tokcycle Package Examples

February 4, 2020

Contents

1 Examples, examples, and more examples

Often, the best way to learn a new tool is to see examples of it being used. Here, a number of examples are gathered that span the spectrum of tokcycle usage.

1.1 Application basics

1.1.1 Using the CGMS directives

Apply different directives to Characters (under-dot), Groups (visible braces), Macros (boxed, detokenized), and Spaces (visible space).

The \underdot macro

```
\newcommand\underdot[1]{\ooalign{#1\cr\hfil{\raisebox{-5pt}{.}}\hfil}}
```

```
\tokcycle{\addcytoks{\underdot{#1}}}
         {\addcytoks{\{}\processtoks{#1}
          \addcytoks{\}}}
         {\addcytoks{\fbox{\detokenize{#1}}}}
         {\addcytoks{\textvisiblespace}}
{This \textit{is \textbf{a}} test.}
\the\cytoks
```

This␣\textit{is␣\textbf{a}}␣test.

1.1.2 Escaping text

Text between two successive escape characters is bypassed by tokcycle and instead echoed to the output register. Default escape character is |. One can change it with `\settcEscapechar` macro.

The unexpandable `\plusl` macro

```
\newcommand\plusl[1]{\char\numexpr`#1+1\relax}
```

Escaping text in the input stream

```
\tokcycle
{\addcytoks{\plusl{#1}}}
{\processtoks{#1}}
{\addcytoks{#1}}
{\addcytoks{#1}}
{This \fbox{code is a test
  |(I can also escape text)|}
  of |\rule{1em}{.5em}|
  {\bfseries mine}.}
\the\cytoks
```

Uijt dpef jt b uftu (I can also escape text) pg **njof**/

1.1.3 Unexpandable, unexpanded, and expanded Character directives

This section concerns the issue of whether the characters of the input stream are transformed before or after being placed in the output token register (`\cytoks`).

Transform characters (+1 ASCII) via unexpandable macro:

Unexpandable Character directive

```
\tokcycle
{\addcytoks{\plusl{#1}}}
{\processtoks{#1}}
{\addcytoks{#1}}
{\addcytoks{#1}}{%
  This \textit{code \textup{is} a test} of mine.}
\the\cytoks
```

Uijt *dpef* jt *b* *uftu* pg njof/

`\cytoks` *alt*detokenization:

```
\plusl{T}\plusl{h}\plusl{i}\plusl{s} \textit{\plusl{c}\plusl{o}\plusl{d}\plusl{e} \textup{\plusl{i}
\plusl{s}} \plusl{a} \plusl{t}\plusl{e}\plusl{s}\plusl{t}} \plusl{o}\plusl{f} \plusl{m}\plusl{i}
\plusl{n}\plusl{e}\plusl{.}
```

Capitalize vowels (but don't expand the character directive)

The expandable `\vowelcap` macro

```
\newcommand\vowelcap[1]{%
  \ifx a#1A\else
  \ifx e#1E\else
  \ifx i#1I\else
  \ifx o#1O\else
  \ifx u#1U\else
  #1\fi\fi\fi\fi\fi
}
```

Not expanded Character directive

```
\tokcycle
{\addcytoks{\vowelcap{#1}}}
{\processtoks{#1}}
{\addcytoks{#1}}
{\addcytoks{#1}}{%
  This \textit{code \textup{is} a test} of mine.}
\the\cytoks
```

ThIs *cOdE* Is *A tEst* Of mInE.

`\cytoks` *alt*detokenization:

```
\vowelcap{T}\vowelcap{h}\vowelcap{i}\vowelcap{s} \textit{\vowelcap{c}\vowelcap{o}\vowelcap{d}
\vowelcap{e} \textup{\vowelcap{i}\vowelcap{s}} \vowelcap{a} \vowelcap{t}\vowelcap{e}\vowelcap{s}
\vowelcap{t}} \vowelcap{o}\vowelcap{f} \vowelcap{m}\vowelcap{i}\vowelcap{n}\vowelcap{e}\vowelcap{.}
```

Capitalize vowels (expanding the character directive)

Expanded Character directive

```
\tokcycle
{\addcytoks[x]{\vowelcap{#1}}}
{\processtoks{#1}}
{\addcytoks{#1}}
{\addcytoks{#1}}{%
  This \textit{code \textup{is} a test} of mine.}
\the\cytoks
```

ThIs *cOdE* Is *A tEst* Of mInE.

