diagramme sind in einem Codeblock mit [plantuml,,svg] ohne Callouts zu übermitteln. Es darf kein Titel (mit einem = beginnend) übermittelt werden.

Der Cache

Die Promots und die Antworten werden in die Datei *chatgpt_cache.json* geschrieben. Wenn der Prompt unverändert ist, wird beim neuerlichen Verarbeiten der Datei die Antwort aus dem Cache genommen. Falls du Prompts mit der Option *save_message* änderst, oder nicht erwartete Antworten erhältst, empfiehlt es sich, diese Datei zu löschen.

Beispiele

Um die Verwendung zu demonstrieren, gibt es folgende Beispiele:

[preprocessor_simple_test.adoc]

Ruft das vordefinierte Makro example_macro im Preprocessor Programm auf. Aufruf:

```
dotnet run --project Preprocessor -- preprocessor_simple_test.adoc |
convert_adoc.cmd - preprocessor_simple_test.pdf
```

[preprocessor_test.adoc]

Verwendet das ChatGPT Makro, um eine Datenbank Aufgabenstellung zu bearbeiten. API Key erforderlich! Aufruf:

```
dotnet run --project Preprocessor -- preprocessor_test.adoc | convert_adoc.cmd -
preprocessor_test.pdf
```

[kommentar.adoc]

Verwendet das ChatGPT Makro, um einen Kommentar auf Basis der URL eines Artikels zu verfassen. API Key erforderlich! Aufruf:

```
dotnet run --project Preprocessor -- kommentar.adoc | convert_adoc.cmd -
kommentar.pdf
```

Du kannst statt der Pipe auch die Ausgabe von `dotnet run` in eine Datei umleiten. Dann kannst du das Ergebnis des Preprocessors als AsciiDoc Dokument betrachten und ggf. nachbearbeiten.