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 problemmatic 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`

`\Expand`

```
\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.

`\Expand`

- `\Expand` - 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 `\expandafter`s may be used to expand a different token), and the outermost grouping braces are discarded.

`\MultiExpand`

- `\MultiExpand{`⟨*number*⟩`}` - Expands the following token or group the given number of times. For example,

```
\MultiExpand{3}{\expandafter\expandafter\expandafter\a
  \expandafter\b\c}
```

  expands first `\c`, then `\b`, then `\a`, 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 `\expandafter` three times, which is clearly wrong.

`\UnsafeExpand`

- `\UnsafeExpand` - This version simply inserts an `\expandafter`, performing the expansion as in `\Expand` above, but reinserting the result back into the stream to be processed. Thus, any tokens like `\Expand` or `\Super` in the expansion will be acted on. Unlike the previous two commands, groups are *not* treated differently.

`\NoExpand`

- `\NoExpand` - If a token is preceeded by `\NoExpand` then it is inserted in the definition exactly as-is. This is required to insert any of the special tokens `\Expand`, `\NoExpand`, etc, as well as the internal token `\Q@END`, into a definition. If the token immediately following `\NoExpand` 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.

`\Super`

- `\Super` - When redefining an already existing macro, `\Super` 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.

`\Recurse`

- `\Recurse` - This is complementary to `\Super` and, while not strictly necessary, is included for clarity. `\Recurse` is equivalent to `\NoExpand\macro` when defining `\macro`. However, since `\macro` 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 preceeded 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.

```
⟨*package⟩
\makeatletter
\let\xa\expandafter
```

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

```
\newif\ifID@aborted
\newif\ifID@star
\newif\ifID@starstar
\newif\ifID@bang
```

`\ID@toks` `\ID@count` 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`.

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

`\Inline` `\ID@scandef` 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 LaTeX or TeX 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`?

```
\DeclareRobustCommand\Inline{%
  \begingroup
  % Define a few ``quarks''
  \def\Expand{\Expand}\def\Super{\Super}%
  \def\UnsafeExpand{\UnsafeExpand}\def\MultiExpand{\MultiExpand}%
  \def\Recurse{\Recurse}\def\NoExpand{\NoExpand}%
  \def\Q@END{\Q@END}%
  % Define a toks register
  \ID@toks{}%
  % Signal that we need to look for a star
  \@testtrue\ID@starfalse\ID@starstarfalse\ID@bangfalse
  % Start scanning for \def or \gdef
  \futurelet\@foo\ID@scandef
}
\newcommand\ID@scandef{%
  \let\next\ID@saveprefix % Default behavior
  % If this is the first few tokens after the \Inline, check for * or !
  \if@test
```

```
    \ifx\@foo*%
      \ifID@star
        \ifID@bang\let\next\ID@sd@lastcheck\else\let\next\ID@sd@checkagain\fi
        \ID@starstartrue
      \else
        \let\next\ID@sd@checkagain
        \ID@startrue
      \fi
    \fi
    \ifx\@foo!%
      \ifID@bang\else % two bangs - can this be anything but an error?
        \ID@bangtrue
        \xa\let\xa\next\ifID@starstar\ID@sd@lastcheck\else\ID@sd@checkagain\fi
      \fi
    \fi
  \fi
  % Now look for a \def or \gdef
  \ifx\@foo\def
    \def\next{\ID@start\def}%
  \fi
  \ifx\@foo\gdef
    \def\next{\ID@start\gdef}%
  \fi
  \ifcat\noexpand\@foo\space
    \def\next{\ID@toks\xa\xa\xa{\xa\the\xa\ID@toks\space}%
      \xa\futurelet\xa\@foo\xa\ID@scandef\ID@unspace}% copied from ID@space
  \fi
  % Error checking (minimal)
  \@testfalse
  \ifx\@foo\edef\@testtrue\fi\ifx\@foo\xdef\@testtrue\fi
  \ifx\@foo\newcommand\@testtrue\fi\ifx\@foo\renewcommand\@testtrue\fi
  \ifx\@foo\DeclareRobustCommand\@testtrue\fi
  \if@test\PackageError{inlinedef}{Only \protect\def\space and \protect\gdef\space are
    allowed after \protect\Inline,\MessageBreak but some other type of
    definition was found}\@eha\let\next\ID@abort\fi
  \@testfalse
  \ifx\@foo\bgroup\@testtrue\fi\ifx\@foo\let\@testtrue\fi
  \if@test\PackageError{inlinedef}{No \protect\def\space or \protect\gdef\space found
    after \protect\Inline}\@ehd\def\next{\ID@abort{}}\fi
  \next
}
```

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

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

`\ID@saveprefix` `\ID@abort` `\ID@start` 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`.