`\cytoks` *alt*detokenization:

```
ThIs \textit{cOdE \textup{Is} A tEst} Of mInE.
```

1.1.4 Unexpanded vs. pre-expanded input stream

Normal token cycle (input stream not pre-expanded)

```
\tokcycle
{\addcytoks[x]{\vowelcap{#1}}}
{\processtoks{#1}}
{\addcytoks{#1}}
{\addcytoks{#1}}%
{This \fbox{code
  is a test \today} of
  {\bfseries mine}.}
\the\cytoks
```

ThIs cOdE Is A tEst February 4, 2020 Of **mInE**.

`\cytoks` *alt*detokenization:

```
ThIs \fbox{cOdE Is A tEst \today} Of {\bfseries mInE}.
```

Note that, when pre-expanding the input stream, one must `\noexpand` the macros that are *not* to be pre-expanded.

Pre-`\expanded` token cycle input stream

```
\expandedtokcyclexpress
{This \noexpand\fbox{code
  is a test \today} of
  {\noexpand\bfseries mine}.}
\the\cytoks
```

ThIs cOdE Is A tEst FEbrUAry 4, 2020 Of **mInE**.

`\cytoks` *alt*detokenization:

```
ThIs \fbox{cOdE Is A tEst FEbrUAry 4, 2020} Of {\bfseries mInE}.
```

1.2 Grouping

Differentiating explicit groups, e.g., {...}, from implicit groups, e.g. `\bgroup`...`\egroup`, is done automatically by tokcycle. The user has options on how tokcycle should treat these tokens. The desired options are to be set prior to the tokcycle invocation.

1.2.1 Treatment options for implicit groups

The macro `\stripimplicitgroupingcase` can take three possible integer arguments: 0 (default) to automatically place unaltered implicit group tokens in the output register; 1 to strip implicit group tokens from the output; or -1 to instead pass the implicit group tokens to the Character directive (as implicit tokens) for separate processing (typically, when detokenization is desired).

Using `\stripimplicitgroupingcase` to affect treatment of implicit grouping tokens

```
\resettokcycle
\Characterdirective{\addcytoks[x]{%
  \vowelcap{#1}}}
\def\z{Announcement:
  {\bfseries\bgroup\itshape
  Today \egroup it is} \today,
  a Wednesday}
\expandafter\tokencyclexpress\z
\endtokencyclexpress\medskip

\detokenize\expandafter{\the\cytoks}
\bigskip

\stripimplicitgroupingcase{1}
\expandafter\tokencyclexpress\z
\endtokencyclexpress\medskip

\detokenize\expandafter{\the\cytoks}
```

AnnOUncEmEnt: ***TOdAy* It Is** February 4, 2020, A WEdnEsdAy

AnnOUncEmEnt: {\bfseries \bgroup \itshape TOdAy \egroup It Is} \today , A WEdnEsdAy

AnnOUncEmEnt: ***TOdAy It Is*** February 4, 2020, A WEdnEsdAy

AnnOUncEmEnt: {\bfseries \itshape TOdAy It Is} \today , A WEdnEsdAy

1.2.2 Treatment options for explicit groups

For explicit group tokens, e.g., { }, there are only two options to be had. These are embodied in the if-condition `\ifstripgrouping` (default `\stripgroupingfalse`). Regardless of which condition is set, the tokens within the explicit group are still passed to the Group directive for processing.
Permutations of the following code are used in the subsequent examples. Group stripping, brought about by `\stripgroupingtrue`, involves removing the grouping braces from around the group. The choice of `\processtoks` vs. `\addcytoks` affects whether the tokens inside the group are recommitted to tokcycle for processing, or are merely sent to the output register in their original unprocessed form.
Note that, in these examples, underdots and visible spaces will only appear on characters and spaces that have been directed to the Character and Space directives, respectively. Without `\processtoks`, that will not occur to tokens *inside* of groups.

Code permutations on group stripping and inner-group token processing

```
\stripgroupingfalse  OR  \stripgroupingtrue
\tokcycle{\addcytoks{\underdot{#1}}}
         {\processtoks{#1}}  OR  {\addcytoks{#1}}}
         {\addcytoks{#1}}
         {\addcytoks{\textvisiblespace}}
{This \fbox{is a \fbox{token}} test.}
\the\cytoks
```

`\stripgroupingfalse \processtoks`

This␣ is␣a␣ token ␣test.

`\stripgroupingfalse \addcytoks`

This␣ is a token ␣test.

\stripgroupingtrue \processtoks

This␣i␣s␣a␣t␣oken␣test.

\stripgroupingtrue \addcytoks

This␣i␣s a token␣test.

Note that the content of groups can be altogether eliminated if *neither* `\processtoks{#1}` nor `\addcytoks{#1}` are used in the Group directive.

