The locality package*

Jason Gross
JasonGross9+locality@gmail.com

June 29, 2010

1 Introduction

The locality package provides various macros to keep changes local to the current group. This allows one to (re)define helper macros without worrying about accidentally changing the functionality of another package's or the user's definitions. Additionally, it allows recursive macros to have some definitions persist between calls, and others be local.

2 Usage

I give the usage and specification of every macro defined. I give bugs when I know them (please email me if you find other bugs, or have fixes for the bugs I list). I sometimes give extra description or justification.

`\manyaftergroup` Usage: `\manyaftergroup{`⟨*tokens*⟩`}`
Specification: The ⟨*tokens*⟩ get placed after the current group
Bugs: No braces are permitted, spaces are stripped

I've often wanted to use TEX's `\aftergroup` with a variable-length argument. This macro allows this. It is expandable (may be used in `\edef`), but it gobbles spaces.

ToDo: Write a version of this macro that preserves spaces, allows braces.

`\locallydefine` Usage: `\locallydefine{`⟨*macro*⟩`}{`⟨*processing*⟩`}`
Specification: Execute ⟨*processing*⟩ inside of a group, and make the definition of `\macro` persist after the group ends.
Bugs: Changes via `\let` to an unexpandable macro yield an infinite recursive loop.

Normally, when you define a macro in a group, it's definition reverts after the group ends, unless you use `\global`. If you use `\global`, then the new definition replaces the current definition on all levels. This macro provides something in between.

`\DeclareNonlocalMacro` Usage: `\DeclareNonlocalMacro{`⟨*macro*⟩`}`

*This document corresponds to locality v0.1, dated 2010/06/29.

Specification: Any changes to the definition of `\macro` persist after the end of the current group.
Bugs: Changes via `\let` to an unexpandable macro yield an infinite recursive loop. Only works with `\begingroup \endgroup` (not with braces).

This macro generalized `\locallydefine`.

`\DeclareNonlocalCount` Usage: `\DeclareNonlocalCount{`⟨*count*⟩`}`
`\DeclareNonlocalLength` `\DeclareNonlocalLength{`⟨*length*⟩`}`
`\DeclareNonlocalToks` `\DeclareNonlocalToks{`⟨*toks*⟩`}`
Specification: Any changes to the value in ⟨*register*⟩ persist after the end of the current group.
Bugs: Only works with `\begingroup \endgroup` (not with braces). Only works if the argument is a single token.

These do the same thing to registers (⟨*count*⟩s, ⟨*length*⟩s, and ⟨*tok*⟩s) that `\DeclareNonlocalMacro` does to macros.

`\savevalues` Usage: `\savevalues{`⟨*list of macros (no separator)*⟩`}`
`\restorevalues` `\restorevalues{`⟨*list of macros (no separator)*⟩`}`
Specification: Every token in the passed argument is backed up by `\savevalues`, and the most recently backed up values are restored by `\restorevalues`.

`\makecommandslocal` Usage: `\makecommandslocal{`⟨*list of macros (no separator)*⟩`}{`⟨*code*⟩`}`
Specification: Every token in the first argument is made local to ⟨*code*⟩; changes made to their definitions do not perist outside of ⟨*code*⟩.

This macro is the natural opposite of `\DeclareNonlocalMacro`; it allows some macros to behave as if ⟨*code*⟩ was inside a group, while the rest of the macros behave as if they were not.

`\ignoreglobal` Usage: `\ignoreglobal`
`\obeyglobal` Usage: `\unignoreglobal`
`\unignoreglobal` Usage: `\obeyglobal`
Specification: The macro `\ignoreglobal` causes global changes, such as `\edef`, `\xdef`, and those prefaced by `\global`, to be local. The macro `\obeyglobal` causes these to be treated as global. The macro `\unignoreglobal` undoes the changes made by the last `\ignoreglobal`. If you call `\ignoreglobal` twice, then you must call `\unignoreglobal` twice to allow global changes.

`\makecounterslocal` Usage: `\makecounterslocal`
Specification: The macro `\makecounterslocal` redefines the LaTeX ⟨*counter*⟩ macros so that their changes are local, instead of global. At the end of the group in which `\makecounterslocal` is called, ⟨*counter*⟩ macros revert to being global.