```
\newcommand*\ID@saveprefix[1]{%
  \ID@toks\xa{\the\ID@toks#1}%
  \futurelet\@foo\ID@scandef
}
```

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

```
\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.

```
\newcommand\ID@start{}\def\ID@start#1#2#3#4#{%
  \xa\def\xa\ID@prefix\xa{\the\ID@toks#1}%
  \ID@def#3{#4}%
}
```

`\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.

```
\newcommand*\ID@fixparams{\begingroup\ID@toks{}\futurelet\@foo\ID@fp@start}
\newcommand*\ID@fp@start{%
  \let\next\ID@fp@normal
  \ifx\@foo\Q@END\let\next\ID@fp@end\fi
 \ifcat\noexpand\@foo##\let\next\ID@fp@param\fi % was \ifx\@foo - broken?
  \ifcat\noexpand\@foo\space
    \def\next{\ID@toks\xa\xa\xa{\xa\the\xa\ID@toks\space}%
      \xa\futurelet\xa\@foo\xa\ID@fp@start\ID@unspace}% copied from ID@space
  \fi
  \next
}
```

`\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.

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

```
  \futurelet\@foo\ID@fp@start
}
\newcommand*\ID@fp@end[1]{%
  \xa\endgroup\xa\def\xa\ID@fixedparams\xa{\the\ID@toks}%
}
```

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.

```
\newcommand\ID@def[3]{%
  % Other definitions
  \global\ID@abortedfalse
  \let\@reservedc#1%
  \def\@macroname{#1}% for error message
  \ID@fixparams#2\Q@END
  % These are used by \Super but easier to define here
  \def\@reservedb#2{}%
  \edef\@reservedb{\xa\ID@getprefix\meaning\@reservedb\Q@END}%
  \ifx#1\undefined % hopefully nobody's going around defining \undefined
    \let\@reserveda\undefined
  \else
    \edef\@reserveda{\xa\ID@getprefix\meaning#1\Q@END}%
  \fi
  % Scan it all into \ID@toks
  \ifID@bang\else\ID@checkusage\fi
  \ifID@aborted\else
    \ID@toks{}\ID@scan#3\Q@END{}% we need the {} so that the the #1# works...
  \fi
  \ifID@aborted
    \def\command{}% gracefully ignore
  \else
    \let#1\relax % don't want it expanded in the |\edef| below
    % We don't need to worry about scope anymore
    \toks0\ID@toks % likely redundant, but what if ID@toks=1 or 2?
    \toks1\xa{\ID@prefix}% (easiest way to avoid expansion...)
    \toks2{#2}%
    \edef\command{\the\toks1#1\the\toks2{\the\toks0}}%
    % We could also write this with 3 levels of \xa...
  \fi
```