1.2.3 Group nesting

The \reducecolor and \restorecolor macros

```
\newcounter{colorindex}
\newcommand\restorecolor{\setcounter{colorindex}{100}}
\newcommand\reducecolor[1]{%
  \color{red!\thecolorindex!cyan}%
  \addtocounter{colorindex}{-#1}%
  \ifnum\thecolorindex<1\relax\setcounter{colorindex}{1}\fi}
```

Group nesting is no impediment to tokcycle

```
\restorecolor
\tokcycle
  {\addcytoks{(#1)}}
  {\addcytoks{\reducecolor{11}}%
   \addcytoks{[}\processtoks{#1}%
   \addcytoks{]}}
  {\addcytoks{#1}}
  {}{%
  {1{{3{{5{{7{{9{1{}0}}8}}6}}4}}2}}}
\the\cytoks
```

[(1)[[(3)[[(5)[[(7)[[(9)[(1)[](0)]](8)]](6)]](4)]](2)]]

1.3 Direct use of tokcycle

tokcycle (in regular or xpress form) may be invoked directly from the document, without being first encapsulated within a macro or environment.

1.3.1 Modifying counters as part of the Character directive

Using a period token (.) to reset a changing color

```
\restorecolor
\tokencycle
  {\addcytoks{\bgroup\reducecolor{3}#1\egroup}%
   \ifx.#1\addcytoks{\restorecolor}\fi}
  {\processtoks{#1}}
  {\addcytoks{#1}}
  {\addcytoks{#1}}%
This right \textit{here is a sentence in italic}.
And \textbf{here we have another sentence in bold}.

{\scshape Now in a new paragraph, the sentence
is long.} Now, it is short.
\endtokencycle
```

This right *here is a sentence in italic.* And **here we have another sentence in bold.** NOW IN A NEW PARAGRAPH, THE SENTENCE IS LONG. Now, it is short.

1.4 Macro encapsulation of tokcycle

1.4.1 Spacing out text

The \spaceouttext macro

```
\newcommand\spaceouttext[2]{%
  \tokcycle
    {\addcytoks{##1\nobreak\hspace{#1}}}%
    {\processtoks{##1}}
    {\addcytoks{##1}}%
    {\addcytoks{##1\hspace{#1}}}
    {#2}%
  \the\cytoks\unskip}
```

\spaceouttext demo

```
\spaceouttext{3pt plus 3pt}{This
  \textit{text \textbf{is}
  very} spaced out}. Back
  to regular text.

\spaceouttext{1.5pt}{This
  \textit{text \textbf{is}
  somewhat} spaced out}.
  Back to regular text.
```

T h i s *t e x t* ***i s*** *v e r y* s p a c e d o u t. Back to regular text.
T h i s *t e x t* ***i s*** *s o m e w h a t* s p a c e d o u t. Back to regular text.

1.4.2 Alternate presentation of detokenized content

This macro attempts to give a more natural presentation of `\detokenize`'d material. It is **not** to be confused as a replacement for `\detokenize`. In certain applications, it may offer a more pleasingly formatted typesetting of detokenized material.
It is an unusual application of tokcycle in that it does not actually use the `\cytoks` token register to collect its output. This is only possible because all macros in the input stream are detokenized, rather than executed.

The `\altdetokenize` macro

```
\newif\ifmacro
\newcommand\altdetokenize[1]{\begingroup\stripgroupingtrue\macrofalse
  \tokcycle
    {\ifmacro\def\tmp{##1}\ifcat\tmp A\else\unskip\allowbreak\fi\macrofalse\fi
     \detokenize{##1}}
    {\ifmacro\unskip\macrofalse\fi\{\processtoks{##1}\ifmacro\unskip\fi\}\allowbreak}
    {\tctestifx{\\##1}{\\}{\ifmacro\unskip\allowbreak\fi
     \allowbreak\detokenize{##1}\macrotrue}}
    { \hspace{0pt plus 3em minus .3ex}}
    {#1}%
  \unskip
\endgroup}
```

`\altdetokenize` demo

```
\string\altdetokenize: \\
\texttt{\altdetokenize{a\mac a \mac2
    {\mac}\mac{a\mac\mac}\mac}}!

\string\detokenize: \\
   \texttt{\detokenize{a\mac a \mac2
    {\mac}\mac{a\mac\mac}\mac}}!
```

\altdetokenize:
a\mac a \mac2 {\mac}\mac{a\mac\mac}\mac!
\detokenize:
a\mac a \mac 2 {\mac }\mac {a\mac \mac
}\mac !

