The getargs Package

Provides the \getargs list parsing macro and associated configurations

Steven B. Segletes
steven.b.segletes.civ@mail.mil

May 20, 2016
v1.01

This is the 3rd incarnation of the \getargs macro in some form or another. The first was in the stringstrings package, but it is slow, can only parse an expanable list with space delimiters, and brings all the baggage of my bloated stringstrings package with it. An much more efficient form was reformulated in my readarray package, under the name \getargsC, but like its predecessor, it can only parse with space delimiters. So here, I reintroduce the \getargs macro, both efficient and capable of parsing based on any delimiter character desired. In addition, it allows for the root-name of the parsed-argument list to be customized. It is my intent to eventually have the parsers of those other packages point to this package, to achieve uniformity.

\setparsechar

This package is very short, providing three user macros plus one diagnostic macro. The macro \setparsechar, invoked as

```
\setparsechar{<parse-character>}
```

provides the means to designate the delimiter character by which the parser breaks the argument list into pieces. Because of the popularity of the csv format, the parsing delimiter character is set to a comma (,), when the package is initially invoked. The parsing delimiter character can also be set to a space with \setparsechar{ }. If the parsing delimiter is not a space, however, then leading and trailing spaces in the parsed argument list are retained (note that, in the standard LaTeX fashion, multiple adjacent spaces are parsed as a single input space).[1]

\getargs

The primary user macro of this package is \getargs, which is used to parse a delimited argument list. The invocation syntax is

```
\getargs{<argument-list to parse using parse-character>}
```

Like prior formulations of the \getargs macro, the result of the parsing are twofold:

1. The number of items parsed from the argument-list is stored, via \xdef, in the macro named, by default, \narg.

[1] The current parse character is stored in \getargsparsechar, for reference only. However, a new parse character can only be set with an invocation of \setparsechar.

2. The actual tokens of the parsed argument-list are placed in a series of macros, which are, by default, named `\argi`, `\argii`, `\argiii`, `\argiv`, ..., in roman-numeral naming fashion.

`\setparserootname`

The root name of the macros into which the parsed argument-list are placed can itself be designated by the user, using the `\setparserootname` macro. This macro is invoked as

```
\setparserootname[<root-name>]
```

The default root-name employed is "arg", if the optional argument is not specified. When the root-name of the `\getargs` parser is changed, not only are the subsequently parsed arguments stored in new macro names `\<root-name>i`, `\<root-name>ii`, `\<root-name>iii`, *etc.*, but the total number of arguments parsed is no longer stored in `\narg`, but now in `\n<root-name>`.

`\showargs`

The final macro provided is the `\showargs` diagnostic macro. It can be invoked with or without the `[x]` optional argument, which determines whether the parsed argument list is presented as tokens (the default, via `\detokenize`[2]) or, if it is presented in expanded form, when the `[x]` optional argument is employed. **Be forewarned that not all parsed tokens will present without error in expanded form.** If the parsing separates a macro from its arguments, or if it separates some types of opening and closing delimiters (paired `$`, for example), then errors will be generated, not from the parsing, but from an attempt to `\showargs` the result in expanded `[x]` form.

The `\showargs` invocation will first list the number of items most recently parsed from the input list associated with the current `<root-name>`. It will provide that root-name of the parsed items and whether or not the parsed items that follow are presented as raw tokens or in expanded form. Finally, it will sequentially list the parsed items between vertical dividing rules, in a line breakable way.

This version of `\getargs` and `\showargs` will overwrite any existing version that is already loaded (for example, from the stringstrings package), without providing warning or error. If that is the intent of the user, then make sure the getargs package is loaded after the other conflicting packages.

[2] Dont forget that LaTeX's `\detokenize` always presents macro names with a trailing space, regardless of whether that space actually exists in the parsed argument.

Examples

- **The difference between expanded and raw-token `\showargs`**

```
\def\myname{Steven Segletes}
\getargs{Signed/dated as follows, \myname, \today}
\showargs[x]\par
\showargs
```

3 \arg... items (expanded): |Signed/dated as follows| Steven Segletes| May 20, 2016|.

3 \arg... items (tokens): |Signed/dated as follows| \myname | \today |.

When presented in expanded form, the macros are fleshed out with their expansions. However, one can see that the original tokens remain in the tokenized presentation of `\argii` and `\argiii`.

- **The behavior of leading/trailing spaces (with a non-space parse character)**

```
\getargs{A, A, A ,  A  }
\showargs
```

4 \arg... items (tokens): |A| A| A | A |.

Note above, in the expression of `\argiv`, that multiple spaces in the input are parsed, according to the LaTeX standard, as single spaces. Thus, `\argiii` and `\argiv` are functionally identical.

