

The inlinedef package

Stephen Hicks*

v1.0 – 2008/07/10

1 Usage

1.1 The problem

Often package writers want to redefine certain macros to do slightly more than what they did previously, adding a control sequence or two to the beginning or the end of the definition. The easiest way to accomplish this is to use something like

```
\let\old@macro\macro
\def\macro{... \old@macro...}
```

But this sort of construction can cause problems if another package also wants to redefine the same macro and happens to choose the same name to save it to. It's also an ugly solution in that it pollutes the global namespace with extra macro names. A much cleaner solution is to define the new macro with the old macro expanded inline, as in `\edef\macro{... \macro...}`. This is generally problematic because there are often undefined control sequences and macros that we don't want to expand quite yet. A compromise is to use `\expandafter`, but this leads to error-prone and unreadable code:

```
\expandafter\def\expandafter\macro\expandafter{\expandafter
... \macro...}
```

1.2 The solution

What we really want is a way to expand just a few tokens in the definition and leave the rest untouched. We provide a command `\Inline` that can be inserted before a `\def` or `\gdef` (optionally prefixed by `\long`, `\outer`, and/or `\global`, as in `\Inline\long\outer\gdef...`). Within `\Inline` definitions, only tokens preceded by `\Expand` are expanded. Thus, the previous example becomes

```
\Inline\def\macro{... \Expand\macro...}
```

*email: sdh33@cornell.edu

1.3 Special commands

While nearly everything can be done with `\Expand` alone, we provide a few more keywords for completeness and convenience.

- | | |
|----------------------------|---|
| <code>\Expand</code> | <ul style="list-style-type: none">• <code>\Expand</code> - Performs a single expansion on the token or group immediately following and places the result directly into the definition without further processing. In the case of a group, only the first token is expanded (although <code>\expandafter</code> may be used to expand a different token), and the outermost grouping braces are discarded. |
| <code>\MultiExpand</code> | <ul style="list-style-type: none">• <code>\MultiExpand{<number>}</code> - Expands the following token or group the given number of times. For example,

<pre style="margin-left: 4em;"><code>\MultiExpand{3}{\expandafter\expandafter\expandafter\ax
 \expandafter\bx}</code></pre>
expands first <code>\c</code>, then <code>\b</code>, then <code>\a</code>, and inserts the whole expansion into the definition with no braces. Note that the braces are important. Otherwise it will just try to expand the first <code>\expandafter</code> three times, which is clearly wrong. |
| <code>\UnsafeExpand</code> | <ul style="list-style-type: none">• <code>\UnsafeExpand</code> - This version simply inserts an <code>\expandafter</code>, performing the expansion as in <code>\Expand</code> above, but reinserting the result back into the stream to be processed. Thus, any tokens like <code>\Expand</code> or <code>\Super</code> in the expansion will be acted on. Unlike the previous two commands, groups are <i>not</i> treated differently. |
| <code>\NoExpand</code> | <ul style="list-style-type: none">• <code>\NoExpand</code> - If a token is preceded by <code>\NoExpand</code> then it is inserted in the definition exactly as-is. This is required to insert any of the special tokens <code>\Expand</code>, <code>\NoExpand</code>, etc, as well as the internal token <code>\Q@END</code>, into a definition. If the token immediately following <code>\NoExpand</code> is an open-brace then the entire text of the group will be inserted without expansion, and the outer level of grouping will be lost. |
| <code>\Super</code> | <ul style="list-style-type: none">• <code>\Super</code> - When redefining an already existing macro, <code>\Super</code> will expand to the previous definition of the macro. Any macro parameters are automatically substituted. If the macro is undefined, or if the new parameter text doesn't match with the old text, then this will cause an error. |
| <code>\Recurse</code> | <ul style="list-style-type: none">• <code>\Recurse</code> - This is complementary to <code>\Super</code> and, while not strictly necessary, is included for clarity. <code>\Recurse</code> is equivalent to <code>\NoExpand\macro</code> when defining <code>\macro</code>. However, since <code>\macro</code> is, by default, not expanded anyway, this is a bit redundant. |

1.4 Calling options

When the name of the macro we're defining is encountered, there are three different ways we might proceed: leave it alone (`\NoExpand`), expand it with implicit parameters (`\Super`), or expand it with explicit parameters (`\UnsafeExpand`). We therefore allow zero, one, or two stars to come after `\Inline` to change this behavior.

- `\Inline` • `\Inline` - Without any stars, we default to leaving the macro name alone, as in `\NoExpand\macro`. This is the most consistent behavior with the rest of the package and works regardless of whether the macro is being defined or redefined.
- `\Inline*` • `\Inline*` - With a single star, we treat the macro name as a call to `\Super` and expand it with parameters inserted automatically. This is preferred over `\Expand` because it doesn't lead to the possible surprises in the case of recursively-defined macros.
- `\Inline**` • `\Inline**` - Finally, with two stars, the macro name is treated as if it were preceded by `\UnsafeExpand`. Any parameters must be inserted explicitly, and the expansion is itself subject to inline processing. Note that this form is the most dangerous.
- `\Inline!` One final option applies only in the case of redefining an already-existing macro. In this case, if the parameter text of the new definition differs from the parameter text of the old definition, we will produce an error. This error can be suppressed by adding a bang to the end of `\Inline` (either before or after the stars), acknowledging that any ill consequences that result are your own fault.

1.5 Known issues

- If a macro is defined with a character other than `#` catcoded to 6, then `\Super` will fail unless the same character is used in the redefinition.

1.6 Related packages

moredefs The `moredefs` package in the `frankenstein` collection provides some similar syntactic sugar, but is not as expressive.