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

\ID@scan
\ID@switch

This is the main loop. We look at each token in turn and deal with it, mostly 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.

```
\newcommand\ID@scan{\futurelet\@foo\ID@switch}
\newcommand\ID@switch{%
  \let\next\ID@normal
  \ifx\@foo\Q@END
    \let\next\@gobble
  \fi
  \ifx\@foo\@reservedc % macro name... what to do?
    \ifID@star
      \ifID@starstar
        \let\next\ID@expandmacro
      \else
        \let\next\ID@expandsuper
      \fi
    \fi
  \fi
  \ifx\@foo\Super
    \let\next\ID@expandsuper
  \fi
  \ifx\@foo\Expand
    \let\next\ID@expandnext
  \fi
  \ifx\@foo\UnsafeExpand
    \let\next\ID@expandunsafe
  \fi
  \ifx\@foo\MultiExpand
    \let\next\ID@expandmulti
  \fi
  \ifx\@foo\NoExpand
    \let\next\ID@noexpandnext
  \fi
  \ifx\@foo\Recurse
    \def\next{\xa\xa\xa\ID@scan\xa\xa\xa\NoExpand\xa\@macroname\@gobble}%
  \fi
  \ifcat\noexpand\@foo\space
    \let\next\ID@space
  \fi
  \ifcat\noexpand\@foo\bgroup
    \let\next\ID@trygroup
  \fi
```

```
  \next
}
```

\ID@space
\ID@unspace

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.

```
\newcommand\ID@space{%
  \ID@toks\xa\xa\xa{\xa\the\xa\ID@toks\space}%
  \xa\ID@scan\ID@unspace
}
\newcommand\ID@unspace{}
\xa\def\xa\ID@unspace\space{}
```

\ID@trygroup
\ID@recurse

The next two macros are used to check if the `\bgroup` token was an explicit or 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.

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

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`.

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

\ID@normal
ID@noexpandnext

This is what we do when it's not anything special.

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

`\ID@checkusage` `\ID@checkredef`

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

```
\newcommand*\ID@checkusage{%
  % Make sure parameter lists are the same, does nothing if undefined
  \ifx\@reserveda\@reservedb
  \else
    % Error messages
    \ifx\@reserveda\undefined % undefined - okay
    \else
      \global\ID@abortedtrue
      \ifx\@foo\Super
        \PackageError{inlinedef}{Cannot use \protect\Super\space in \expandafter
          \protect\@macroname\space because\MessageBreak
          parameter lists don't match:\MessageBreak
          `\@reservedb' (new) != `\@reserveda' (old)}\@eha
      \else
        \ifID@bang % auto-expansion forbidden
          \PackageError{inlinedef}{Cannot use \protect\Inline* auto-expansion in
            \expandafter\protect\@macroname\MessageBreak
            because parameter lists don't match:\MessageBreak
            `\@reservedb' (new) != `\@reserveda' (old)}\@eha
        \else
          \PackageError{inlinedef}{Parameter lists for
            \expandafter\protect\@macroname\space don't match:\MessageBreak
            `\@reservedb' (new) != `\@reserveda' (old)\MessageBreak
            Use !-form of \protect\Inline\space to ignore this}\@eha
        \fi
      \fi
    \fi
  \fi
}
\newcommand*\ID@checkredef{%
  \ifx\@reserveda\undefined % undefined - okay
    \PackageError{inlinedef}{Cannot use \ifx\@foo\Super\protect\Super\space
      \else\protect\Inline** \fi in \expandafter\protect\@macroname\space
      because \MessageBreak it hasn't been defined yet}%
      \@eha
    \global\ID@abortedtrue
  \fi
}
```

`\ID@expandsuper` `\ID@expandnext` `\ID@expandmulti` `\ID@expandunsafe` `\ID@expandmacro`

