Package tokcycle (v1.11)

Steven B Segletes <steven.b.segletes.civ@mail.mil>
contributor: Christian Tellechea[1]
February 4, 2020

The tokcycle package helps one to build tools to process tokens from an input stream. If a macro to process an arbitrary single (non-macro, non-space) token can be built, then tokcycle can provide a wrapper for cycling through an input stream (including macros, spaces, and groups) on a token-by-token basis, using the provided macro on each successive character.

tokcycle characterizes each successive token in the input stream as a *Character*, a *Group*, a *Macro*, or a *Space*. Each of these token categories are processed with a unique directive, to bring about the desired effect of the token cycle. *If*-condition flags are provided to identify active, implicit, and catcode-6 tokens as they are digested. The package provides a number of options for handling groups.

Contents

[1] I am extremely grateful to Christian <unbonpetit@netc.fr> for his assistance in the development of this package. The \addcytoks macro was provided by him. He gave constant reminders on what the parser should be able to achieve, thus motivating me to spend the extra time striving for a generality of application that would not come naturally to me. I value highly his collegiality and hold his expertise in the highest regard.

1 The tokcycle macros and environments

The purpose of the tokcycle package is to provide a tool to assist the user in developing token-processing macros and environments. The types of processing are limited only by the creativity of the user, but examples might include letter-case-operations, letter spacing, dimensional manipulation, simple-ciphering, {group} manipulation, macro removal, etc. In one sense, it can be thought of as a streaming editor that operates on LaTeX input streams.

The package can be loaded into both plain TeX, by way of the invocation `\input tokcycle.tex` as well as LaTeX, via `\usepackage{tokcycle}`. It provides a total of 6 macros/pseudo environments, based on three criteria:

- Two pseudo-environments with the phrase "`tokencycle`" in the name, and four macros containing the phrase "`tokcycle`". The pseudo-environments operate within a group and typeset their result upon completion. The macros operate within the document's current scope, but do not typeset the result automatically. In the case of both macros and pseudo-environments, the transformed result is available for later use, being stored in the package token register named `\cytoks`.

- Two macros and one pseudo-environment containing the phrase "`xpress`". Without the phrase, the macro/environment requires four *processing directives* to be *explicitly* specified, followed by the input stream. With the phrase present, only the input stream is to be provided. In the `xpress` case, the *processing directives* are to have been separately specified via external macro and/or are taken from the most recent tokcycle macro invocation (failing that, are taken from the package initialization).

- Two macros containing the phrase "`expanded`". When present, the input stream of the token cycle is subject to the new TeX primitive, `\expanded`, prior to processing by the tokcycle macro. Expansion of specific macros in the input stream can be inhibited in the input stream with the use of `\noexpand`. Note that there are no `expanded` environments in tokcycle, as tokcycle environments do not pre-tokenize their input stream.

The basic approach of all tokcycle macros/environments is to grab each successive token (or group) from the input stream, decide what category it is, and use the currently active processing directives to make any desired transformation of the token (or group). Generally, with rare exception, the processed tokens should be stored in the token register `\cytoks`, using the tools provided by the package. The cycle continues until the input stream is terminated.

As tokens/groups are read from the input stream, they are categorized according to four type classifications, and are subjected to the user-specified processing directive associated with that category of token/group. The tokcycle categories by which the input stream is dissected include *Character*, *Group*, *Macro*, and *Space*.

Catcode-0 tokens are directed to the *Macro* directive for processing. Catcode-10 tokens are directed to the *Space* directive. When an explicit catcode-1 token is

encountered in the tokcycle input stream, the contents of the entire group (sans the grouping) are directed to the *Group* directive for further processing, which may in turn, redirect the individual tokens to the other categories. The handling options of implicit cat-1 and 2 tokens are described later in this document. Valid tokens that are neither catcode 0, 1, 2, nor 10, except where noted, are directed to the *Character* directive for processing.