2 Implementation

`\xa` Make the @-sign into a letter for use in macro names. As long as the packages are well-behaved, we can put this here and not later. We also define `\xa` to be `\expandafter` for convenience.

```
1 <*package>
2 \makeatletter
3 \let\xa\expandafter
```

`\ifID@aborted` We define a conditional so that we can gracefully abort in case of an error.

```
\ifID@star 4 \newif\ifID@aborted
\ifID@starstar 5 \newif\ifID@star
\ifID@bang 6 \newif\ifID@starstar
7 \newif\ifID@bang
```

`\ID@toks` At some point we need to stop using the internal toks registers and allocate our own, because somebody might want to `\Expand{\the\toks@}` and expect something else. We can get by with a single one though by defining a `\ID@pdef` that doubles all the # signs and then does a regular def, so that `\ID@pdef\cs\ID@toks ... \cs` will be the same as `\the\ID@toks`.

```
8 \newtoks\ID@toks
9 \newcount\ID@count
```

`\Inline` These are the macros that get it all started. `\Inline` opens up a group (which is closed at the end of `\ID@def`) and initializes a toks register (we don't bother allocating it since we're in a group and don't call any L^AT_EX or T_EX macros that make use of allocated toks registers. Then we scan the tokens until we find either an `\edef` or an `\xdef`. If it's anything else, we just add it to the toks register. We also have a list of bad tokens that will cause an error message, so that we don't go too far before figuring out what went wrong. Should `\Inline` be `\outer`?

```
10 \DeclareRobustCommand\Inline{%
11   \begingroup
12   % Define a few ‘quarks’
13   \def\Expand{\Expand}\def\Super{\Super}%
14   \def\UnsafeExpand{\UnsafeExpand}\def\MultiExpand{\MultiExpand}%
15   \def\Recurse{\Recurse}\def\NoExpand{\NoExpand}%
16   \def\Q@END{\Q@END}%
17   % Define a toks register
18   \ID@toks{}%
19   % Signal that we need to look for a star
20   \@testtrue\ID@starfalse\ID@starstarfalse\ID@bangfalse
21   % Start scanning for \def or \gdef
22   \futurelet\@foo\ID@scandef
23 }
24 \newcommand\ID@scandef{%
25   \let\next\ID@saveprefix % Default behavior
26   % If this is the first few tokens after the \Inline, check for * or !
27   \if@test
```

```

28 \ifx\@foo*%
29 \ifID@star
30 \ifID@bang\let\next\ID@sd@lastcheck\else\let\next\ID@sd@checkagain\fi
31 \ID@starstartrue
32 \else
33 \let\next\ID@sd@checkagain
34 \ID@startrue
35 \fi
36 \fi
37 \ifx\@foo!%
38 \ifID@bang\else % two bangs - can this be anything but an error?
39 \ID@bangtrue
40 \xa\let\xa\next\ifID@starstar\ID@sd@lastcheck\else\ID@sd@checkagain\fi
41 \fi
42 \fi
43 \fi
44 % Now look for a \def or \gdef
45 \ifx\@foo\def
46 \def\next{\ID@start\def}%
47 \fi
48 \ifx\@foo\gdef
49 \def\next{\ID@start\gdef}%
50 \fi
51 \ifcat\noexpand\@foo\space
52 \def\next{\ID@toks\xa\xa\xa{\xa\the\xa\ID@toks\space}%
53 \xa\futurelet\xa\@foo\xa\ID@scandef\ID@unspace}% copied from ID@space
54 \fi
55 % Error checking (minimal)
56 \@testfalse
57 \ifx\@foo\edef\@testtrue\fi\ifx\@foo\xdef\@testtrue\fi
58 \ifx\@foo\newcommand\@testtrue\fi\ifx\@foo\renewcommand\@testtrue\fi
59 \ifx\@foo\DeclareRobustCommand\@testtrue\fi
60 \if@test\PackageError{inlinedef}{Only \protect\def\space and \protect\gdef\space are
61 allowed after \protect\Inline,\MessageBreak but some other type of
62 definition was found}\@eha\let\next\ID@abort\fi
63 \@testfalse
64 \ifx\@foo\bgroup\@testtrue\fi\ifx\@foo\let\@testtrue\fi
65 \if@test\PackageError{inlinedef}{No \protect\def\space or \protect\gdef\space found
66 after \protect\Inline}\@ehd\def\next{\ID@abort{}}\fi
67 \next
68 }

```

`\ID@sd@checkagain` These just get the scandef loop started again and set `\@testtrue` if we're still looking for stars and/or bangs.

```

69 \def\ID@sd@checkagain#1{\@testtrue\futurelet\@foo\ID@scandef}
70 \def\ID@sd@lastcheck#1{\futurelet\@foo\ID@scandef}

```

`\ID@saveprefix` These are the three macros called by `\ID@scandef` to either save the prefix (`\long`, `\outer`, etc) to a token register, (attempt to) abort the procedure in case of an error, or else get the definition started once we find the `\edef` or `\xdef`.

```

71 \newcommand*\ID@saveprefix[1]{%
72   \ID@toks\x{\the\ID@toks#1}%
73   \futurelet\@foo\ID@scandef
74 }

```

In case the error was just the wrong type of `\def`, we consume up to and including the first explicit group.

```

75 \newcommand\ID@abort{}\def\ID@abort#1#\endgroup\@gobble}

```

To get the definition process started, we take `#1` as the definition command to save (either `\def` or `\gdef`), `#2` as the command that was provided (which we discard), `#3` is the name of the macro to define, and `#4` is the parameter text, delimited by a begin-group character.

```

76 \newcommand\ID@start{}\def\ID@start#1#2#3#4#{%
77   \xa\def\x{\ID@prefix\x{\the\ID@toks#1}%
78   \ID@def#3{#4}%
79 }

```

`\ID@fixparams` `\ID@fp@start` In order for `\Super` to work properly, we need to fix the parameter list to put the `#1` in braces, since it actually consists of two tokens. Therefore, `\ID@fixparams` takes everything between it and `\Q@END` and puts it in `\toks@fixedparams`. If it finds a `#`, then it checks whether the argument is delimited or not, and if not, it inserts a pair of braces. We currently define these with `\newcommand*`, though if there were a reason we could conceivably make them `\long`. Update: we now use `\ID@toks` and then define `\ID@fixedparams` from there.

```

80 \newcommand*\ID@fixparams{\begingroup\ID@toks{}\futurelet\@foo\ID@fp@start}
81 \newcommand*\ID@fp@start{%
82   \let\next\ID@fp@normal
83   \ifx\@foo\Q@END\let\next\ID@fp@end\fi
84   \ifcat\noexpand\@foo##\let\next\ID@fp@param\fi % was \ifx\@foo - broken?
85   \ifcat\noexpand\@foo\space
86     \def\next{\ID@toks\x\x\x\x{\xa\the\x\ID@toks\space}%
87     \xa\futurelet\x\@foo\x\ID@fp@start\ID@unspace}% copied from ID@space
88   \fi
89   \next
90 }

