The ldcsetup package*

DPC, FMi

2007/09/12

Abstract

Basic code shared by several other packages currently under development.

```
1 ⟨*package⟩
2 \ProvidesExplPackage
3   {\filename}{\filedate}{\fileversion}{\filedescription}
4 \RequirePackage{keyval,l3calc}
5 \ExplSyntaxOff
```

1 Ignoring White Space

Within a package, or in a document preamble, you can use `\IgnoreWhiteSpace` this makes white space ignored, and `@` a letter, and `~` a space character. It lasts until the end of the current package or preamble.

Puts an ignored space at the end of the line, so '`\ `' does the right thing. (Perhaps should instead just change catcode of `^^M`)

```
6 \def\IgnoreWhiteSpace{%
7   \edef\@tempa{%
8   \catcode`\noexpand\~=\the\catcode`\~\relax
9   \catcode`\noexpand\ =\the\catcode`\ \relax
10  \catcode`\noexpand\^^I=\the\catcode`\^^I\relax
11  \catcode`\noexpand\@=\the\catcode`\@\relax
12  \endlinechar=\the\endlinechar\relax
13  }%
14  \ifx\@currname\@empty
15    \expandafter\AtBeginDocument\expandafter{\@tempa}%
16  \else
17    \expandafter\AtEndOfPackage\expandafter{\@tempa}%
18  \fi
19  \catcode`\~=10\relax
20  \catcode`\ =9\relax
21  \catcode`\^^I=9\relax
22  \makeatletter
```

*This file has version number 636, last revised 2007/09/12.

```
  \endlinechar=‘ %
  \relax
}
```

The new syntax requires : and _ to be letters; instead of putting that into `\IgnoreWhiteSpace` we make a new command which helps to gradually change packages from old to new syntax.

```
\def\InternalSyntaxOn{%
  \edef\@tempa{%
  \catcode‘\noexpand\~=\the\catcode‘\~\relax
  \catcode‘\noexpand\ =\the\catcode‘\ \relax
  \catcode‘\noexpand\^^I=\the\catcode‘\^^I\relax
  \catcode‘\noexpand\@=\the\catcode‘\@\relax
  \catcode‘\noexpand\:=\the\catcode‘\:\relax
  \catcode‘\noexpand\_=\the\catcode‘\_\relax
  \endlinechar=\the\endlinechar\relax
%FMi fix is not properly reset
  \endlinechar=13\relax
  }%
%%% ????
% this needs further work, don’t think it is working
  \ifx\InternalSyntaxOff\relax
    \expandafter\def\expandafter\InternalSyntaxOff\expandafter
{\@tempa\let\InternalSyntaxOff\relax}%
  \fi
  \ifx\@currname\@empty
    \expandafter\AtBeginDocument\expandafter{\@tempa}%
  \else
    \expandafter\AtEndOfPackage\expandafter{\@tempa}%
  \fi
  \catcode‘\~=10\relax
  \catcode‘\ =9\relax
  \catcode‘\^^I=9\relax
  \makeatletter
  \catcode‘\_=11\relax
  \catcode‘\:=11\relax
  \endlinechar=‘ %
  \relax
}
\let\InternalSyntaxOff\relax
```

Do it for this package.

```
%\InternalSyntaxOn
\ExplSyntaxOn
```

2 KV fixes

Allow[1] key names to start with \.

[1] Not needed now?

```
\def\define@key#1#2{%
  \@ifnextchar[
     {\KV@def{#1}{\string#2}}
     {\@namedef{KV@#1@\string#2}####1}}
\def\setkeys#1#2{%
  \def\KV@prefix{KV@#1@\expandafter\string}%
  \KV@do#2,\relax,}
```

Make KV # safe.

```
\newtoks\KV@toks
\def\KV@@sp@c#1\@nil#2\relax#3{\KV@toks{#1}\edef#3{\the\KV@toks}}
```

Generate error messages on missing ‘,’.

