Gherkin Preprocessor

Overview

Preprocessing can reduce redundancy in feature files, thus making them more maintainable. The preprocessor transforms a file using some preprocessing features.

Preprocessing features include:

- Define symbol with replacement string
- Define symbol as result of a calculation
- Create local functions that can be used in calculations
- Include common files (e.g. background) in multiple feature files

Define Symbols

By defining a symbol with #define and a replacement string, the symbol will be replaced wherever it appears in the file. Example:

Scenario: One Define is Replaced

When processing:

```
First line
#define X 12
Value is X
Last line
```

Then result is:

```
First line
Value is 12
Last line
```

Define Symbols with Calculation

By defining a symbol with #define, an equals sign, and a replacement string, the replacement string will be transformed by:

- Replacing symbols that have been defined with their replacement string
- Performing calculations using double arithmetic with optional int

- Performing local functions

Replacing symbols and performing calculation

Scenario: Expression

When processing:

```
#define X 1
#define Y = X + 3
Feature: Test Feature
Scenario: One
Given value is X
When processed
Then result is Y
```

Then result is:

```
Feature: Test Feature
Scenario: One
Given value is 1
When processed
Then result is 4
```

Local Functions can be added for calculations

Local functions can be defined and used in calculations. Local function signatures should be:

```
std::string function(std::string parameters[]);
```

Here's an example. GET_TODAY(), TODAY_WITH_OFFSET() and DATE_WITH_OFFSET() are local functions that have been added to the preprocessor. They use a DD-MON-YYYY format. Parameters are passed in as strings and the return from process() is replaced in the #define string.

Scenario: Difference in Days

When processing:

```
#define NEW_DATE = DATE_WITH_OFFSET(6-JUN-2020, 3)
Difference is NEW_DATE
```

Then result is:

```
 Difference is 9-JUN-2020
```

Include Files

Files containing Gherkin statements or #defines can be included in a feature files using a #include. The lines in the file replace the #include line. Include files can be nested – they can contain other includes.

Scenario: Process a simple include

Given file SampleInclude.txt exists with:

```
Something to include
```

When processing:

```
First line
#include "SampleInclude.txt"
Last line
```

Then result is:

```
First line
Something to include
Last line
```

CSV Include Files

Include files which have a CSV suffix will be transformed into table format.

Given include file TestCSV.csv exists with:

```
FieldOne, FieldTwo
1,2
```

When Processing:

```
Feature: Test Feature
Scenario Outline: Two
Given value is <FieldOne>
When processed
Then result is <FieldTwo>
Examples:
#include "TestCSV.csv"
```

Then result is:

```
Feature: Test Feature
Scenario Outline: Two
```

```
Given value is <FieldOne>
When processed
Then result is <FieldTwo>
Examples:
|FieldOne|FieldTwo|
|1|2|
```

Running the Preprocessor

The processor is executing with:

```
gherkinpreprocessor  <in-file> <out-file>
```

If you want to store #include files in a common directory, you can use < > instead of " "

```
#include <include-file>
```

and set the directory with

```
gherkinpreprocessor  <in-file> <out-file> -system  <directory>
```

If you include on the first line of a file:

```
@needs_preprocessing
```

It will be removed in the output file. The input file will be preprocessed regardless of whether this appears or not. This is just if you want to check what files need to be preprocessed, instead of processing every one.

Normally #defines cannot be redefined. You can allow them to be re-defined by having on the first line of the file:

```
@allow_redefines
```