The syntax of the non-`xpress` macros/environments is

```
\tokcycle or \expandedtokcycle
  {<Character processing directive>}
  {<Group-content processing directive>}
  {<Macro processing directive>}
  {<Space processing directive>}
  {<token input stream>}
```

or, alternately, for the pseudo-environment,

```
\tokencycle
  {<Character processing directive>}
  {<Group-content processing directive>}
  {<Macro processing directive>}
  {<Space processing directive>}<token input stream>\endtokencycle
```

For the `xpress` macros, the syntax is

```
\tokcyclexpress or \expandedtokcyclexpress
 {<token input stream>}
```

or, alternately, for the `xpress`-pseudo-environment,

```
\tokencycleexpress<token input stream>\endtokencyclexpress
```

In addition to the above macros/environments, the means is provided to define new tokcycle environments:

```
\tokencycleenvironment\environment_name
  {<Character processing directive>}
  {<Group-content processing directive>}
  {<Macro processing directive>}
  {<Space processing directive>}
```

This will then permit simplified invocations of the form

```
\environment_name<token input stream>\endenvironment_name
```

1.1 Externally specified directives and directive resets

For use in `xpress` mode, the directives for the *C-G-M-S* categories may be externally pre-specified, respectively, via the four macros `\Characterdirective`, `\Groupdirective`, `\Macrodirective`, and `\Spacedirective`, each taking an argument containing the particulars of the directive specification.

Each of these directives may be individually reset to the package default with the following argument-free invocations: `\resetCharacterdirective`,

\resetGroupdirective, \resetMacrodirective, or \resetSpacedirective. In addition, \resettokcycle is also provided, which not only resets all four directives collectively, but it also resets, to the default configuration, the manner in which explicit and implicit group tokens are processed. Finally, it resets the \aftertokcycle macro to empty.

The default directives at package outset and upon reset are

```
\Characterdirective{\addcytoks{#1}}
\Groupdirective{\processtoks{#1}}
\Macrodirective{\addcytoks{#1}}
\Spacedirective{\addcytoks{#1}}
\aftertokcycle{}
\stripgroupingfalse
\stripimplicitgroupingcase{0}
```

The interpretation of these directives will be explained in the remainder of this document. Let it suffice for now to say that the default directive settings pass through the input stream to the output stream, without change.

2 Commands in the tokcycle directives

The command-line token cycling tools provided in the package are listed in section 1. For each of those commands and/or pseudo-environments, the user must (explicitly or implicitly) detail a set of directives to specify the manner in which the *Character*, *Group*, *Macro*, and *Space* tokens found in the input stream are to be processed. The *C-G-M-S* processing directives consist of normal TEX/LATEX commands and user-defined macros to accomplish the desired effect. There are, however, several macros provided by the package to assist the user in this endeavor.

The recommended way to apply this package is to collect the tokcycle-transformed results of the input stream in a token register provided by the package, named \cytoks. Its contents can then be typeset via \the\cytoks. The macro for appending things to \cytoks, to be used in the package directives, is \addcytoks.

2.1 Adding tokens to the output stream: \addcytoks

The macro provided to append tokens to the \cytoks token register is named \addcytoks[]{}. Its mandatory argument consists of tokens denoting *what* you would like to append to the \cytoks register, while the optional argument denotes *how* you would like them appended (valid options include positive integers [<*n*>] and the letter [x]).

When the optional argument is omitted, the specified tokens are appended literally to the register (without expansion). An integer option, call it n, takes the the mandatory argument, and expands it n-times, and appends the result to \cytoks. The [x] option employs the \expanded primitive to maximally expand the argument before appending it to \cytoks.

The [x] option will prove useful when the *Character* directive involves a transformation that is fully expandable. Its use will allow the expanded result of the transformation to be placed in the token register, rather than the unexpanded transformation instructions.