3 Implementation

3.1 Helper functions

The following definitions are preliminary, to allow various tricks with `\def`.

```
\def\@nil{\@nil\relax} % this way, I'll know if I've messed up; I'll get
% a stack overflow error.
\def\if@nil#1{\@if@nil#1\@@nil}
```

```
\def\@if@nil#1#2\@@nil{\ifx\@nil#1}
```

obal@non@collision@unique@count

At various places, I want to have a macro associated with a certain name which hasn't been used before. I use this count to number them. I use a count, instead of a counter, so that I can control whether or not it's global. The long name, with lots of `@`s, is to (hopefully) avoid collisions.

```
\newcount\locality@global@non@collision@unique@count
\locality@global@non@collision@unique@count=0
```

\manyaftergroup

The macro `\manyaftergroup` works by parsing it's argument one token at a time, and applying `\aftergroup` to each argument. It checks for the end with `\@nil`.

```
\long\def\@manyaftergroup#1{\ifx#1\@nil \else \aftergroup#1
\expandafter\@manyaftergroup\fi}
\newcommand{\manyaftergroup}[1]{\@manyaftergroup#1\@nil}
```

\locallydefine

Execute the second argument passed locally, and then preserve the definition of the first argument passed.

```
\newcommand{\locallydefine}[2]{{#2\expandafter}%
  \expandafter\def\expandafter#1\expandafter{#1}}
```

The `\DeclareNonlocal` macros do some fancy stuff with `\begingroup` and `\endgroup`, so the old definitions must be saved.

```
\let\@old@begingroup=\begingroup
\let\@old@endgroup=\endgroup
```

These macros are extended versions of the `\locallydefine` macro; they redefine `\endgroup` to preserve definitions after the current group ends.

\DeclareNonlocalMacro

This macro redefines `\endgroup` to do this for macro passed to it.

First, back up `\endgroup` to a new macro.

```
\newcommand{\DeclareNonlocalMacro}[1]{\expandafter\let
  \csname endgroup \the\locality@local@group@non@local@macro@count
    \endcsname=\endgroup
```

Redefine `\endgroup` to, in order: revert it's definition, insert code to update the definition of the passed macro outside of the group, and call the (reverted) version of `\endgroup`.

```
  \expandafter\def\expandafter\endgroup\expandafter{%
    \expandafter\expandafter\expandafter\let\expandafter
      \expandafter\expandafter\endgroup\expandafter\expandafter
      \csname endgroup \the\locality@local@group@non@local@macro@count
        \endcsname\expandafter\endgroup
    \expandafter\def\expandafter#1\expandafter{#1}}%
  \advance\locality@local@group@non@local@macro@count by 1
}%
```

\DeclareNonlocalCount
\DeclareNonlocalLength

This works the same way as as `\DeclareNonlocalMacro`, but uses `\the` instead of `\def`.

```
\newcommand{\DeclareNonlocalCount}[1]{\expandafter\let
```

```
  \csname endgroup \the\locality@local@group@non@local@macro@count
    \endcsname=\endgroup
  \expandafter\def\expandafter\endgroup\expandafter{%
    \expandafter\expandafter\expandafter\let
      \expandafter\expandafter\expandafter\endgroup
      \expandafter\expandafter
      \csname endgroup \the\locality@local@group@non@local@macro@count
        \endcsname\expandafter\endgroup
    \expandafter#1\expandafter=\the#1 }% Note the space.  This is to
% prevent something like
% |\newcount\tempc\begingroup \DeclareNonlocalCount\tempc \tempc=1\endgroup1|
% from setting |\tempc| to 11.
  \advance\locality@local@group@non@local@macro@count by 1
}%
\let\DeclareNonlocalLength=\DeclareNonlocalCount
```

`\DeclareNonlocalToks` This works the same way as as `\DeclareNonlocalCount`, but puts braces around the assigned value; `\toks0=1` fails, and should be `\toks0={1}`.

```
\newcommand{\DeclareNonlocalToks}[1]{\expandafter\let
  \csname endgroup \the\locality@local@group@non@local@macro@count
    \endcsname=\endgroup
  \expandafter\def\expandafter\endgroup\expandafter{%
    \expandafter\expandafter\expandafter\let\expandafter\expandafter
      \expandafter\endgroup\expandafter\expandafter
      \csname endgroup \the\locality@local@group@non@local@macro@count'
      \endcsname\expandafter\endgroup
    \expandafter#1\expandafter=\expandafter{\the#1}}%
  \advance\locality@local@group@non@local@macro@count by 1
}%
```

I redefine `\begingroup` to reset the locality macros, so nesting works.

```
\newcount\locality@local@group@non@local@macro@count % Hopefully, this
% won't collide with anything.  I hope putting this out here allows
% proper nesting of groups
\def\begingroup{\@old@begingroup
  \let\endgroup=\@old@endgroup
  \locality@local@group@non@local@macro@count=0
}
```

`\savevalues` `\restorevalues` These macros parse their arguments token by token, renaming each `macro` to `@\macro backup`, or vice versa.

```
\def\@savevalues#1{\ifx#1\@nil \else \expandafter\let
  \csname @\string#1\space backup\endcsname=#1
  \expandafter\@savevalues\fi}
\newcommand{\savevalues}[1]{\@savevalues#1\@nil}

\def\@restorevalues#1{%
\ifx
  #1\@nil
```

```
\else
  \expandafter\let\expandafter#1\expandafter
    =\csname @\string#1\space backup\endcsname
    \expandafter
  \let\csname @\string#1\space backup\endcsname
    =\relax
  \expandafter\@restorevalues
\fi
}
\newcommand{\restorevalues}[1]{\@restorevalues#1\@nil}
```

`\makecommandslocal` Save the macros, run the code, then restore the values.

```
\newcommand{\makecommandslocal}[2]{\savevalues{#1}#2\restorevalues{#1}}
```

`\makecounterslocal` To make counters local without redefining them too badly (for example, this should work with the `calc` package, as long as you load `calc` first), we disable `\global`, and set `\gdef` and `\xdef` to `\def` and `\edef` respectively.