```

`\ID@fp@normal` `\ID@fp@param` `\ID@fp@end` These are the two commands that `\ID@fixparams` calls to actually consume each token, depending on whether it was a parameter.

```

91 \newcommand*\ID@fp@normal[1]{%
92   \ID@toks\x{\the\ID@toks#1}\futurelet\@foo\ID@fp@start
93 }
94 \newcommand*\ID@fp@param[2]{%
95   % We used to just use ###2 but need two more now...
96   % Need another doubling because we're now using it inside a def...
97   \def\@arg{#####2}% assume delimited unless we find # or \Q@END
98   \ifcat\noexpand\@foo##\def\@arg{#####2}\fi
99   \ifx\@foo\Q@END\def\@arg{#####2}\fi
100  \ID@toks\x\x\x\x{\xa\the\x\ID@toks\@arg}%

```

```

101 \futurelet\@foo\ID@fp@start
102 }
103 \newcommand*\ID@fp@end[1]{%
104 \xa\endgroup\xa\def\xa\ID@fixedparams\xa{\the\ID@toks}%
105 }

```

This should deal with everything *except* a single #, but that’s a hairy situation in the first place and we really don’t want to allow using `\Super` in that case. We could probably make an error message to say so. The only other alternative would be to “go back in time” and change the last `{#1}` to a `#1{}` and even then we end up with an extra `}` on the input stream. I can’t actually figure out how to test if this has happened in `...@insertp`, anyway.

`\ID@def` Here is where the “main loop” is initiated. We start by pretending to allocate another token register (though we actually just `\toksdef` it), and then define a number of quarks which we use as delimiters for various purposes. Finally, we start scanning. Afterwards, we test if there was an error and if not, we expand the definition command *after* the `\endgroup` so that we can clean up all the local variables.

```

106 \newcommand\ID@def[3]{%
107 % Other definitions
108 \global\ID@abortedfalse
109 \let\@reservedc#1%
110 \def\@macroname{#1}% for error message
111 \ID@fixparams#2\Q@END
112 % These are used by \Super but easier to define here
113 \def\@reservedb#2{}%
114 \edef\@reservedb{\xa\ID@getprefix\meaning\@reservedb\Q@END}%
115 \ifx#1\undefined % hopefully nobody’s going around defining \undefined
116 \let\@reserveda\undefined
117 \else
118 \edef\@reserveda{\xa\ID@getprefix\meaning#1\Q@END}%
119 \fi
120 % Scan it all into \ID@toks
121 \ifID@bang\else\ID@checkusage\fi
122 \ifID@aborted\else
123 \ID@toks{\ID@scan#3\Q@END}{% we need the {} so that the the #1# works...
124 \fi
125 \ifID@aborted
126 \def\command{}}% gracefully ignore
127 \else
128 \let#1\relax % don’t want it expanded in the |\edef| below
129 % We don’t need to worry about scope anymore
130 \toks0\ID@toks % likely redundant, but what if ID@toks=1 or 2?
131 \toks1\xa{\ID@prefix}% (easiest way to avoid expansion...)
132 \toks2{#2}%
133 \edef\command{\the\toks1#1\the\toks2{\the\toks0}}%
134 % We could also write this with 3 levels of \xa...
135 \fi

```

```

136 \global\ID@toks\xa{\ID@fixedparams}% just to test...
137 \expandafter\endgroup\command
138 }

```

`\ID@scan` This is the main loop. We look at each token in turn and deal with it, mostly
`\ID@switch` by inserting it into `\ID@toks`. If it's a `\Q@END` then we're done. If it's `\Super` or
`\Expand` then we need to do something special. If it's a space, then we need to
add the space to `\ID@toks`. Finally, if it's a `\bgroup` then we need to figure out
whether it's an explicit or an implicit group. In the former case, we descend into
it (writing `{...}` to `\ID@toks`) and in the latter, we just pick it up like normal.

```

139 \newcommand\ID@scan{\futurelet\@foo\ID@switch}
140 \newcommand\ID@switch{%
141   \let\next\ID@normal
142   \ifx\@foo\Q@END
143     \let\next\@gobble
144   \fi
145   \ifx\@foo\@reservedc % macro name... what to do?
146     \ifID@star
147       \ifID@starstar
148         \let\next\ID@expandmacro
149       \else
150         \let\next\ID@expandsuper
151       \fi
152     \fi
153   \fi
154   \ifx\@foo\Super
155     \let\next\ID@expandsuper
156   \fi
157   \ifx\@foo\Expand
158     \let\next\ID@expandnext
159   \fi
160   \ifx\@foo\UnsafeExpand
161     \let\next\ID@expandunsafe
162   \fi
163   \ifx\@foo\MultiExpand
164     \let\next\ID@expandmulti
165   \fi
166   \ifx\@foo\NoExpand
167     \let\next\ID@noexpandnext
168   \fi
169   \ifx\@foo\Recurse
170     \def\next{\xa\xa\xa\ID@scan\xa\xa\xa\NoExpand\xa\@macroname\@gobble}%
171   \fi
172   \ifcat\noexpand\@foo\space
173     \let\next\ID@space
174   \fi
175   \ifcat\noexpand\@foo\bgroup
176     \let\next\ID@trygroup
177   \fi

```



```
178 \next
179 }
```

`\ID@space` It's a bit tricky to deal with spaces properly. In particular, picking up just a space from the token list takes some doing. We need a fully-expandable macro so that the whole thing disappears. `\ID@space` then adds a space to `\ID@toks` and then expands `\ID@unspace` after `\ID@scan` so that the `\futurelet` sees the next token after the space and can deal with it properly. We need the `\expandafter` in defining `\ID@unspace` to actually get the space token into the parameter text; otherwise, it gets gobbled up by the lexer after reading the control sequence name.

```
180 \newcommand\ID@space{%
181   \ID@toks\xa\xa\xaf\xa\the\xa\ID@toks\space}%
182   \xa\ID@scan\ID@unspace
183 }
184 \newcommand\ID@unspace{}
185 \xa\def\xa\ID@unspace\space{}
```