1.4.3 Capitalize all words, including compound and parenthetical words

The `\Titlecase` and `\nextcap` macros

```
\newcommand\TitleCase[1]{%
  \def\capnext{T}
  \tokcycle
    {\addcytoks{\nextcap{##1}}}
    {\processtoks{##1}}
    {\addcytoks{##1}}
    {\addcytoks{##1\def\capnext{T}}}
    {#1}%
  \the\cytoks
}
\newcommand\nextcap[1]{%
       \edef\tmp{#1}%
       \tctestifx{-#1}{\def\capnext{T}}{}%
       \tctestifcon{\if T\capnext}%
         {\tctestifcon{\ifcat\tmp A}%
           {\uppercase{#1}\def\capnext{F}}%
           {#1}}%
         {#1}%
}
```

A demo of `\Titlecase` showing raw (escaped) input and processed output

```
\TitleCase{%
|here, {\bfseries\today{}, is [my]}
  really-big-test
  (\textit{capitalizing} words).|

 here, {\bfseries\today{}, is [my]}
  really-big-test
  (\textit{capitalizing} words).}
```

here, **February 4, 2020, is [my]** really-big-test (*capitalizing* words).
Here, **February 4, 2020, Is [My]** Really-Big-Test (*Capitalizing* Words).

1.4.4 Scaling rule dimensions

This example only applies if one can guarantee that the input stream will contain only text and rules...

The `\growdim` macro

```
\newcommand\growdim[2]{%
\tokcycle{\addcytoks{##1}}
         {\addcytoks{#1\dimexpr##1}}
         {\addcytoks{##1}}
         {\addcytoks{##1}}{%
    #2}%
\the\cytoks}
```

Using tokcycle to change `\rule` dimensions

```
\growdim{2}{This rule is exactly 4pt:
  \rule|{4pt}{4pt}| , whereas this
  rule is 2x bigger than 4pt:
  \rule{4pt}{4pt} .}\par
\growdim{4}{This rule is exactly 5pt:
  \rule|{5pt}{5pt}| , whereas this
  rule is 4x bigger than 5pt:
  \rule{5pt}{5pt} .}
```

This rule is exactly 4pt: ■ , whereas this rule is 2x bigger than 4pt: ■ .
This rule is exactly 5pt: ■ , whereas this rule is 4x bigger than 5pt: ■ .

1.4.5 String search, including non-regex material

The `\findinstring` macro for string searches

```
\newcommand\findinstring[2]{\begingroup%
  \stripgroupingtrue
  \setcounter{runcount}{0}%
  \tokcycle
    {\nextctltok{##1}}
    {\nextctltok{\opengroup}\processtoks{##1}\nextctltok{\closegroup}}
    {\nextctltok{##1}}
    {\nextctltok{\tcspace}}
    {#1}%
  \edef\numlet{\theruncount}%
  \expandafter\def\expandafter\searchword\expandafter{\the\cytoks}%
%
  \aftertokcycle{\matchfound}%
  \setcounter{runcount}{0}%
  \def\matchfound{F}%
  \tokcycle
    {\nextcmptok{##1}}
    {\nextcmptok{\opengroup}\processtoks{##1}\nextcmptok{\closegroup}}
    {\nextcmptok{##1}}
    {\nextcmptok{\tcspace}}
    {#2}%
\endgroup}
\newcounter{runcount}
\makeatletter
\newcommand\rotcytoks[1]{\cytoks\expandafter\expandafter\expandafter{%
  \expandafter\tc@gobble\the\cytoks#1}}
\makeatother
\newcommand\testmatch[1]{\ifx#1\searchword\gdef\matchfound{T}\fi}%
\newcommand\rotoradd[2]{\stepcounter{runcount}%
  \ifnum\theruncount>\numlet\relax#1\else#2\fi
  \expandafter\def\expandafter\tmp\expandafter{\the\cytoks}}
\newcommand\nextcmptok[1]{\rotoradd{\rotcytoks{#1}}{\addcytoks{#1}}\testmatch{\tmp}}
\newcommand\nextctltok[1]{\stepcounter{runcount}\addcytoks{#1}}
```

Demo of the `\findinstring` macro

```
1. \findinstring{this}{A test of the times}
   \findinstring{the} {A test of the times}\par
2. \findinstring{This is}{Here, This is a test}
   \findinstring{Thisis} {Here, This is a test}\par
3. \findinstring{the}        {This is the\bfseries{} test}
   \findinstring{he\bfseries}{This is the\bfseries{} test}\par
4. \findinstring{a{bc}} {gf{vf{a{b c}g}gh}hn}
   \findinstring{a{b c}}{gf{vf{a{b c}g}gh}hn}\par
5. \findinstring{a\notmymac{b c}}{gf{vf{a\mymac{b c}g}gh}hn}
   \findinstring{a\mymac{b c}}   {gf{vf{a\mymac{b c}g}gh}hn}\par
6. \findinstring{\textit{Italic}}{this is an \textit{italic} test}
   \findinstring{\textit{italic}}{this is an \textit{italic} test}
```

1. F T
2. T F
3. T T
4. F T
5. F T
6. F T

1.5 tokcycle-based environments

The `\tokcycleenvironment` macro allows users to define their own tokcycle environments. Here are some examples.