More exactly if two ‘=’ appear after a key generate an error. The current KV just silently ignores everything after the second ‘=’.

```
\def\KV@equal{=}
\def\KV@split#1=#2=#3\relax{%
  \KV@@sp@def\@tempa{#1}%
  \def\@tempd{#3}%
  \ifx\@tempa\@empty\else
    \expandafter\let\expandafter\@tempc
      \csname\KV@prefix\@tempa\endcsname
    \ifx\@tempc\relax
      \KV@error{\@tempa\space \expandafter\@gobbletwo\string\@undefined}\@eha
    \else
      \ifx\@tempd\@empty
        \KV@default
      \else
        \KV@@sp@def\@tempb{#2}%
        \ifx\@tempd\KV@equal
          \expandafter\@tempc\expandafter{\@tempb}\relax
        \else
          \KV@error{Extra~=~ sign~ after~ ‘#1’}\KV@erry
        \fi
      \fi
    \fi
  \fi}
\def\KV@erry{\expandafter\KV@errx\meaning\@tempd\relax ignored\MessageBreak
             missing~comma~in~‘\expandafter\strip@prefix\meaning\@tempb’~?}
\def\KV@errx#1>#2==\relax{%
 ‘#2’ }
\def\KV@default{%
  \expandafter\let\expandafter\@tempb
    \csname\KV@prefix\@tempa @default\endcsname
  \ifx\@tempb\relax
    \KV@error{No value specified for \@tempa}\@eha
  \else
    \@tempb\relax
```

```
  \fi}
```

LaTeX style error message.

```
\def\KV@error#1#2{\PackageError{keyval}{#1}{#2}}
```

Instead of doing a full KV parse, and evaluating all the keys, you might want to parse a parameter list, even for undefined keys, removing spaces, splitting up the ',' and '=' and resolving cases where no value is supplied. This is used in `\DeclareGenericFunction` which *defines* a set of keys via such a KV parse.

A setting of `, key = value ,` will result in `\KV@elt{key}{value}` being added to the list, a seting of `, key ,` will result in `\KV@default@elt{key}` being added. At the end of the parse the list is executed. No keys are checked at this stage (so no csnames are used up) the two `\KV...@elt` commands must be defined as appropriate.

```
\def\KV@parse#1{
  \begingroup
```

Locally fudge `\KV@@def` to just add to a token register rather than make a definition.

```
\def\KV@@sp@c##1\@nil##2\relax##3{\addto@hook##3{{##1}}}
```

Locally fudge `\KV@split` to just add to a token register rather than execute the code for the found keys.

```
\def\KV@split##1=##2=##3\relax{%
  \def\@tempd{##3}%
  \expandafter\ifx\expandafter=\@firstofone##1=\else
      \ifx\@tempd\@empty
        \addto@hook\KV@toks\KV@default@elt
        \KV@@sp@def\KV@toks{##1}%
      \else
        \ifx\@tempd\KV@equal
          \addto@hook\KV@toks\KV@elt
          \KV@@sp@def\KV@toks{##1}%
          \KV@@sp@def\KV@toks{##2}%
        \else
          \KV@err{Extra '=' after '##1'}\@ehd
        \fi
    \fi
  \fi}
```

Initialise

```
  \KV@toks{}
```

Do the parse

```
   \KV@do#1,\relax,
```

Evaluate the token register outside the group.

```
  \expandafter
  \endgroup
  \the\KV@toks}
```

3 Calc fixes

Count Register assignments via calc. Here we also provide some global versions since a normal global prefix won't have any effect.

```
\let:NN \SetInternalCounter \calc_int_set:Nn
\let:NN \GSetInternalCounter\calc_int_gset:Nn
```

4 Misc code that is best collected in one place

This section will receive odd code that would be of potential use to other packages and should eventually made it to the kernel (perhaps).

`\UndeclareRobustCommand` Remove a robust command from memory if it isn't used any longer. Of course this doesn't free the hash table but at least the space gets reclaimed.

```
\def\UndeclareRobustCommand#1{%
  \cs_gundefine:N #1
  \exp_args:Nc \cs_gundefine:N {\cs_to_str:N #1~}
}
```

⟨/package⟩