`\ID@trygroup` The next two macros are used to check if the `\bgroup` token was an explicit or `\ID@recurse` an implicit grouping character. If it's explicit then the next macro that takes an argument will scoop the whole thing up at once, and so we need to be aware of this to deal with it. `\ID@trygroup` uses the special `#` delimiter and compares the argument with `\@empty` to see if anything comes before the next `{}`. If it doesn't find anything then it was an explicit group and we recurse. One consequence of this is that we always need to put a `{}` after `\Q@END` so that we don't get an error here.

```
186 \newcommand\ID@trygroup{}
187 \long\def\ID@trygroup#1#1{% check for explicit/implicit grouping!
188   \def\@reservedd{#1}%
189   \xa\let\xa\next
190   \ifx\@reservedd\@empty\ID@recurse\else\ID@normal\fi
191   \next#1%
192 }
```

Here we need to do some gymnastics to get the `{` and `}` tokens into the `toks` register. It would be easiest if we could just add them one at a time, but we can only add *balanced text*, so we need to expand the whole thing *first* and then add it back to the register we expanded it into. Thus, we enter a new level of grouping to save the contents of `\ID@toks`, expand the inner group, and use `\expandafter` across an `\endgroup` to get the correct tokens in the right place in `\ID@toks`.

```
193 \newcommand\ID@recurse[1]{%
194   \begingroup\ID@toks{}% start a new level of grouping and empty \ID@toks
195   \ID@scan#1\Q@END{}% % parse...
196   \xa\endgroup\xa   % this fiasco should get the job done...!
197   \ID@toks\xa\xa\xaf\xa\the\xa\ID@toks\xa{\the\ID@toks}%
198   \ID@scan
199 }
```

`\ID@normal` This is what we do when it's not anything special.
`ID@noexpandnext`

```

200 \newcommand\ID@normal[1]{\ID@toks\xa{\the\ID@toks#1}\ID@scan}
201 \newcommand\ID@noexpandnext[2]{\ID@toks\xa{\the\ID@toks#2}\ID@scan}

```

\ID@checkusage Here we define tests that will issue errors if the parameter texts aren't the same,
 \ID@checkredef or the original function isn't defined.

```

202 \newcommand*\ID@checkusage{%
203 % Make sure parameter lists are the same, does nothing if undefined
204 \ifx\@reserveda\@reservedb
205 \else
206 % Error messages
207 \ifx\@reserveda\undefined % undefined - okay
208 \else
209 \global\ID@abortedtrue
210 \ifx\@foo\Super
211 \PackageError{inlinedef}{Cannot use \protect\Super\space in \expandafter
212 \protect\@macroname\space because\MessageBreak
213 parameter lists don't match:\MessageBreak
214 '\@reservedb' (new) != '\@reserveda' (old)}\@eha
215 \else
216 \ifID@bang % auto-expansion forbidden
217 \PackageError{inlinedef}{Cannot use \protect\Inline* auto-expansion in
218 \expandafter\protect\@macroname\MessageBreak
219 because parameter lists don't match:\MessageBreak
220 '\@reservedb' (new) != '\@reserveda' (old)}\@eha
221 \else
222 \PackageError{inlinedef}{Parameter lists for
223 \expandafter\protect\@macroname\space don't match:\MessageBreak
224 '\@reservedb' (new) != '\@reserveda' (old)\MessageBreak
225 Use !-form of \protect\Inline\space to ignore this}\@eha
226 \fi
227 \fi
228 \fi
229 \fi
230 }
231 \newcommand*\ID@checkredef{%
232 \ifx\@reserveda\undefined % undefined - okay
233 \PackageError{inlinedef}{Cannot use \ifx\@foo\Super\protect\Super\space
234 \else\protect\Inline** \fi in \expandafter\protect\@macroname\space
235 because \MessageBreak it hasn't been defined yet}%
236 \@eha
237 \global\ID@abortedtrue
238 \fi
239 }

```

\ID@expandsuper These correspond to the two special tokens, \Super and \Expand. The first one
 \ID@expandnext tests that the parameter list is alright and that the original command wasn't
 \ID@expandmulti undefined. If all is well, it expands everything in the right order. The second one
 \ID@expandunsafe is simpler, just inserting an \expandafter before the continuation (\ID@scan) to
 \ID@expandmacro expand whatever comes next once. There is (yet) no way to fully-expand, although

several `\Expands` and `\expandafters` can be stacked cleverly to expand several things in a specific order.

```

240 \newcommand*\ID@expandsuper[1]{%
241   \ID@checkusage\ID@checkredef
242   \ifID@aborted\else
243     \ID@toks\x@\x@\x@\x@\x@\x@\x@
244     {\x@\x@\x@\the\x@\x@\x@\ID@toks\x@\@reservedc\ID@fixedparams}%
245   \fi
246   \ID@scan
247 }
248 \newcommand\ID@expandnext[2]{%
249   \ID@toks\x@\x@\x@\x@\the\x@\ID@toks#2}\ID@scan
250 }
251 \newcommand\ID@expandmulti[3]{%
252   \begingroup % #1 is the \MultiExpand...
253   \ID@count#2\relax % this will need to be allocated too!
254   \ID@toks{#3}%
255   \@testtrue\ifnum\ID@count<\@ne\@testfalse\fi
256   \@whilesw\if@test\fi{%
257     \ID@toks\x@\x@\x@\the\ID@toks}% one expansion...
258     \advance\ID@count\m@ne\ifnum\ID@count<\@ne\@testfalse\fi
259   }%
260   \x@endgroup\x@\ID@toks\x@\x@\x@\x@\the\x@\ID@toks\the\ID@toks}\ID@scan
261 }
262
263 \newcommand*\ID@expandunsafe[1]{\expandafter\ID@scan}
264 \newcommand*\ID@expandmacro[1]{\expandafter\ID@scan\@reservedc}

```

`\ID@getprefix` This is used to compare argument lists.

```

265 \newcommand\ID@getprefix{\long\def\ID@getprefix#1:#2->#3\Q@END{\detokenize{#2}}

```

Finally we clean up by restoring @’s catcode.

```

266 \makeatother
267 \endpackage

```

3 Test suite

We include a somewhat-comprehensive test suite to make sure that everything is working. If it works properly, it should output nothing.

