The texassert package*

Hanson Char
hanson.char@gmail.com

November 9, 2024

Abstract

An assertion library for unit testing in plain TeX.

1 Introduction

This package emerged from a desire to explore l3build and literate programming. It provides a collection of Plain TEX macros that I originally used for unit testing, now converted into a .dtx file, allowing for regeneration of the original source files from the literate code.

All .tex files in this package are written in Plain TeX, offering a simple mechanism for performing assertions in unit testing Plain TEX macros. I hope you find it useful. *Profitez!*

2 Usage Examples

2.1 Length Assertions

To unit test the \lengthof macro in this library, for example, we can create a file length-tests.tex with something like:

```
% Import the necessary macros
\input import \import{lengthof} \import{assert}

% Length of an empty string is zero
\lengthof{} \asserteq\the\length=0

% Length of ‘0’ is one
\lengthof{0} \asserteq\the\length=1

% Length of ‘12.3456’ is seven
\lengthof{12.3456} \asserteq\the\length=7

% Summary of the assertions made so far
\assertionsummary
\bye
```

Compile it with a TEX engine, e.g. pdftex length-tests.tex, we get an output file length-tests.pdf with something like:

Assertion Summary: 3/3 assertions passed i.e. 0/3 assertions failed.

*This document corresponds to texassert v0.0.2, dated 2024/11/07.

2.2 (More Examples ...)

TODO

3 Source Repository

https://github.com/hansonchar/texassert

4 Useful Resources

Not so much related to the library provided by this package per se, but some commands and external resources which I found directly useful or necesssary for the purpose of *constructing* this package per se.

1. Examples in the `l3build` repository. The `simple-tree` example in particular.
2. `texdoc l3build` – information directly related to `l3build`.
3. `texdoc doc` – the `doc` package used by `l3build` implicitly.
4. `texdoc docstrip` – the `docstrip` package used by `l3build` implicitly.
5. `texdoc source2e` – information related to various macros that are or can be used in a `.dtx` file.
6. `texdoc dtxtut` – Scott Pakin. *How to Package Your LaTeX Package.* January 21, 2024. (I had lots of *Aha!* moments in reading this.)
7. Michel Gossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion.* Addison Wesley, Reading, Massachusetts, October 1, 1994. ISBN 0-201-54199-8.

5 Implementation

`import.tex` Contains `\import`.

`\import` Used to prevent the same file from being `\input` more than once.

```
\def\import#1{%
  \expandafter\ifx\csname import:#1\endcsname\relax
    \input #1
    \expandafter\gdef\csname import:#1\endcsname{}%
    % Imported #1\par
  \fi
}
```

`common.tex` Contains common code and configuration used in this library.

```
\showboxdepth=\maxdimen \showboxbreadth=\maxdimen

\newtoks\result \newtoks\tokstemp
\newcount\n
\newcount\integer
```

```

\def\true{\let\bool=\iftrue}
\def\false{\let\bool=\iffalse}
```

`\debug` Writes a line of debug message immediately to the terminal and the log file when debugging is enabled (via `\debugtrue` which is the default).

```
\newif\ifdebug
\debugtrue
\def\debug#1{\ifdebug \immediate\write16{[DEBUG] #1}\fi}
```

`\ifEmpty` Checks if the given parameter is empty.

```
\newif\ifempty
\def\checkifempty#1{{\expandafter\def\expandafter\input\expandafter{#1}%
  \global\ifx\input\empty \emptytrue\else\emptyfalse\fi}}

% Assigning \iffalse to \then and use as a parameter delimiter is critical
% in making the if-macros skippable.
% Source: https://tug.org/TUGboat/tb45-1/tb139wermuth-isint.pdf
\let\then=\iffalse
\def\ifEmpty#1\then{%
  \checkifempty{#1}\ifempty
}
```

`\ifUndefined` Checks if the given control sequence is undefined.

```
\long\def\ifUndefined#1\then{{%
  \edef\x{\meaning#1}%
  \let\e=\escapechar \escapechar=-1
  \edef\y{\string\undefined}\escapechar=\e
  \def\true{\iftrue}\def\false{\iffalse}%
  \def\next{\expandafter\expandafter\expandafter
    \aftergroup\ifx\x\y\true\else\false\fi}\next}}
```

`\ifDefined` Checks if the given control sequence is defined.

```
\long\def\not#1#2\then{#1#2\then \false \else \true \fi \bool}
\long\def\ifDefined#1\then{\ifUndefined#1\then \false \else \true\fi \bool}
```

`lengthof.tex` Contains the code used to find out the length of a given string.

`\lengthof` Computes the length of the given string parameter.

```
\input import \import{common}

\newcount\length
\edef\temp{\the\catcode`@}\catcode`@=11

\def\lengthof#1{\length=0 %
  \bgroup
    \edef\lengthof@input{#1}%
    \ifEmpty\lengthof@input\then
      \let\next=\relax
    \else
      \def\next{\expandafter\lengthofA\lengthof@input\eot}%
    \fi
    \next
  \egroup
}
\def\lengthofA#1#2\eot{\global\advance\length by1 %
  \ifEmpty#2\then
    \let\next=\relax
  \else
    \def\next{\lengthofA#2\eot}%
  \fi
  \next
}

\catcode‘@=\temp % restore the original catcode for @
```

`checkeq.tex` Contains the code used to check if two given strings are equal.