These correspond to the two special tokens, `\Super` and `\Expand`. The first one tests that the parameter list is alright and that the original command wasn't undefined. If all is well, it expands everything in the right order. The second one is simpler, just inserting an `\expandafter` before the continuation (`\ID@scan`) to 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.

```
\newcommand*\ID@expandsuper[1]{%
  \ID@checkusage\ID@checkredef
  \ifID@aborted\else
    \ID@toks\xa\xa\xa\xa\xa\xa\xa
      {\xa\xa\xa\the\xa\xa\xa\ID@toks\xa\@reservedc\ID@fixedparams}%
  \fi
  \ID@scan
}
\newcommand\ID@expandnext[2]{%
  \ID@toks\xa\xa\xa{\xa\the\xa\ID@toks#2}\ID@scan
}
\newcommand\ID@expandmulti[3]{%
  \begingroup % #1 is the \MultiExpand...
    \ID@count#2\relax % this will need to be allocated too!
    \ID@toks{#3}%
    \@testtrue\ifnum\ID@count<\@ne\@testfalse\fi
    \@whilesw\if@test\fi{%
      \ID@toks\xa\xa\xa{\the\ID@toks}% one expansion...
      \advance\ID@count\m@ne\ifnum\ID@count<\@ne\@testfalse\fi
    }%
  \xa\endgroup\xa\ID@toks\xa\xa\xa{\xa\the\xa\ID@toks\the\ID@toks}\ID@scan
}

\newcommand*\ID@expandunsafe[1]{\expandafter\ID@scan}
\newcommand*\ID@expandmacro[1]{\expandafter\ID@scan\@reservedc}
```

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

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

Finally we clean up by restoring @'s catcode.

```
\makeatother
```

⟨/package⟩

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.

```
⟨*testsuite⟩
\makeatletter
\errorcontextlines=10
\def\WantError#1#2#3{%
  \let\WE@packageerror\PackageError
  \def\PackageError##1##2##3{%
    \protected@edef\@goterror{##2}\protected@edef\@wanterror{#2}%
    \edef\@goterror{\xa\detokenize\xa{\@goterror}}%
    \edef\@wanterror{\xa\detokenize\xa{\@wanterror}}%
    \protected@edef\@gotpackage{##1}\protected@edef\@wantpackage{#1}%
    \edef\@gotpackage{\xa\detokenize\xa{\@gotpackage}}%
    \edef\@wantpackage{\xa\detokenize\xa{\@wantpackage}}%
    \global\let\PackageError\WE@packageerror
    \@tempswafalse
    \ifx\@gotpackage\@wantpackage\else\message{^^J(arg 1 differs)^^J}\@tempswatrue\fi
    \ifx\@goterror\@wanterror\else\message{^^J(arg 2 differs)^^J}\@tempswatrue\fi
    \ifx#3##3\else\message{^^J(arg 3 differs)^^J}\@tempswatrue\fi
    \if@tempswa\PackageError{inlinedef (test)}{wrong error}\@eha\PackageError{##1}{##2}##3\f
  }%
}

\def\CheckError{%
  \ifx\PackageError\WE@packageerror\else
    \PackageError{inlinedef (test)}{expected error not thrown}\@eha\fi
  \global\let\PackageError\WE@packageerror
}
\newcommand\CheckDefinition[1][]{\@CheckDefinition{#1}}
\def\@CheckDefinition#1#2#3#{\@checkdefn{#1}#2{#3}}
\def\@checkdefn#1#2#3#4{#1\def\@reserveda#3{#4}\ifx#2\@reserveda\else
  \message{^^J^^J\meaning#2^^J(got)vs(wanted)^^J\meaning\@reserveda^^J^^J}
  \PackageError{inlinedef (test)}{definition of \detokenize{#2}didn't match}\@eha\fi
}
\let\eha\@eha\let\ehd\@ehd
\makeatother
```

Here we predefine copies of the errors so that we can look for them easily

```
\catcode`\#=12
\def\pound{#}
\catcode`\#=6

\def\WantSuperNoMatch#1#2#3{%
  \WantError{inlinedef}{Cannot use \protect\Super\space in
    \protect#1\space because\MessageBreak
    parameter lists don't match:\MessageBreak
    `#3' (new) != `#2' (old)}\eha
```

```
}
\def\WantStarNoMatch#1#2#3{%
  \WantError{inlinedef}{Cannot use \protect\Inline* auto-expansion in
    \protect#1\MessageBreak because
    parameter lists don't match:\MessageBreak
    `#3' (new) != `#2' (old)}\eha
}
\def\WantNoMatchBang#1#2#3{%
\WantError{inlinedef}{Parameter lists for
  \protect#1\space don't match:\MessageBreak
  `#3' (new) != `#2' (old)\MessageBreak
  Use !-form of \protect\Inline\space to ignore this}\eha
}
\def\WantOnlyDefGdef{%
  \WantError{inlinedef}{Only \protect\def\space and \protect\gdef\space are
    allowed after \protect\Inline,\MessageBreak but some other type of
    definition was found}\eha
}
\def\WantNoDefGdef{%
  \WantError{inlinedef}{No \protect\def\space or \protect\gdef\space found
    after \protect\Inline}\ehd
}
\def\WantSuperNoRedef#1{%
  \WantError{inlinedef}{Cannot use \protect\Super\space in \protect#1\space
    because \MessageBreak it hasn't been defined yet}\eha
}
```