First we define a few helper-functions to test for errors, etc.

```
268 <*testsuite>
269 \makeatletter
270 \errorcontextlines=10
271 \def\WantError#1#2#3{%
272   \let\WE@packageerror\PackageError
273   \def\PackageError##1##2##3{%
274     \protected@edef\@goterror{##2}\protected@edef\@wanterror{##2}%
275     \edef\@goterror{\xa\detokenize\xa{\@goterror}}%
276     \edef\@wanterror{\xa\detokenize\xa{\@wanterror}}%
277     \protected@edef\@gotpackage{##1}\protected@edef\@wantpackage{##1}%
278     \edef\@gotpackage{\xa\detokenize\xa{\@gotpackage}}%
279     \edef\@wantpackage{\xa\detokenize\xa{\@wantpackage}}%
280     \global\let\PackageError\WE@packageerror
281     \@tempswafalse
282     \ifx\@gotpackage\@wantpackage\else\message{^^J(arg 1 differs)^^J}\@tempswatrue\fi
283     \ifx\@goterror\@wanterror\else\message{^^J(arg 2 differs)^^J}\@tempswatrue\fi
284     \ifx##3##3\else\message{^^J(arg 3 differs)^^J}\@tempswatrue\fi
285     \if@tempswa\PackageError{inlinedef (test)}{wrong error}\@eha\PackageError{##1}{##2}##3\fi
286   }%
287 }
288
289 \def\CheckError{%
290   \ifx\PackageError\WE@packageerror\else
291     \PackageError{inlinedef (test)}{expected error not thrown}\@eha\fi
292   \global\let\PackageError\WE@packageerror
293 }
294 \newcommand\CheckDefinition[1][\@CheckDefinition{##1}]
295 \def\@CheckDefinition#1#2#3#4{\@checkdefn{##1}##2{##3}}
296 \def\@checkdefn#1#2#3#4#5{\def\@reserveda#3{##4}\ifx#2\@reserveda\else
297   \message{^^J^^Jmeaning#2^^J(got)vs(wanted)^^Jmeaning\@reserveda^^J^^J}
298   \PackageError{inlinedef (test)}{definition of \detokenize{##2}didn't match}\@eha\fi
299 }
300 \let\eha\@eha\let\ehd\@ehd
301 \makeatother
302 \catcode'\#=12
303 \def\pound{#}
304 \catcode'\#=6
305
306 \def\WantSuperNoMatch#1#2#3{%
307   \WantError{inlinedef}{Cannot use \protect\Super\space in
308     \protect#1\space because\MessageBreak
309     parameter lists don't match:\MessageBreak
310     '#3' (new) != '#2' (old)}\eha
```

```

311 }
312 \def\WantStarNoMatch#1#2#3{%
313   \WantError{inlinedef}{Cannot use \protect\Inline* auto-expansion in
314     \protect#1\MessageBreak because
315     parameter lists don't match:\MessageBreak
316     '#3' (new) != '#2' (old)}\eha
317 }
318 \def\WantNoMatchBang#1#2#3{%
319   \WantError{inlinedef}{Parameter lists for
320     \protect#1\space don't match:\MessageBreak
321     '#3' (new) != '#2' (old)\MessageBreak
322     Use !-form of \protect\Inline\space to ignore this}\eha
323 }
324 \def\WantOnlyDefGdef{%
325   \WantError{inlinedef}{Only \protect\def\space and \protect\gdef\space are
326     allowed after \protect\Inline,\MessageBreak but some other type of
327     definition was found}\eha
328 }
329 \def\WantNoDefGdef{%
330   \WantError{inlinedef}{No \protect\def\space or \protect\gdef\space found
331     after \protect\Inline}\ehd
332 }
333 \def\WantSuperNoRedef#1{%
334   \WantError{inlinedef}{Cannot use \protect\Super\space in \protect#1\space
335     because \MessageBreak it hasn't been defined yet}\eha
336 }

```

Now we start the actual tests.

```

337 % I. Basic stuff
338 %   A. Simple definition
339 \let\a\undefined
340 \Inline\def\a{b}
341 \CheckDefinition\a{b}
342
343 %   B. Simple redefinition
344 \def\a{b}
345 \Inline\def\a{d}
346 \CheckDefinition\a{d}
347
348 %   C. Erroneous redefinition (needs !)
349 \def\a{b}
350 \WantNoMatchBang\a}{\pound1}
351 \Inline\def\a#1{c}
352 \CheckError
353 \CheckDefinition\a{b} % shouldn't have changed
354
355 \def\a{b}
356 \Inline!\def\a#1{c}
357 \CheckDefinition\a#1{c}
358

```

```

359 %    D. Local/global definition
360 \def\{a}{b}
361 \begingroup
362 \Inline\def\{a}{c}
363 \endgroup
364 \CheckDefinition\{a}{b}
365
366 \begingroup
367 \Inline\gdef\{a}{c}
368 \endgroup
369 \CheckDefinition\{a}{c}
370
371 {\Inline\global\def\{a}{d}}
372 \CheckDefinition\{a}{d}
373
374 %    E. Collecting arguments
375 \Inline\long\def\{a}{e}
376 \CheckDefinition[\long]\{a}{e}
377
378 \Inline\outer\def\{a}{f}
379 \edef\{a}{\meaning\{a}}
380 \edef\{b}{\detokenize{\outer macro:->f}}
381 \xa\CheckDefinition\x{a}\xa\{b}
382
383 \Inline\long\outer\def\{a}{g}
384 \edef\{a}{\meaning\{a}}
385 \edef\{b}{\string\long\string\outer\space\detokenize{macro:->g}}
386 \xa\CheckDefinition\x{a}\xa\{b}
387
388 \def\{a}{g}
389 \Inline!\long\def\{a#1}{h}
390 \CheckDefinition[\long]\{a#1}{h}
391
392 % II. Special tokens
393 %    A. Recursion
394 \def\{a}{b}
395 \Inline\def\{a}{a c}
396 \CheckDefinition\{a}{a c}
397
398 %    B. Expansion
399 \def\{a}{b}
400 \Inline\def\{a}{a\Expand a c}
401 \CheckDefinition\{a}{abc}
402
403 \def\{a}{b}
404 \Inline\def\{a}{\Expand a\Expand a\Expand c}
405 \CheckDefinition\{a}{abc}
406
407 \def\{a}{b}
408 \def\{b}{c}