`\checkeq` Used to check if two given strings are equal. Assume no space in the strings.

```
\input import \import{lengthof}

\newif\ifeq
\edef\temp{\the\catcode‘@}\catcode‘@=11

\global\eqtrue
% Assume no spaces
\def\checkeq#1#2{{%
  \edef\checkeq@fstparam{#1}%
  \edef\checkeq@sndparam{#2}%
  \lengthof\checkeq@fstparam \edef\lena{\number\length}%
  \lengthof\checkeq@sndparam \edef\lenb{\number\length}%
  \ifx\lena\lenb
    \ifnum\length=0
      \global\eqtrue \let\next=\relax
    \else
      \expandafter\expandafter\expandafter
        \def\expandafter\expandafter\expandafter
          \next\expandafter\expandafter\expandafter
            {\expandafter\expandafter\expandafter
              \checkeqA\expandafter\checkeq@fstparam
                \expandafter\eot\checkeq@sndparam\eot}%
    \fi
  \else
    \global\eqfalse \let\next=\relax
  \fi
  \next
}}
\def\checkeqA#1#2\eot#3#4\eot{%
  \if#1#3{}% the trailing ‘{}%’ is necessary to avoid adding extra spaces
    \ifx\relax#2\relax
      \global\eqtrue \let\next=\relax
    \else
      \def\next{\checkeqA#2\eot#4\eot}%
    \fi
  \else
    \global\eqfalse \let\next=\relax
  \fi
  \next
}

\catcode‘@=\temp % restore the original catcode for @
```

assert.tex Contains the code used for assertion purposes.

```
\input import \import{checkeq}

\ifDefined\ProvidesPackage\then
  \ProvidesPackage{texassert}
\fi

\newcount\countassertions
\newcount\countassertionspassed
\newcount\countassertionsfailed
\newif\ifassertmessageonly
\edef\temp{\the\catcode‘@}\catcode‘@=11

\let\assertDone=\iffalse
\def\unexpected{\toks0={unexpected!}}
\def\expected{\toks0={expected}}
\def\assert{\asserteq\the\toks0={expected}}
\def\assertTrue#1\assertDone{#1\then \expected \else \unexpected\fi \assert}
\def\assertFalse#1\assertDone{#1\then \unexpected \else \expected\fi \assert}

\def\resetassertions{%
  \countassertions=0
  \countassertionspassed=0
  \countassertionsfailed=0
}
```

\asserteq Asserts that the two given string are equal, taking catcode into account.

```
\def\asserteq#1=#2{{%
  \global\advance\countassertions by1
  \edef\assert@a{#1}%
  % \message{assert@a: [\meaning\assert@a]}%
  \edef\assert@b{#2}%
  % \message{assert@b: [\meaning\assert@b]}%
  \ifx\assert@a\assert@b\relax\relax
    \global\advance\countassertionspassed by1
  \else
    \global\advance\countassertionsfailed by1%
    \message{...}%
    \def\errmsg{*** assertion (\the\countassertions) failure:
      ‘#1’ not equal ‘#2’ ***}%
    \message{\errmsg}%
    \ifassertmessageonly\else
      \medbreak
      \indent\indent{\errmsg}%
      \medbreak\fi
  \fi
}}
```

\asserteqnocat Asserts that the two given string are equal, disregarding any catcode differences.

```
\def\asserteqnocat#1=#2{{%
```

```
  \global\advance\countassertions by1
  \edef\assert@a{#1}%
  % \message{assert@a: [\meaning\assert@a]}%
  \edef\assert@b{#2}%
  % \message{assert@b: [\meaning\assert@b]}%
  \checkeq\assert@a\assert@b
  \ifeq
    \global\advance\countassertionspassed by1
  \else
    \global\advance\countassertionsfailed by1
    \message{...}%
    \def\errmsg{*** assertion (\the\countassertions) failure:
      ‘#1’ not equal ‘#2’ ***}%
    \message{\errmsg}%
    \ifassertmessageonly\else
      \medbreak
      \indent\indent{\errmsg}%
      \medbreak\fi
  \fi
}}
```

\assertneq Asserts that the two given string are not equal, taking catcode into account.

```
\def\assertneq#1=#2{{%
  \global\advance\countassertions by1
  \edef\assert@a{#1}%
  % \message{assert@a: [\meaning\assert@a]}%
  \edef\assert@b{#2}%
  % \message{assert@b: [\meaning\assert@b]}%
  \ifx\assert@a\assert@b\relax\relax
    \global\advance\countassertionsfailed by1%
      \message{...}%
      \def\errmsg{*** assertion (\the\countassertions) failure:
        ‘#1’ equal ‘#2’ ***}%
      \message{\errmsg}%
      \ifassertmessageonly\else
        \medbreak
        \indent\indent{\errmsg}%
        \medbreak\fi
  \else
    \global\advance\countassertionspassed by1
  \fi
}}
```

\assertionsummary Typesets a summary of the assertions made. Then reset to a state as if no assertion has been made.

```
\def\assertionsummary{{%
  \def\sp{ }%
  \def\summary{%
    Assertion Summary: \the\countassertionspassed/\the\countassertions\sp
      assertions passed i.e.
    \the\countassertionsfailed/\the\countassertions\sp assertions failed.}%
  \message{\summary}%
  \ifassertmessageonly\else
    \medbreak
201     \summary
202   \fi}\resetassertions}
203
204 \catcode`@=\temp % restore the original catcode for @
```

`texassert.sty` Used for packaging purposes.

```
205 \input{assert}
```

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Change History

v0.0.1 – 2024-11-05

General: Initial version 1

v0.0.2 – 2024-11-07

General: Migrate source files to `texassert.dtx` 1