Now we start the actual tests.

```
% I. Basic stuff
%    A. Simple definition
\let\a\undefined
\Inline\def\a{b}
\CheckDefinition\a{b}

%    B. Simple redefinition
\def\a{b}
\Inline\def\a{d}
\CheckDefinition\a{d}

%    C. Erroneous redefinition (needs !)
\def\a{b}
\WantNoMatchBang\a{}{\pound1}
\Inline\def\a#1{c}
\CheckError
\CheckDefinition\a{b} % shouldn't have changed

\def\a{b}
\Inline!\def\a#1{c}
\CheckDefinition\a#1{c}

```

```
 %    D. Local/global definition
\def\a{b}
\begingroup
\Inline\def\a{c}
\endgroup
\CheckDefinition\a{b}

\begingroup
\Inline\gdef\a{c}
\endgroup
\CheckDefinition\a{c}

{\Inline\global\def\a{d}}
\CheckDefinition\a{d}

 %    E. Collecting arguments
\Inline\long\def\a{e}
\CheckDefinition[\long]\a{e}

\Inline\outer\def\a{f}
\edef\a{\meaning\a}
\edef\b{\detokenize{\outer macro:->f}}
\xa\CheckDefinition\xa\a\xa{\b}

\Inline\long\outer\def\a{g}
\edef\a{\meaning\a}
\edef\b{\string\long\string\outer\space\detokenize{macro:->g}}
\xa\CheckDefinition\xa\a\xa{\b}

\def\a{g}
\Inline!\long\def\a#1{h}
\CheckDefinition[\long]\a#1{h}

 % II. Special tokens
 %    A. Recursion
\def\a{b}
\Inline\def\a{a\a c}
\CheckDefinition\a{a\a c}

 %    B. Expansion
\def\a{b}
\Inline\def\a{a\Expand\a c}
\CheckDefinition\a{abc}

\def\a{b}
\Inline\def\a{\Expand a\Expand\a\Expand c}
\CheckDefinition\a{abc}

\def\a{\b}
\def\b{c}
```

```
\Inline\def\a{a\Expand\a c}
\CheckDefinition\a{a\b c}

\toks0{b}\toks1{d}
\Inline\def\a{a\the\toks0c\the\toks1e}
\CheckDefinition\a{a\the\toks0c\the\toks1e}

\Inline\def\a{a\Expand{\the\toks0}c\Expand{\the\toks1}e}
\CheckDefinition\a{abcde}

\Inline\def\a{\Expand{a\the\toks0}c\Expand{\the\toks1}e}
\CheckDefinition\a{a\the\toks0cde}

\Inline\def\a{\Expand{\expandafter a\the\toks0}c\Expand{\the\toks1e}}
\CheckDefinition\a{abcde}

 %    C. MultiExpand
\def\x{\y}
\def\y{\z}
\def\z{0}
\Inline\def\a{a\MultiExpand0\x b}
\CheckDefinition\a{a\x b}

\Inline\def\a{a\MultiExpand1\x b}
\CheckDefinition\a{a\y b}

\Inline\def\a{a\MultiExpand2\x b}
\CheckDefinition\a{a\z b}

\Inline\def\a{a\MultiExpand3\x b}
\CheckDefinition\a{a0b}

\Inline\def\a{a\MultiExpand{10}\x b}
\CheckDefinition\a{a0b}

 %      i. use with \expandafter
\Inline\def\a{a\MultiExpand2{\expandafter\expandafter\x\x}b}
\CheckDefinition\a{a\y\y b}

\Inline\def\a{a\MultiExpand1{\expandafter\expandafter\x\x}b}
\CheckDefinition\a{a\expandafter\y\x b}

\Inline\def\a{a\MultiExpand2{\expandafter\expandafter\expandafter\x\x}b}