```

```

409 \Inline\def\{a\Expand\{a c}
410 \CheckDefinition\{a\{a b c}
411
412 \toks0{b}\toks1{d}
413 \Inline\def\{a\{the\toks0c\the\toks1e}
414 \CheckDefinition\{a\{the\toks0c\the\toks1e}
415
416 \Inline\def\{a\{Expand\{the\toks0}c\Expand\{the\toks1}e}
417 \CheckDefinition\{a\{abcde}
418
419 \Inline\def\{a\{Expand\{a\the\toks0}c\Expand\{the\toks1}e}
420 \CheckDefinition\{a\{a\the\toks0cde}
421
422 \Inline\def\{a\{Expand\{expandafter a\the\toks0}c\Expand\{the\toks1}e}
423 \CheckDefinition\{a\{abcde}
424
425 % C. MultiExpand
426 \def\x{y}
427 \def\y{z}
428 \def\z{0}
429 \Inline\def\{a\{a\MultiExpand0\x b}
430 \CheckDefinition\{a\{a\x b}
431
432 \Inline\def\{a\{a\MultiExpand1\x b}
433 \CheckDefinition\{a\{a\y b}
434
435 \Inline\def\{a\{a\MultiExpand2\x b}
436 \CheckDefinition\{a\{a\z b}
437
438 \Inline\def\{a\{a\MultiExpand3\x b}
439 \CheckDefinition\{a\{a0b}
440
441 \Inline\def\{a\{a\MultiExpand{10}\x b}
442 \CheckDefinition\{a\{a0b}
443
444 % i. use with \expandafter
445 \Inline\def\{a\{a\MultiExpand2{\expandafter\expandafter\x}b}
446 \CheckDefinition\{a\{a\y b}
447
448 \Inline\def\{a\{a\MultiExpand1{\expandafter\expandafter\x}b}
449 \CheckDefinition\{a\{a\expandafter\y\x b}
450
451 \Inline\def\{a\{a\MultiExpand2{\expandafter\expandafter\expandafter\x}b}
452 \CheckDefinition\{a\{a\x\z b}
453
454 \Inline\def\{a\{a\MultiExpand3{\expandafter\expandafter\expandafter\x}b}
455 \CheckDefinition\{a\{a\y\z b}
456
457 \Inline\def\{a\{a\MultiExpand4{\expandafter\expandafter\expandafter\x}b}
458 \CheckDefinition\{a\{a\z b}

```

```

459
460 \Inline\def\{a\{MultiExpand5{\expandafter\expandafter\expandafter\x\x}b}
461 \CheckDefinition\{a{a0\z b}
462
463 %    D. UnsafeExpand
464 \def\x{b\Super c}
465 \Inline\def\{a\{Expand\x d}
466 \CheckDefinition\{a{ab\Super cd}
467
468 \def\{a{0}
469 \Inline\def\{a\{UnsafeExpand\x d}
470 \CheckDefinition\{a{ab0cd}
471
472 \def\{a{b\Super d}
473 \Inline\def\{a\{UnsafeExpand\{a e}
474 \CheckDefinition\{a{abb\Super dde}
475
476 \def\{a{b\Super d}
477 \Inline**\def\{a\{a e}
478 \CheckDefinition\{a{abb\Super dde}
479
480 % Would be nice if we could catch TeX capacity exceeded errors...
481 % Then try \def\{a{b\{a d}\Inline**\def\{a\{a e}
482
483 \def\x#1{b#1d}
484 \Inline\def\{a\{a\x ce}
485 \CheckDefinition\{a\{a\x ce}
486
487 \Inline\def\{a\{UnsafeExpand\x ce}
488 \CheckDefinition\{a{abcde}
489
490 %    E. NoExpand
491 \Inline\def\{a\{NoExpand\Expand\x b}
492 \CheckDefinition\{a\{a\Expand\x b}
493
494 \Inline*\def\{a\{NoExpand\{a b}
495 \CheckDefinition\{a\{a b}
496
497 \Inline**\def\{a\{NoExpand\{a b}
498 \CheckDefinition\{a\{a b}
499
500 \Inline\def\{a\{NoExpand{\Expand\x\Expand\y}b}
501 \CheckDefinition\{a\{a\Expand\x\Expand\y b}
502
503 %    F. Super
504 \def\{a{bcd}
505 \Inline\def\{a\{Super e}
506 \CheckDefinition\{a{abcde}
507
508 \def\{a#1{b#1d}

```



```

509 \Inline\def\#1{a\Super e}
510 \CheckDefinition\#1{ab#1de}
511
512 \def\#1{b#1d}
513 \Inline*\def\#1{a\Super e}
514 \CheckDefinition\#1{ab#1de}
515
516 \def\#1{b#1d}
517 \Inline**\def\#1{a\Super e}
518 \CheckDefinition\#1{ab#1de}
519
520 % G. Recurse
521 \def\#1{q}
522 \Inline\def\#1{a\Recurse b}
523 \CheckDefinition\#1{a\ a b}
524
525 \Inline*\def\#1{a\Recurse b}
526 \CheckDefinition\#1{a\ a b}
527
528 \Inline**\def\#1{a\Recurse b}
529 \CheckDefinition\#1{a\ a b}
530
531 % III. Tricky parsing
532 % A. Spaces
533 \def\#1{b c d}
534 \Inline\def\#1{a \Super e}
535 \CheckDefinition\#1{a b c de}
536
537 \def\#1{b c d}
538 \Inline\def\#1{a \Expand\ a e}
539 \CheckDefinition\#1{a b c de}
540
541 \def\#1{b c d}
542 \Inline\def\#1{a \Expand{\ a} e}
543 \CheckDefinition\#1{a b c d e}
544
545 \Inline\def\#1{a\NoExpand{\Expand\ x\Expand\ y} b}
546 \xa\CheckDefinition\xa\ xa{\ xa a\ xa\Expand\ xa\ x\ xa\Expand\ xa\ y\space b}
547
548 % B. Grouping
549 \def\#1{b{c d}e}
550 \Inline\def\#1{a\Super}f\Super}
551 \CheckDefinition\#1{ab{c d}e}fb{c d}e}
552
553 \Inline\def\#1{{ }{}}
554 \Inline\def\#1{\Expand\ a\ a}{ } {{\Super}{}}
555 \CheckDefinition\#1{{ }{ }\ a}{ } {{ }{ }}{ }}
556
557 % C. Parameters
558 \def\#1bcd#2{[#1...#2]}