1.5.1 "Removing" spaces, but still breakable/hyphenatable

The `\spaceBgone` environment

```
\tokcycleenvironment\spaceBgone
    {\addcytoks{##1}}
    {\processtoks{##1}}
    {\addcytoks{##1}}
    {\addcytoks{\hspace{.2pt plus .2pt minus .8pt}}}%
```

```
\spaceBgone
  Here we have a \textit{test} of
  whether the spaces are removed.
  We are choosing to use the
  tokencycle environment.

  We are also testing the use of
  paragraph breaks in the
  environment.
\endspaceBgone
```

Here we have a *test* of whether the spaces are removed. We are choosing to use the tokencycle environment. We are also testing the use of paragraph breaks in the environment.

1.5.2 Remapping text

The \remaptext environment with supporting macros

```
\tokcycleenvironment\remaptext
    {\addcytoks[x]{\tcremap{##1}}}
    {\processstoks{##1}}
    {\addcytoks{##1}}
    {\addcytoks{##1}}
\newcommand*\tcmapto[2]{\expandafter\def\csname tcmapto#1\endcsname{#2}}
\newcommand*\tcremap[1]{\ifcsname tcmapto#1\endcsname
                \csname tcmapto#1\endcsname\else#1\fi}
\tcmapto am   \tcmapto bf   \tcmapto cz   \tcmapto de   \tcmapto ey
\tcmapto fl   \tcmapto gx   \tcmapto hb   \tcmapto ic   \tcmapto jn
\tcmapto ki   \tcmapto lr   \tcmapto mh   \tcmapto nt   \tcmapto ok
\tcmapto ps   \tcmapto qa   \tcmapto ro   \tcmapto sq   \tcmapto tw
\tcmapto uj   \tcmapto vp   \tcmapto wd   \tcmapto xg   \tcmapto yu
\tcmapto zv
```

Demo of \remaptext

```
\remaptext
What can't we \textit{accomplish} if we try?

Let us be of good spirit and put our minds to it!
\endremaptext
```

Wbmw zmt'w dy *mzzkhsrcqb* cl dy wou?
Lyw jq fy kl xkke qscocw mte sjw kjo hcteq wk cw!

Because \tcremap is expandable, the original text is totally absent from the processed output:

\cytoks *alt*detokenization:

```
Wbmw zmt'w dy \textit{mzzkhsrcqb} cl dy wou?  \par Lyw jq fy kl xkke qscocw mte sjw kjo hcteq wk cw!
```

1.6 Advanced topics: implicit tokens and catcode changes

1.6.1 Trap Active Characters (catcode 13)

Active characters in the tokcycle input stream are processed in their original form. Their active substitutions arising from \defs only occur *afterwards*, when the tokcycle output is typeset. They may be identified with the \ifactivetok test. If \let to a character, they may be identified in the Character directive; If \let to a control sequence or defined via \def, they may be identified in the Macro directive.

Processing active characters

```
\resettokcycle
\tokencyclexpress
This is a test!!\endtokencyclexpress

\catcode`!=\active
\def !{?}
\tokencyclexpress
This is a test!!\endtokencyclexpress

\Characterdirective{\tctestifcon\ifactivetok
  {\addcytoks{\fbox{#1-chr}}}{\addcytoks{#1}}}
\Macrodirective{\tctestifcon\ifactivetok
  {\addcytoks{\fbox{#1-mac}}}{\addcytoks{#1}}}
\tokencyclexpress
This is a test!!\endtokencyclexpress

\catcode`T=\active
\let T+
\tokencyclexpress
This is a test!!\endtokencyclexpress

\detokenize\expandafter{\the\cytoks}
```

This is a test!!
This is a test??
This is a test ?-mac ?-mac
+-chr his is a test ?-mac ?-mac
\fbox {T-chr}his is a test\fbox {!-mac}\fbox {!-mac}

If the input stream is pre-*expanded*, any active substitutions that are expandable (i.e., those involving \def as well as those \let to something expandable) are made before reaching tokcycle processing. They are, thus, no longer detected as active, unless \noexpand is applied before the pre-expansion. In this example, the ! that is not \noexpanded is converted to a ? prior to reaching tokcycle processing (and thus, not detected as \active):

Expanded input stream acts upon active \defed characters unless \noexpand is applied

```
\expandedtokcyclexpress{This is a
    test!\noexpand!}
\the\cytoks\par
\detokenize\expandafter{\the\cytoks}
```