2.1.1 #1

In the context of the *C*, *G*, *M*, and *S* processing directives, one may refer to #1 (e.g., in the argument to \addcytoks). TEX users know that the first parameter to a TEX macro is denoted as #1. The specification of all tokcycle processing directives is structured in such a way that "#1" may be directly employed to indicate the current token (or group) under consideration in the input stream.

2.2 *Group* directive: \ifstripgrouping, and \processtoks

The *Group* directive is unique, in that it is the only directive whose argument (#1) may consist of more than a single token. There are two issues to consider when handling the tokens comprising a group: do I retain or remove the grouping (cat-1,2 tokens)? Do I process the group's token content individually through the token cycle, or collectively as a unit?

Grouping in the output stream is determined by the externally set condition \ifstripgrouping. The package default is \stripgroupingfalse, such that any explicit grouping that appears in the input stream will be echoed in the output stream. The alternative, \stripgroupingtrue, is dangerous in the sense that it will strip the cat-1,2 grouping from the group's tokens, thereby affecting or even breaking code that utilizes such grouping for macro arguments. Modify \ifstripgrouping with care.

The issue of treating the tokens comprising the content of a group individually or collectively is determined by the choice of macro inside the *Group* directive. Within the *Group* directive, the argument #1 contains the collective tokens of the group (absent the outer cat-1,2 grouping). The default directive \processtoks{#1} will recommit the group's tokens individually to be processed through the token cycle. In contrast, the command \addcytoks{#1} in the *Group* directive would directly add the collective group of tokens to the \cytoks register, without being processed individually by tokcycle.

2.2.1 Implicit grouping tokens: \stripimplicitgroupingcase

Implicit grouping tokens (e.g., \bgroup & \egroup) can be handled in one of *three* separate ways. Therefore, rather than using an *if*-condition, the external declaration \stripimplicitgroupingcase{} is provided, which takes one of 3 integers as its argument (0, 1, or −1). The package-default case of "0" indicates that the implicit-group tokens will not be stripped, but rather echoed directly into the output stream. The case of "1" indicates that the implicit-group tokens will be stripped and not placed into the output stream (as with explicit grouping, this is a dangerous case and should be specified with care).

Finally, the special case of -1 indicates that the implicit-group tokens should instead be passed to the *Character* directive for further handling (note that the `\implicittoktrue` condition will be set[2]). Such a special treatment has limited application—for example, when the intent is to detokenize these tokens.

2.3 Escaping content from tokcycle processing

There are times you may wish to prevent tokens in the tokcycle input stream from being operated on by tokcycle. Rather, you just want the content passed through unchanged to the output; that is, with the intent to have multi-token content bypass the tokcycle directives altogether.

The method developed by the package is to enclose the escaped content in the input stream between a set of tokcycle escape characters, initially set to a vertical rule character found on the keyboard: "|". The main proviso is that the escaped content cannot straddle a group boundary (the complete group may, however, be escaped). The escape character can be changed with `\settcEscapechar{<`*escape-token*`>}`.

2.4 Flagged tokens

Certain token types are trapped and flagged via true/false *if*-conditions. These *if*-conditions can be examined within the appropriate directive (generally the *Character* directive), to direct the course of action within the directive.

2.4.1 Active characters

`\ifactivetok`: Active (cat-13) tokens that occur in the input stream result in the flag `\ifactivetok` being set `\activetoktrue`. Note that the expansion of the token's active `\def` occurs *after* tokcycle processing. With active `\let`'s, there is no text substitution; however, the assignment is already active at the time of tokcycle processing. The only exception to this rule is with pre-expanded input, `\expandedtokcycle[xpress]`. If an active token's substitution is governed by a `\def`, the text substitution will have occurred before reaching the token cycle.

`\ifactivetokunexpandable`: This flag is similar to `\ifactivetok`, in that a token must be active for this to be set true. However, in addition, it is only true if the active token is `\let` to a character or a primitive, neither of which can be expanded. Active characters assigned via `\def` or else `\let` to a macro will *not* qualify as `\activetokunexpandabletrue`.