```

```

559 \Inline\def\abcd#2{a\Super b}
560 \CheckDefinition\abcd#2{a[#1...#2]b}
561
562 \def\abcd#2{y}
563 \Inline\def\abcd#2{x\UnsafeExpand\abcd#2}
564 \CheckDefinition\abcd#2{xyz}
565
566 \def\abcd#2{#1y#2}
567 \Inline\def\abcd#2{x\UnsafeExpand\abcd#2}
568 \CheckDefinition\abcd#2{x#1y#2z}
569
570 %      i. spaces!
571 \def\abcd #1 {y}
572 \Inline\def\abcd #1 {x\Super z}
573 \CheckDefinition\abcd #1 {xyz}
574
575 \xa\def\abcd\abcd\space{y}
576 \xa\Inline\abcd\abcd\space{x\Super z}
577 \xa\CheckDefinition\abcd\abcd\space{xyz}
578
579 %      ii. funky catcodes
580 %%% This test fails.
581 %\begingroup
582 % \catcode'\&=6
583 % \def\abcd{b#1d}
584 % \Inline\def\abcd{a\Super e}
585 % \CheckDefinition\abcd{b&1d}
586 %\endgroup
587
588 %      D. Active characters
589 \begingroup
590 \catcode'A=13
591 \defA#1{b#1d}
592 \Inline\defA#1{aAe}
593 \CheckDefinitionA#1{aAe}
594
595 \defA#1{b#1d}
596 \Inline*\defA#1{aAe}
597 \CheckDefinitionA#1{ab#1de}
598 \endgroup
599
600 % IV. Auto-expansion
601
602 \def\abcd{y}
603 \Inline\def\abcd{x\abcd}
604 \CheckDefinition\abcd{x\abcd}
605
606 \def\abcd{y}
607 \Inline*\def\abcd{x\abcd}
608 \CheckDefinition\abcd{xyz}

```

```

609
610 \def\a#1{y}
611 \Inline**\def\a#1{x\{a{#1}z}
612 \CheckDefinition\a#1{xyz}
613
614 % A. With delimited arguments
615 \def\a[#1]#2{#1y#2}
616 \Inline*\def\a[#1]#2{x a z}
617 \CheckDefinition\a[#1]#2{x#1y#2z}
618
619 \def\a[#1]#2{#1y#2}
620 \Inline**\def\a[#1]#2{x\{a{#1}{#2}z}
621 \CheckDefinition\a[#1]#2{x#1y#2z}
622
623 % V. Errors
624
625 \def\bar#1{d #1 f}
626 \def\x{b}
627
628 \WantSuperNoMatch\a{\pound1}{.\pound1}
629 \def\a#1{x}
630 \Inline!\def\a.#1{y\Super}
631 \CheckError
632 \CheckDefinition\a#1{x}
633
634 \WantStarNoMatch\a{\pound1}{.\pound1}
635 \def\a#1{x}
636 \Inline*!\def\a.#1{y\}a}
637 \CheckError
638 \CheckDefinition\a#1{x}
639
640 \def\a#1{x}
641 \Inline!\def\a.#1{y} % ok
642 \CheckDefinition\a.#1{y}
643
644 \WantOnlyDefGdef
645 \def\foo{a}
646 \Inline\edef\foo{b}
647 \CheckError
648 \CheckDefinition\foo{a}
649 \let\foo\undefined
650
651 \WantOnlyDefGdef
652 \Inline\global\outer\xdef{}
653 \CheckError
654
655 \WantOnlyDefGdef
656 \Inline\global\outer\abc\space vbcda s \newcommand{}
657 \CheckError
658

```

```

659 \WantNoDefGdef
660 \Inline\let\relax\relax
661 \CheckError
662
663 \WantNoDefGdef
664 \Inline{}
665 \CheckError
666
667 \WantSuperNoRedef\foo
668 \Inline\def\foo#1{a \Expand\x\space cd #1 fg \x\space i\Super}
669 \CheckError
670
671 \WantNoMatchBang\a{}{\pound1}
672 \def\a{b}
673 \Inline\def\a#1{a\ a c}
674 \CheckError
675
676 % Miscellaneous (read: "old") tests
677
678 \def\test#1{d #1 f}
679 \Inline\def\test#1{a \Expand\x\space c\Super g \x\space i}
680 \CheckDefinition\test#1{a b\space cd #1 fg \x\space i}
681
682 \Inline*\def\bar#1{a \Expand\x\space c\bar g \x\space i}
683 \CheckDefinition\bar#1{a b\space cd #1 fg \x\space i}
684
685 \def\bar#1{d #1 f}
686 \Inline**\def\bar#1{a \Expand\x\space c\bar{#1}g \x\space i}
687 \CheckDefinition\bar#1{a b\space cd #1 fg \x\space i}
688
689 \Inline\def\foo#1{a \Expand\x\space cd #1 fg \x\space i}
690 \CheckDefinition\foo#1{a b\space cd #1 fg \x\space i}
691
692 \def\a{b}
693 \Inline!\def\a#1{a\Expand\ a c}
694 \CheckDefinition\a#1{abc}
695
696 \def\a{b}
697 \Inline!**\def\a#1{a\ a c}
698 \CheckDefinition\a#1{abc}
699
700 \Inline\def\a#1{a\ a c}
701 \CheckDefinition\a#1{a\ a c}
702
703 \Inline\def\a#1{a\Recurse c}
704 \CheckDefinition\a#1{a\ a c}
705
706 \Inline!\def\a{a\NoExpand{b\Super c}d}
707 \CheckDefinition\a{ab\Super cd}
708