+-chr his is a test?
\fbox {T-chr}his is a test?\fbox {!-mac}

However, pre-tokenization does not suffer this behavior:

Pre-tokenized input stream does not affect active characters

```
\def\tmp{This is a test!!}
\expandafter\tokcyclexpress\expandafter{\tmp}
\the\cytoks\par
\detokenize\expandafter{\the\cytoks}
```

+-chr his is a test ?-mac ?-mac
\fbox {T-chr}his is a test\fbox {!-mac}\fbox {!-mac}

One aspect of TEX to remember is that catcodes are assigned at tokenization; however, for active characters, the substitution assignment is evaluated only upon execution. So, if a cat-13 token is placed into a `\def`, it will remain active even if the catcode of that character code is later changed. But if the cat-13 active definition is changed prior to the execution of the `\def`'ed token, the revised token assignment will apply.
The following example demonstrates this concept, while showing, whithout changing the input in any way, that tokcycle can properly digest active and implicit grouping (cat-1,2) characters:

Active and implicit grouping tokens digestible by tokcycle

```
\catcode`Y=13
\catcode`Z=13
\let Y{
\let Z}
\let\Y{
\let\Z}
\def\tmp{\textit YabcZ de\Y\itshape f\Zg}%

\def Y{\bgroup[NEW]}% APPLIES AT EXECUTION
\catcode`Y=11% DOES NOT AFFECT Y IN \tmp

\expandafter\tokcyclexpress\expandafter{\tmp}
\the\cytoks

\detokenize\expandafter{\the\cytoks}
```

[NEW]abc def*g*
\textit YabcZ de\Y \itshape f\Zg

1.6.2 Trap Catcode 6 (explicit & implicit) tokens

Typically, cat-6 tokens (like #) are used to designate the following digit (1-9) as a parameter. Since they are unlikely to be used in that capacity inside a tokcycle input stream, the package behavior is to convert them into something cat-12 and set the if-condition `\catSIXtrue`. In this manner, `\ifcatSIX` can be used inside the Character directive to convert cat-6 tokens into something of the user's choosing.
As to this cat-12 conversion, explicit cat-6 characters are converted into the same character with cat-12. On the other hand, implicit cat-6 control sequences (e.g., `\let\myhash#`) are converted into a fixed-name macro, `\implicitsixtok`, whose cat-12 substitution text is a `\string` of the original implicit-macro name.

Treatment of cat-6 tokens

```
\resettokcycle
\Characterdirective{\ifcatSIX
  \addcytoks{\fbox{#1}}
  \else\addcytoks{#1}\fi}
\let\myhash#
\tokcyclexpress{This# isQ
  \textit{a Q# test\myhash}!}
\the\cytoks\bigskip\par
\detokenize\expandafter{\the\cytoks}
```

This # isQ *a Q* # *test* \myhash!

This\fbox {#} isQ \textit {a Q\fbox {#} test\fbox {\implicitsixtok }}!

Multiple explicit cat-6 tokens are not a problem

```
\catcode`Q=6
\tokcyclexpress{This# isQ
  \textit{a Q# test\myhash}!}
\the\cytoks
```

This # is Q *a* *Q* # *test* \myhash!

For what is, perhaps, a rare situation, one can even process input streams that contain cat-6 macro parameters. A package macro, `\whennotprocessingparameter#1{`<*directive when not a parameter*>`}`, can be used inside of the Character directive to intercept parameters. In this example, a macro is defined and then executed, subject to token replacements brought about by the expandable Character directive.

Preserving parameters (e.g. `#1`, `#2`) in the tokcycle input stream

```
\Characterdirective{%
  \whennotprocessingparameter#1{%
    \addcytoks[x]{\vowelcap{#1}}}}
\tokcyclexpress{%
  \def\zQ#1#2{[one:#1](two:#2)}
  This is a \zQ big test.

  \renewcommand\zQ[2]{\ifx t#1[#1]\fi(#2)}
  This is a \zQ test.}
\the\cytoks
```

ThIs Is A [OnE:b](twO:I)g tEst.
ThIs Is A [t](E)st.

`\cytoks` *alt*detokenization:

```
\def\zQ#1#2{[OnE:#1](twO:#2)} ThIs Is A \zQ bIg tEst.  \par\renewcommand\zQ[2]{\ifx t#1[#1]\fi(#2)}
ThIs Is A \zQ tEst.
```

1.6.3 Trap implicit tokens in general

Implicit control sequences (assigned via `\let` to characters) were already mentioned in the context of cat-6. However, implicit control sequences can be of any valid catcode (except for cat-0, which we instead call macros or primitives). The condition `\ifimplicittok` is used to flag such tokens for special processing, as well as active tokens that are `\let` to anything unexpandable.
In the next example, implicit, cat-6 and implicit-cat-6 tokens may all be differentiated, shown here with a multiplicity of `\fboxes`.