- **Changing the parsing character**

```
\setparsechar{&}
\getargs {y&\frac{x}{y}&(x_0-y_0)^3}
$\showargs[x]$
```

$3\backslash arg\ldots items(expanded):|y|\frac{x}{y}|(x_0-y_0)^3|$.

Note that the parsing of math expressions did not take place in math mode. However, as long as they are presented in math mode, all is well.

The changed parsing character will remain `&` until subsequently changed (or until the group ends, if it was changed within a group).

- **Space as the parsing character**

```
\setparsechar{ }
\getargs{A B  C   D}\showargs
```

4 \arg... items (tokens): |A|B|C|D|.

When a space-character is used as the parsing character, one can see above that multiple leading/trailing spaces are absorbed in the parsing, so that `\argiii` is left as a simple "C", despite being surrounded by multiple spaces.

- **Parsed macros are not expanded at time of parsing**

```
\setparsechar{&}
\def\A{Alpha $\alpha$}
\getargs {parameter (1)& &parameter {\A}}
\argiii{} VS. \def\A{Beta $\beta$}\argiii
```

parameter Alpha α VS. parameter Beta β

Because `\argiii` is stored as this: "`parameter {\A }`", it follows that after `\A` is redefined, the redefinition carries over into the expansion of `\argiii`.

- **Nested parsing**

```
\setparserootname[ROW]
\setparsechar{\\}
\getargs {A_{11} & A_{12} & A_{13}\\ A_{21} & A_{22} & A_{23}}
\setparsechar{&}
\setparserootname[ROWiCOL]
\expandafter\getargs\expandafter{\ROWi}
\setparserootname[ROWiiCOL]
\expandafter\getargs\expandafter{\ROWii}
$(\ROWiCOLi) (\ROWiCOLii) (\ROWiCOLiii)$\\
$(\ROWiiCOLi) (\ROWiiCOLii) (\ROWiiCOLiii)$
```

$(A_{11})(A_{12})(A_{13})$
$(A_{21})(A_{22})(A_{23})$

Above, I perform all the tasks manually, but it is not hard to set it in a loop based on the respective values of `\nROW` (2), `\nROWiCOL` (3), and `\nROWiiCOL` (3). This is a powerful way to retieve and store all the elements of a matrix in a structured way. Note that `\ROWi` and `\ROWii` each had to be expanded exactly once in order to be digested as input to `\getargs`.

Source Code

```
\def\getargsversionnumber{v1.01}
\ProvidesPackage{getargs}
[2016/05/20 \getargsversionnumber\
 Macro to parse an argument list, using user-specified parsing character]
% CREATED BY Steven B. Segletes <steven.b.segletes.civ@mail.mil>
% THIS PACKAGE IS RELEASED IN ACCORDANCE WITH THE LaTeX PUBLIC PROJECT LICENSE
% LPPL v1.3c (http://ctan.org/license/lppl1.3) OR ITS SUCCESSORS

% V1.00-Initial release
\newcounter{getarg@ctr}
\let\getargs\relax
\newcommand\getargs{}

\newcommand\setparsechar[1]{%
  \def\getargparsechar{#1}%
  \renewcommand{\getargs}[1]{%
    \setcounter{getarg@ctr}{0}%
    \parse@args##1#1\relax\relax%
  }%
  \def\parse@args##1#1##2\relax{%
    \stepcounter{getarg@ctr}%
    \expandafter\gdef\csname\getarg@root\romannumeral\value{getarg@ctr}\endcsname{##1}%
    \ifx\relax##2\relax%
      \expandafter\xdef\csname n\getarg@root\endcsname{\thegetarg@ctr}\else%
        \parse@args##2\relax\fi%
  }%
}

\newcommand\setparserootname[1][arg]{\def\getarg@root{#1}}

\let\showargs\relax
\newcommand\showargs[1][t]{%
  \fboxrule=.7pt\relax\fboxsep=\dimexpr-.5pt-\fboxrule\relax%
  \csname n\getarg@root\endcsname{} \textbackslash\getarg@root\ldots{} items %
    \if x#1(expanded)\else (tokens)\fi%
  : \showargs@help{#1}{1}{\csname n\getarg@root\endcsname}\unskip\fbox{\strut}.%
}

\newcommand\showargs@help[3]{%
  \setcounter{getarg@ctr}{#2}%
  \if x#1%
    \fbox{\strut}\csname\getarg@root\romannumeral\value{getarg@ctr}\endcsname%
    \hskip0pt\relax%
  \else
    \fbox{\strut}%
      \expandafter\detokenize\expandafter\expandafter\expandafter{%
      \csname\getarg@root\romannumeral\value{getarg@ctr}\endcsname}\hskip0pt\relax%
  \fi
  \ifnum\value{getarg@ctr}<#3\relax\stepcounter{getarg@ctr}%
    \showargs@help{#1}{\thegetarg@ctr}{#3}%
  \fi%
}

\setparserootname
\setparsechar{,}
\endinput
```