`\ignoreglobal` `\obeyglobal` `\unignoreglobal` We save the values of `\global`, `\gdef`, `\xdef`, globally, so that multiple calls don't fail.

We also save the value of `\@cons`, a special macro used in counters, which uses `\xdef` to append something to a list. Since it must be redefined for counters, I'll redefine it here to do without `\xdef` fails.

For reference, the original definition of `\@cons`, from `latex.ltx`, is `\def\@cons#1#2{\begingroup\let\@elt\relax\xdef#1{#1\@elt #2}\endgroup}`. I try to make this forward-compatible, but if the definition of `\@cons` changes too badly, this'll break.

```
\savevalues{\global\gdef\xdef\@cons}
{\def\begingroup{\begingroup\DeclareNonlocalMacro{##1}}%
\expandafter\expandafter\expandafter}%
\expandafter\expandafter\expandafter\def
\expandafter\expandafter\expandafter\locality@cons
\expandafter\expandafter\expandafter#%
\expandafter\expandafter\expandafter1%
\expandafter\expandafter\expandafter#%
\expandafter\expandafter\expandafter2%
\expandafter\expandafter\expandafter{\@cons{#1}{#2}}%
\newcommand{\obeyglobal}{\restorevalues{\globabl\gdef\xdef\@cons}}
\let\unignoreglobal=\obeyglobal
\newcommand{\ignoreglobal}{%
\let\global=\relax\let\gdef=\def\let\xdef=\edef
\let\@cons=\locality@cons
\expandafter\def\expandafter\unignoreglobal\expandafter{\expandafter
\def\expandafter\unignoreglobal\expandafter{\unignoreglobal}%
  \unignoreglobal}}
```

Now, the actual macro.

We redefine `\stepcounter`, `\addtocounter`, `\refstepcounter`, `\setcounter`, `\@addtoreset`, and `\@definecounter`.

Since `\newcounter` does everything with `\@addtoreset` and `\@definecounter`, it doesn't need and changes.

```
\newcommand{\makecounterslocal}{% FIX, to make more robust
  \expandafter\def\expandafter\stepcounter
    \expandafter##\expandafter1\expandafter{%
    \expandafter\ignoreglobal\stepcounter{##1}%
    \unignoreglobal
  }%
 %
  \expandafter\def\expandafter\refstepcounter
    \expandafter##\expandafter1\expandafter{%
    \expandafter\ignoreglobal\refstepcounter{##1}%
    \unignoreglobal
  }%
 %
  \expandafter\def\expandafter\setcounter
    \expandafter##\expandafter1%
    \expandafter##\expandafter2\expandafter{%
    \expandafter\ignoreglobal\setcounter{##1}{##2}%
    \unignoreglobal
  }%
 %
  \expandafter\def\expandafter\addtocounter
    \expandafter##\expandafter1%
    \expandafter##\expandafter2\expandafter{%
    \expandafter\ignoreglobal\addtocounter{##1}{##2}%
    \unignoreglobal
  }%
 %
  \expandafter\def\expandafter\@addtoreset
    \expandafter##\expandafter1%
    \expandafter##\expandafter2\expandafter{%
    \expandafter\ignoreglobal\@addtoreset{##1}{##2}%
    \unignoreglobal
  }%
 %
  \expandafter\def\expandafter\@definecounter
    \expandafter##\expandafter1\expandafter{%
    \expandafter\ignoreglobal\@definecounter{##1}%
    \unignoreglobal
  }%
```

Following the example of the `calc` package, if the amstext package is loaded we must add the `\iffirstchoice@` switch as well. We patch the commands this way since it's good practice when we know how many arguments they take.

```
  \@ifpackageloaded{amstext}{%
    \expandafter\def\expandafter\stepcounter
      \expandafter##\expandafter1\expandafter{%
      \expandafter\iffirstchoice@\stepcounter{##1}\fi
    }
```

```
    \expandafter\def\expandafter\addtocounter
      \expandafter##\expandafter1%
      \expandafter##\expandafter2\expandafter{%
      \expandafter\iffirstchoice@\addtocounter{##1}{##2}\fi
    }
  }{}
}
```