\CheckDefinition\a{a\x\z b}

\Inline\def\a{a\MultiExpand3{\expandafter\expandafter\expandafter\x\x}b}
\CheckDefinition\a{a\y\z b}

\Inline\def\a{a\MultiExpand4{\expandafter\expandafter\expandafter\x\x}b}
\CheckDefinition\a{a\z\z b}
```

```
459
460 \Inline\def\a{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{a\Expand\x d}
466 \CheckDefinition\a{ab\Super cd}
467
468 \def\a{0}
469 \Inline\def\a{a\UnsafeExpand\x d}
470 \CheckDefinition\a{ab0cd}
471
472 \def\a{b\Super d}
473 \Inline\def\a{a\UnsafeExpand\a e}
474 \CheckDefinition\a{abb\Super dde}
475
476 \def\a{b\Super d}
477 \Inline**\def\a{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\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{a\UnsafeExpand\x ce}
488 \CheckDefinition\a{abcde}
489
490  %    E. NoExpand
491 \Inline\def\a{a\NoExpand\Expand\x b}
492 \CheckDefinition\a{a\Expand\x b}
493
494 \Inline*\def\a{a\NoExpand\a b}
495 \CheckDefinition\a{a\a b}
496
497 \Inline**\def\a{a\NoExpand\a b}
498 \CheckDefinition\a{a\a b}
499
500 \Inline\def\a{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{a\Super e}
506 \CheckDefinition\a{abcde}
507
508 \def\a#1{b#1d}
```

```
\Inline\def\a#1{a\Super e}
\CheckDefinition\a#1{ab#1de}

\def\a#1{b#1d}
\Inline*\def\a#1{a\Super e}
\CheckDefinition\a#1{ab#1de}

\def\a#1{b#1d}
\Inline**\def\a#1{a\Super e}
\CheckDefinition\a#1{ab#1de}

 %    G. Recurse
\def\a{q}
\Inline\def\a{a\Recurse b}
\CheckDefinition\a{a\a b}

\Inline*\def\a{a\Recurse b}
\CheckDefinition\a{a\a b}

\Inline**\def\a{a\Recurse b}
\CheckDefinition\a{a\a b}

 % III. Tricky parsing
 %    A. Spaces
\def\a{b c d}
\Inline\def\a{a \Super e}
\CheckDefinition\a{a b c de}

\def\a{b  c d}
\Inline\def\a{a \Expand\a e}
\CheckDefinition\a{a b c de}

\def\a{b  c d}
\Inline\def\a{a \Expand{\a} e}
\CheckDefinition\a{a b c d e}

\Inline\def\a{a\NoExpand{\Expand\x\Expand\y} b}
\xa\CheckDefinition\xa\a\xa{\xa a\xa\Expand\xa\x\xa\Expand\xa\y\space b}

 %    B. Grouping
\def\a{b{c d}e}
\Inline\def\a{{a\Super}f\Super}
\CheckDefinition\a{{ab{c d}e}fb{c d}e}

\Inline\def\a{{ }{}}
\Inline\def\a{{\Expand\a\a}{} {{\Super}{}}}
\CheckDefinition\a{{{ }{}\a}{} {{{ }{}}{}}}

 %    C. Parameters
\def\a#1bcd#2{[#1...#2]}
```

```
\Inline\def\a#1bcd#2{a\Super b}
\CheckDefinition\a#1bcd#2{a[#1...#2]b}

\def\a#1\##2{y}
\Inline\def\a#1\##2{x\UnsafeExpand\a{#1}\#{#2}z}
\CheckDefinition\a#1\##2{xyz}

\def\a#1\##2{#1y#2}
\Inline\def\a#1\##2{x\UnsafeExpand\a{#1}\#{#2}z}