Implicit = single box, cat-6 = double box, implicit-cat-6 = triple box

```
\catcode`Q=\active \let QN
\let\littlet=t
\let\littlel=l
\let\svhash#
\Characterdirective{\ifimplicittok
  \ifcatSIX\addcytoks{\fbox{\fbox{\fbox{#1}}}}%
  \else\addcytoks{\fbox{#1}}\fi\else\ifcatSIX
  \addcytoks{\fbox{\fbox{#1}}}\else
  \addcytoks{#1}\fi\fi}

\tokencyclexpress We wi\littlel\littlel#
  \textit{ make a \littlet est #} \littlet

This \textit{is a \textbf{big}} \littlet est.

Qext pa#agraph ending with implicit cat six
  \svhash.\endtokencyclexpress
```

We wi[l][l][#] *make a* [*t*]*est* [#] [t]
This *is a* ***big*** [t]est.
[N]ext pa[#]agraph ending with implicit cat six [\svhash].

In the following example, we use both control sequences and active characters in `\def` and `\let` capacities, to demonstrate how tokcycle digests things. Implicit tokens (tokens `\let` to characters) are shown in a box, with both the token name and the implicit value (note that tokens `\let` to macros and primitives are not considered implicit). Active tokens processed through the character directive are followed with a †, whereas those processed through the macro directive are followed with a ‡.

Non-active vs. active `\def` & `\let`

```
\Characterdirective{\ifimplicittok
  \addcytoks{\fbox{\detokenize{#1}:#1}}%
  \else\addcytoks{#1}\fi\ifactivetok
  \addcytoks{\rlap{\dag}}\fi\addcytoks{~\,}}
\Macrodirective{\ifimplicittok
  \addcytoks{\fbox{\detokenize{#1}}}%
  \else\addcytoks{#1}\fi\ifactivetok
  \addcytoks{\rlap{\ddag}}\fi
  \ifx\par#1\else\addcytoks{~\,}\fi}

\def\A{a}
\let\B i
\let\C\today
\let\D\relax
\def\E{\relax}
\catcode`V=13 \def V{a}
\catcode`W=13 \let Ww
\catcode`X=13 \let X\today
\catcode`Y=13 \let Y\relax
\catcode`Z=13 \def Z{\relax}
\tokcyclexpress{\A\B\C\D\E ab\par VWXYZab}
\the\cytoks
```

a [\B :i] February 4, 2020 a b
a‡[W:w]†February 4, 2020‡‡‡a b

If the input stream is subject to pre-expansion, one will require `\noexpand` for macros where no pre-expansion is desired.


If the input stream is provided pre-tokenized via `\def`, TEX convention requires cat-6 tokens to appear in the input stream as duplicate, e.g. `##`.

1.6.4 Changing grouping tokens (catcodes 1,2)

Changing grouping tokens (catcodes 1,2) may require something more, if the output stream is to be detokenized. In the following examples, pay attention to the detokenized grouping around the argument to `\fbox`.
As we will see, the issues raised here only affect the situation when detokenization of the output stream is required.

tokcycle defaults grouping tokens to braces:

```
\tokencycle
 {\addcytoks{(#1)}}
 {\processtoks{#1}}
 {\addcytoks{#1}}
 {\addcytoks{ }}
This \fbox{is a} test.
\endtokencycle\medskip

\detokenize\expandafter{\the\cytoks}
```

(T)(h)(i)(s) (i)(s) (a) (t)(e)(s)(t)(.)
(T)(h)(i)(s) \fbox {(i)(s) (a)} (t)(e)(s)(t)(.)

One can make brackets cat-1,2, redefining bgroup/egroup to [ ]. However, while one can now use brackets in input stream, braces will still appear in the detokenized tokcycle output stream:

tokcycle will not automatically change its grouping tokens

```
\catcode`\[=1
\catcode`\]=2
\let\bgroup[
\let\egroup]
\tokencycle
 {\addcytoks{(#1)}}
 {\processtoks{#1}}
 {\addcytoks{#1}}
 {\addcytoks{ }}
This \fbox[is a] test.
\endtokencycle\medskip

\detokenize\expandafter{\the\cytoks}
```

(T)(h)(i)(s) (i)(s) (a) (t)(e)(s)(t)(.)
(T)(h)(i)(s) \fbox {(i)(s) (a)} (t)(e)(s)(t)(.)