```

```
709 \Inline*\def\a{gh\a jk}
710 \CheckDefinition\a{ghab\Super cdjk}
711
712 % SURPRISE! unsafe expansion...
713 \def\a{ab\Super cd}
714 \Inline**\def\a{gh\a jk}
715 \CheckDefinition\a{ghabab\Super cdcjdk}
716
717 \def\a{ab\Super cd}
718 \Inline\def\a{gh\UnsafeExpand\a jk}
719 \CheckDefinition\a{ghabab\Super cdcjdk}
720
721 \def\x{x a} % This is a fun one...!
722 \Inline\def\a{\MultiExpand{5}\x}
723 \CheckDefinition\a{x aaaaa}
724
725 \message{^^JAll tests completed.^^J}
726
727 \begin{document}
728 \end{document}
729 </testsuite>
```

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	
<code>\@arg</code>	97–100
<code>\@foo</code>	22, 28, 37, 45, 48, 51, 53, 57–59, 64, 69, 70, 73, 80, 83–85, 87, 92, 98, 99, 101, 139, 142, 145, 154, 157, 160, 163, 166, 169, 172, 175, 210, 233
<code>\@macroname</code>	110, 170, 212, 218, 223, 234
<code>\@reserveda</code>	116, 118, 204, 207, 214, 220, 224, 232
<code>\@reservedb</code>	113, 114, 204, 214, 220, 224
<code>\@reservedc</code>	109, 145, 244, 264
<code>\@reservedd</code>	188, 190
<code>\@testfalse</code>	56, 63, 255, 258
<code>\@testtrue</code>	20, 57–59, 64, 69, 255
B	
<code>\bgroup</code>	64, 175
C	
<code>\command</code>	126, 133, 137
E	
<code>\Expand</code>	<i>1</i> , 2, 13, 157
I	
<code>\ID@abort</code>	62, 66, <u>71</u> , 75
<code>\ID@abortedfalse</code>	108
<code>\ID@abortedtrue</code>	209, 237
<code>\ID@bangfalse</code>	20
<code>\ID@bangtrue</code>	39
<code>\ID@checkredef</code>	<u>202</u> , 231, 241
<code>\ID@checkusage</code>	121, 202, <u>202</u> , 241
<code>\ID@count</code>	<u>8</u> , 9, 253, 255, 258
<code>\ID@def</code>	78, 106, <u>106</u>
<code>\ID@expandmacro</code>	148, <u>240</u> , 264
<code>\ID@expandmulti</code>	164, <u>240</u> , 251
<code>\ID@expandnext</code>	158, <u>240</u> , 248
<code>\ID@expandsuper</code>	150, 155, 240, <u>240</u>
<code>\ID@expandunsafe</code>	161, <u>240</u> , 263
<code>\ID@fixedparams</code>	104, 136, 244
<code>\ID@fixparams</code>	80, <u>80</u> , 111
<code>\ID@fp@end</code>	83, <u>91</u> , 103
<code>\ID@fp@normal</code>	82, 91, <u>91</u>
<code>\ID@fp@param</code>	84, <u>91</u> , 94
<code>\ID@fp@start</code>	80, <u>80</u> , 81, 87, 92, 101
<code>\ID@getprefix</code>	114, 118, 265, <u>265</u>
<code>\ID@noexpandnext</code>	167, <u>200</u> , 201
<code>\ID@normal</code>	141, 190, 200, <u>200</u>
<code>\ID@prefix</code>	77, 131
<code>\ID@recurse</code>	<u>186</u> , 190, 193
<code>\ID@saveprefix</code>	25, 71, <u>71</u>
<code>\ID@scan</code>	123, 139, <u>139</u> , 170, 182, 195, 198, 200, 201, 246, 249, 260, 263, 264
<code>\ID@scandef</code> ..	<u>10</u> , 22, 24, 53, 69, 70, 73
<code>\ID@sd@checkagain</code> ..	30, 33, 40, 69, <u>69</u>
<code>\ID@sd@lastcheck</code>	30, 40, <u>69</u> , 70
<code>\ID@space</code>	173, 180, <u>180</u>
<code>\ID@starfalse</code>	20
<code>\ID@starstarfalse</code>	20
<code>\ID@starstartrue</code>	31
<code>\ID@start</code>	46, 49, <u>71</u> , 76
<code>\ID@startrue</code>	34
<code>\ID@switch</code>	139, <u>139</u> , 140
<code>\ID@toks</code>	8, <u>8</u> , 18, 52, 72, 77, 80, 86, 92, 100, 104, 120, 123, 130, 136, 181, 194, 197, 200, 201, 243, 244, 249, 254, 257, 260
<code>\ID@trygroup</code>	176, 186, <u>186</u> , 187
<code>\ID@unspace</code> ..	53, 87, <u>180</u> , 182, 184, 185
<code>\if@test</code>	27, 60, 65, 256
<code>\ifID@aborted</code>	4, <u>4</u> , 122, 125, 242
<code>\ifID@bang</code>	<u>4</u> , 7, 30, 38, 121, 216
<code>\ifID@star</code>	<u>4</u> , 5, 29, 146
<code>\ifID@starstar</code>	<u>4</u> , 6, 40, 147
<code>\Inline</code>	<i>1</i> , 3, 10, <u>10</u> , 26, 61, 66, 217, 225, 234
<code>\Inline*</code>	<i>3</i>
<code>\Inline**</code>	<i>3</i>
<code>\Inline!</code>	<i>3</i>
M	
<code>\MultiExpand</code>	<i>2</i> , 14, 163, 252
N	
<code>\next</code> ..	25, 30, 33, 40, 46, 49, 52, 62, 66, 67, 82–84, 86, 89, 141, 143,

148, 150, 155, 158, 161, 164, 167, 170, 173, 176, 178, 189, 191	
<code>\NoExpand</code>	2, 15, 166, 170
Q	
<code>\Q@END</code>	16, 83, 97, 99, 111, 114, 118, 123, 142, 195, 265
R	
<code>\Recurse</code>	2, 15, 169
S	
<code>\Super</code>	2, 13, 112, 154, 210, 211, 233
T	
<code>\the</code>	52, 72, 77, 86, 92, 100, 104, 133, 181, 197, 200, 201, 244, 249, 257, 260
U	
<code>\UnsafeExpand</code>	2, 14, 160
X	
<code>\xa</code>	<u>1</u>