The coolstr package*

nsetzer

September 17, 2006

The coolstr package is a "sub" package of the cool package that seemed appropriate to publish independently since it may occur that one wishes to include the ability to check strings without having to accept all the overhead of the cool package itself.

1 Basics

Strings are defined as a sequence of characters (not TEX tokens). The main purpose behind treating strings as characters rather than tokens is that one can then do some text manipulation on them.

2 Implementation

This is just an internal counter for dealing with the strings; most often used for the length

```
\newcounter{COOL@strlen}%
```

`\setstrEnd` `\setstrEnd{`⟨*string*⟩`}` allows the user to set the end of a string 'character' in the rare event that the default value actually appears in the string. The default value is

```
\newcommand{\COOL@strEnd}{\%\%\%}
\newcommand{\COOL@intEnd}{\%@\%@\%@}
```

and may be changed by the following command (which utilizes the `\renewcommand`):

```
\newcommand{\setstrEnd}[1]{\renewcommand{\COOL@strEnd}{#1}}
```

This area defines the core technology behind the coolstr package: the string "gobbler".

```
\newcounter{COOL@strpointer}
```

*This document corresponds to cool v1.0, dated 2006/09/17.

Now we come to "the gobbler"—a recursive function that eats up a string. It must be written in TEX primatives.

The idea behind this is that "the gobbler" eats up everything before the desired character and everything after the desired character.

```
\def\COOL@strgobble[#1]#2#3{%
\ifthenelse{\equal{#3}{\COOL@strEnd}}
{%
\ifthenelse{\value{COOL@strpointer}=#1}
{%
#2
}%
% Else
{%
}%
}%
% Else
{%
\ifthenelse{\value{COOL@strpointer}=#1}
{%
#2
}%
% Else
{%
}%
\stepcounter{COOL@strpointer}%
\COOL@strgobble[#1]#3
}%
}
```

`\strchar` `\strchar{`⟨*index*⟩`}` gives the ⟨*index*⟩ character of the string. Strings start indexing at 1.

```
\newcommand{\strchar}[2]{%
\setcounter{COOL@strpointer}{1}%
\COOL@strgobble[#2]#1\COOL@strEnd%
}
```

This "gobbler" is used to determine if the string contains numeric data. As it stands, not all numeric data is recognized

```
\newboolean{COOL@firstdecimalfound}
\newboolean{COOL@seconddecimalfound}
\newboolean{COOL@efound}
\newboolean{COOL@digitfound}

\def\COOL@numericgobbler#1#2{%
\ifthenelse{\equal{#2}{\COOL@strEnd}}%
{%
\ifthenelse{`#1 < `0 \OR `#1 > `9}%
{%
```

```
Not Numeric%
}%
% Else
{%
Is Numeric%
}%
}%
% Else
{%
\ifthenelse{ `#1 < `0 \OR `#1 > `9 }%
{%
```

at this point it can only be a decimal point or an 'e' if it is to remain numeric

```
\ifthenelse{ `#1 = `. }%
{%
\ifthenelse{ \boolean{COOL@seconddecimalfound} }%
{%
```

this is the THIRD decimal

```
Not numeric%
}%
% else
{%
\ifthenelse{ \boolean{COOL@firstdecimalfound} }%
{%
```

this is the SECOND decimal

```
\ifthenelse{ \boolean{COOL@efound} }%
{%
\setboolean{COOL@seconddecimalfound}{true}%
\COOL@numericgobbler#2%
}%
% else
{%
Not numeric%
}%
}%
% else
{%
```

this is the FIRST decimal

```
\setboolean{COOL@firstdecimalfound}{true}%
\COOL@numericgobbler#2%
}%
}%
}%
% else
{%
\ifthenelse{ \(`#1 = `E\) \OR \(`#1 = `e\) }%
{%
\ifthenelse{ \boolean{COOL@efound} \OR \NOT \boolean{COOL@digitfound} }%
```

```
{%
Not numeric%
}%
% else
{%
\setboolean{COOL@efound}{true}%
\COOL@numericgobbler#2%
}%
}%
% else
{%
Not Numeric
}%
}%
}%
% Else
{%
\setboolean{COOL@digitfound}{true}%
\COOL@numericgobbler#2%
}%
}%
}
```

`\isnumeric` `\isnumeric{`⟨*string*⟩`}` returns some text stating whether or not ⟨*string*⟩ contains numeric data.

```
\newcommand{\isnumeric}[1]{%
\setboolean{COOL@firstdecimalfound}{false}%
\setboolean{COOL@seconddecimalfound}{false}%
\setboolean{COOL@efound}{false}%
\setboolean{COOL@digitfound}{false}%
\COOL@numericgobbler#1\COOL@strEnd%
}
```

In addition to identifying numeric data, it is useful to know if integers are present, thus another "gobbler" is needed

```
\let\COOL@strStop=\relax
\newboolean{COOL@isint}
\def\COOL@intgobbler#1#2\COOL@strEnd{%
\ifcat#11
\ifthenelse{\equal{#2}{\COOL@strStop}}%
{%
\ifthenelse{`#1 < `0 \OR `#1 > `9}%
{%
\setboolean{COOL@isint}{false}%
}%
% Else
{%
\setboolean{COOL@isint}{true}%
}%
```

```
}%
% Else
{%
\ifthenelse{ `#1 < `0 \OR `#1 > `9 }%
{%
\setboolean{COOL@isint}{false}%
}%
% Else
{%
\setboolean{COOL@digitfound}{true}%
\COOL@intgobbler#2\COOL@strEnd%
}%
}%
\else
\setboolean{COOL@isint}{false}%
\fi
}
```

`\isint` `\isint{`⟨*string*⟩`}` doesn't actually return anything, but sets the boolean `COOL@isint` to true or false depending on whether or not ⟨*string*⟩ contains an integer

```
\newcommand{\isint}[1]{\COOL@intgobbler#1\COOL@strStop\COOL@strEnd}
```

Change History

v1.0

General: Initial Release 1

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.

Symbols

\% 2, 3

C

\COOL@intEnd 3
\COOL@intgobbler 118, 140, 147
\COOL@numericgobbler 39, 68, 78, 93, 105, 114
\COOL@strEnd 2, 4, 7, 32, 40, 114, 118, 140, 147
\COOL@strgobble 6, 27, 32
\COOL@strStop 116, 120, 147

I

\isint 147
\isnumeric 109

S

\setstrEnd 2
\strchar 30