# Test Table Writer

The Test Table Writer was written as a method to generate UI Automation tests for OpenConsole. UI Automation has a lot of corner cases, so we developed this workflow to simplify storing and updating these test cases (particularly revolving around movement).

# How to use it

1. Update `UiaTests.csv`:
    - This file is used to store the tests in a compact format. The defined columns include...
        - Degenerate: is this a degenerate range?
        - Position: see the position chart below
        - TextUnit: what text unit to move by
        - MoveAmount: how many times to move
        - Start: the start endpoint of the text range. Represented by a variable name used to signify a position in the buffer. See the variable heuristics section below.
        - End: the start endpoint of the text range. Represented by a variable name used to signify a position in the buffer. See the variable heuristics section below.
        - Result_MoveAmount: the expected amount to have moved by
        - Result_Start: the expected position of the start endpoint after executing the move operation
        - Result_End: the expected position of the end endpoint after executing the move operation
        - Skip: skip the test. Can be used for failing tests.
    - Each row represents a new test in a compact format.
    - Use the position chart and the variable heuristics below to add more tests easily.
2. Run `GenerateTests.ps1`
    - `GenerateTests.ps1` will load `UiaTests.csv` and export the tests and any necessary variables to "src\interactivity\win32\ut_interactivity_win32\GeneratedUiaTextRangeMovementTests.g.cpp".
3. Build and run the tests
    - Build UiaTextRangeTests
    - Go to bin\x64\Debug
    - Run `clear; .\TE.exe /name:*UiaTextRangeTests*GeneratedMovementTests*` `.\Conhost.Interactivity.Win32.Unit.Tests.dll` in PowerShell
4. If the tests pass, upload any changes to `UiaTests.csv` and `GeneratedUiaTextRangeMovementTests.g.cpp`. Be sure to run the code formatter.

# Helpful tips

- How to verify a test is authored correctly
    - use MS Word to generate some text (try typing "=lorem(5,5)" then pressing enter to generate text)
    - use Accessibility Insights to run the UIA API on MS Word's ITextProvider
        - if you create a selection (or move the cursor), then tell Accessibility Insights to get the "Selection" range (or refresh it), you can easily set up a test case
- Run the tests via Visual Studio
    - In the Solution Explorer, right-click Interactivity.Win32.Tests.Unit and select "Set as Startup Project"

- right-click it again and select "Properties"
- In the Debugging section, set..
  - "Command" --> `$(OutDir)/TE.exe`
  - "Command Arguments" --> `$(TargetPath) /name:*uiatextrange*generated* /inproc`

# Position chart

The text buffer is assumed to be partially filled. Specifically, the top half of the text buffer contains text, and each row is filled with 10 segments of alternating text. For visualization, the ascii diagram below shows what the text buffer may look like.

```
+------------------------------+
|1XX   XXX   X2X   XXX   XXX   |
|XXX   XXX   XXX   XXX   XXX   |
|XXX   XXX   X3X   XXX   XXX   |
|XXX   XXX   XXX   XXX   XXX   |
|XXX   XXX   X4X   XXX   XX5   |
|6                             |
|                              |
|              7               |
|                              |
|                             8|
+------------------------------+
9
```

The following positions are being tested:

1. `origin` : buffer origin
2. `midTop` : middle of the top line
3. `midHistory` : middle of the history
4. `midDocEnd` : middle of the last line of text
5. `lastCharPos` : last character in the buffer
6. `docEnd` : one past the last line of text
7. `midEmptySpace` : middle of the empty space in the buffer
8. `bufferEnd` : end of the buffer
9. `endExclusive` : exclusive end of the buffer

This is intended to provide adequate testing coverage for GH#6986.

# Variable Heuristics

Each position above already has a predefined variable name. However, a few heuristics are used to define new variables based on the standard variables above.

- `<name>Left` : the left-most position on the same line as `<name>`
- `<name>P<number>C` , `<name>M<number>C` :
  - `<name>` : start at the position of `<name>`

    - `P` (or `M`): move forwards (aka "plus") by a certain amount (`M` is used to move backwards [aka "minus"])
    - `<number>`: how much to move forwards by
    - `C`: move by character. For simplicity, assumes that each character is one-cell wide.
- `<name>P<number>L`, `<name>M<number>L`:
    - same as above, except move by line. For simplicity, assumes that you won't hit a buffer boundary.
- `segment#L<name>`
    - This is mainly used for word navigation to target a segment of text in the buffer.
    - `segment#` refers to the beginning of a segment of text in a row. The leftmost run of text is `segment0`, whereas the rightmost run of text is `segment4`.
    - `L<name>` refers to the line number we're targeting, relative to the `<name>` variable. For example, `Lorigin` means that the y-position should be that of `origin`.
    - Overall, this allows us to target the beginning of segments of text. `segment0Lorigin` and `segment0LmidTop` both refer to the beginning of the first segment of text on the top line (aka `origin`).

# Helpful terms and concepts

- *degenerate*: the text range encompasses no text. Also, means the start and end endpoints are the same.
- *TextUnit*: a heuristic for how much to move by. Possible values include `TextUnit_Character`, `TextUnit_Word`, and `TextUnit_Line`. See https://docs.microsoft.com/en-us/windows/win32/winauto/uiauto-uiautomationtextunits for more details.