If it is necessary to reflect revised grouping tokens in the output stream, the `\settcgrouping` macro is to be used.

Redefine tokcycle grouping tokens as angle brackets using \settcGrouping

```
\catcode`\<=1
\catcode`\>=2
\catcode`\{=12
\catcode`\}=12
\let\bgroup<
\let\egroup>
\settcGrouping<<#1>>
\tokencycle
 <\addcytoks<(#1)>>
 <\processtoks<#1>>
 <\addcytoks<#1>>
 <\addcytoks< >>
This \fbox<is a> test.
\endtokencycle\medskip

\detokenize\expandafter<\the\cytoks>
```

(T)(h)(i)(s) $\boxed{\text{(i)(s) (a)}}$ (t)(e)(s)(t)(.)

(T)(h)(i)(s) \fbox <(i)(s) (a)> (t)(e)(s)(t)(.)

Angle brackets are now seen in the above detokenization. Until subsequently changed, cat-1,2 angle brackets now appear in detokenized tokcycle groups, even if other cat-1,2 tokens were used in the input stream. Bottom line:

- adding, deleting, or changing catcode 1,2 explicit grouping tokens, e.g., {}, (in conjunction with their associated implicit \bgroup\egroup) tokens will not affect tokcycle's ability to digest proper grouping of the input stream, regardless of which tokens are catcode 1,2 at the moment.
- The grouping tokens used in tokcycle's output default to {} braces (with cat-1,2), but can be changed deliberately using \settcGrouping.
- The package, currently, has no way to reproduce in the output stream the actual grouping tokens that occur in the input stream, but one should ask, for the particular application, if it really matters, as long as the the proper catcodes-1,2 are preserved?

1.6.5 Catcode 10 space tokens

Here we demonstrate that tokcycle can handle arbitrary redesignation of tokens to cat-10, as well as implicit space tokens.

While it should seem natural, we note that implicit space tokens are directed to the Space directive rather than the Character directive. However, \ifimplicittok may still be used to differentiate an explicit space from an implicit one.

Note in the following examples that cat-10 tokens do *not* get under-dots. The next three examples all use the same input, but with different catcode settings for the space and the underscore.

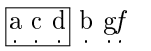
space cat-10, underscore cat-12

```
\catcode`\_=12 %
\catcode`\ =10 %

\tokencycle{\addcytoks{\underdot{#1}}}%
{\processtoks{#1}}%
{\addcytoks{#1}}%
{\addcytoks{#1}}%
\fbox{a_c d} b_g\itshape f\upshape\endtokencycle
```

$\boxed{\text{a}\underline{\ }\text{c d}}$ b$\underline{\ }$g*f*

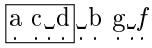
space cat-10, underscore cat-10

```
\catcode`\_=10 %
\catcode`\ =10 %

\tokencycle{\addcytoks{\underdot{#1}}}%
{\processtoks{#1}}%
{\addcytoks{#1}}%
{\addcytoks{#1}}%
\fbox{a_c d} b_g\itshape f\upshape\endtokencycle
```



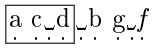
space cat-12, underscore cat-10

```
\catcode`\_=10 %
\catcode`\ =12 %

\tokencycle{\addcytoks{\underdot{#1}}}%
{\processtoks{#1}}%
{\addcytoks{#1}}%
{\addcytoks{#1}}%
\fbox{a_c d} b_g\itshape f\upshape\endtokencycle
```



Implicit spaces work, too

```
\resettokcycle
\Characterdirective{\addcytoks{\underdot{#1}}}
\def\:{\let\mysptoken= } \: %
\catcode`\_=10 %
\catcode`\ =12 %

\tokencyclexpress
\fbox{a\mysptoken{}c d} b_g\itshape f\upshape
\endtokencyclexpress
```



1.6.6 Changes to catcode 0

Cat-0 changes are not a hindrance to tokcycle

```
\let\littlet=t
\catcode`\! 0 !catcode`!\ 12
!Characterdirective{!ifimplicittok
  !addcytoks{!fbox{#1}}!else!ifcatSIX
  !addcytoks{!fbox{!fbox{#1}}}
  !else!addcytoks{#1}!fi!fi}
!tokencyclexpress Here, {!scshape!bgroup
  on !today!itshape{} we are !egroup
  !littlet es!littlet ing} cat-0
  changes{!bgroup}!egroup
!endtokencyclexpress!medskip

!detokenize!expandafter{!the!cytoks}
```

Here, on February 4, 2020 *we are* Testing cat-0 changes

Here, {\scshape \bgroup on \today \itshape {} we are \egroup \fbox {\littlet }es\fbox {\littlet }ing} cat-0 changes{\bgroup }\egroup