\CheckDefinition\a#1\##2{x#1y#2z}

 %      i. spaces!
\def\a #1 {y}
\Inline\def\a#1 {x\Super z}
\CheckDefinition\a#1 {xyz}

\xa\def\xa\a\space{y}
\xa\Inline\xa\def\xa\a\space{x\Super z}
\xa\CheckDefinition\xa\a\space{xyz}

 %      ii. funky catcodes
 %%%% This test fails.
 %\begingroup
 %  \catcode`&=6
 %  \def\a&1{b#1d}
 %  \Inline\def\a#1{a\Super e}
 %  \CheckDefinition\a#1{b&1d}
 %\endgroup

 %    D. Active characters
\begingroup
  \catcode`A=13
  \defA#1{b#1d}
  \Inline\defA#1{aAe}
  \CheckDefinitionA#1{aAe}

  \defA#1{b#1d}
  \Inline*\defA#1{aAe}
  \CheckDefinitionA#1{ab#1de}
\endgroup

 % IV. Auto-expansion

\def\a#1{y}
\Inline\def\a#1{x\a z}
\CheckDefinition\a#1{x\a z}

\def\a#1{y}
\Inline*\def\a#1{x\a z}
\CheckDefinition\a#1{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
```

```
\WantNoDefGdef
\Inline\let\relax\relax
\CheckError

\WantNoDefGdef
\Inline{}
\CheckError

\WantSuperNoRedef\foo
\Inline\def\foo#1{a \Expand\x\space cd #1 fg \x\space i\Super}
\CheckError

\WantNoMatchBang\a{}{\pound1}
\def\a{b}
\Inline\def\a#1{a\a c}
\CheckError

 % Miscellaneous (read: "old") tests

\def\test#1{d #1 f}
\Inline\def\test#1{a \Expand\x\space c\Super g \x\space i}
\CheckDefinition\test#1{a b\space cd #1 fg \x\space i}

\Inline*\def\bar#1{a \Expand\x\space c\bar g \x\space i}
\CheckDefinition\bar#1{a b\space cd #1 fg \x\space i}

\def\bar#1{d #1 f}
\Inline**\def\bar#1{a \Expand\x\space c\bar{#1}g \x\space i}
\CheckDefinition\bar#1{a b\space cd #1 fg \x\space i}

\Inline\def\foo#1{a \Expand\x\space cd #1 fg \x\space i}
\CheckDefinition\foo#1{a b\space cd #1 fg \x\space i}

\def\a{b}
\Inline!\def\a#1{a\Expand\a c}
\CheckDefinition\a#1{abc}

\def\a{b}
\Inline!**\def\a#1{a\a c}
\CheckDefinition\a#1{abc}

\Inline\def\a#1{a\a c}
\CheckDefinition\a#1{a\a c}

\Inline\def\a#1{a\Recurse c}
\CheckDefinition\a#1{a\a c}

\Inline!\def\a{a\NoExpand{b\Super c}d}
\CheckDefinition\a{ab\Super cd}

```

```
\Inline*\def\a{gh\a jk}
\CheckDefinition\a{ghab\Super cdjk}

 % SURPRISE!  unsafe expansion...
\def\a{ab\Super cd}
\Inline**\def\a{gh\a jk}
\CheckDefinition\a{ghabab\Super cdcdjk}

\def\a{ab\Super cd}
\Inline\def\a{gh\UnsafeExpand\a jk}
\CheckDefinition\a{ghabab\Super cdcdjk}

\def\x{\x a} % This is a fun one...!
\Inline\def\a{\MultiExpand{5}\x}
\CheckDefinition\a{\x aaaaa}

\message{^^JAll tests completed.^^J}

\begin{document}
\end{document}
```
⟨/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.