`\ifactivechar`: This flag, rather than testing the token, tests the character code of the token, to see if it is set active. Generally, the token and its character code will be syncronized in their *activeness*. However, if a token is tokenized

[2] as well as the internal condition `\tc@implicitgrptrue`

when active, but the corresponding character code is made non-active in the meantime, prior to the token reaching tokcycle processing, this flag will be set `\activecharfalse`. A similar discrepency will arise if a token is not active when tokenized, but the character code is made active in the interim, prior to tokcycle processing.

2.4.2 Implicit tokens: `\ifimplicittok`

Implicit tokens, those assigned via `\let` to characters[3], are flagged with a `true` setting for `\ifimplicittok`. Generally, implicit tokens will be directed to the *Character* directive for processing. There are, however, two exceptions: i) implicit grouping tokens (e.g., `\bgroup` and `\egroup`) will not appear in any directive unless the `\stripimplicitgroupingcase` has been set to -1; and ii) implicit space tokens (e.g., `\@sptoken`) will be processed by the *Space* directive.

2.4.3 Parameter (cat-6) tokens (e.g., #): `\ifcatSIX`

Typically, category-code 6 tokens (like #) are used to designate a parameter (e.g., `#1`–`#9`). 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 trap and convert cat-6 tokens into something of the user's choosing.

As to the default nature of this conversion (if no special actions are taken), explicit cat-6 characters are converted into the identical character with category code of 12. On the other hand, implicit cat-6 macros (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.

2.4.4 Parameters (`#1`–`#9`): `\whennotprocessingparameter`

While, generally, one would not intend to use parameters in the tokcycle input stream, the package provides, not a flag, but a macro to allow it. The macro is to be used *within* the *Character* directive with the syntax:

`\whennotprocessingparameter#1{<`*non-parameter-processing-directive*`>}`

Here, the `#1` doesn't refer to `#1` as it appears in the input stream, but to the sole parameter of the *Character* directive, referring to the current token being processed. With this syntax, when the token under consideration is a parameter (e.g., `#1`–`#9`), that parameter is added to the `\cytoks` register. If the token under consideration is not a parameter, the code in the final argument is executed.

[3]Some clarification may be needed. Control sequences and active characters that are `\let` to something other than a cat-0 control sequence will be flagged as implicit. If implicit, a token will be processed through the *Character* directive (exceptions noted). On the other hand, if a control sequence or active character is `\let` to a cat-0 control sequence, it will be directed to the *Macro* directive for processing, without the implicit flag.

2.5 Misc: general *if*-condition tools

TEX comes equipped with a variety of `\if...` condition primitives. When dealing with macros for which the order of expansion is important, the `\else` and `\fi` can sometimes get in the way of proper expansion and execution. These four restructured *if* macros came in handy writing the package, and may be of use in creating your directives, without `\else` and `\fi` getting in the way:

`\tctestifcon{<`*TeX-if-condition*`>}{<`*true-code*`>}{<`*false-code*`>}`
`\tctestifx{<|`*ifx-comparison-toks*`>}{<`*true-code*`>}{<`*false-code*`>}`
`\tctestifnum{<|`*ifnum-condition*`>}{<`*true-code*`>}{<`*false-code*`>}`
`\tctestifcatnx<|`*ifcat*`|`*noexpand-comparison-toks*`>{<`*true-code*`>}{<`*false-code*`>}`

2.6 tokcycle macro close-out: `\aftertokcycle`

The tokcycle macros, upon completion, do nothing. Unlike `\tokencycle` environments, they don't even output `\the\cytoks` token register. A command has been provided, `\aftertokcycle`, which takes an argument to denote the actions to be executed following completion of *all* subsequent tokcycle invocations. It might be as simple as `\aftertokcycle{\the\cytoks}`, to have the output stream automatically typeset.

The meaning of `\aftertokcycle` can be reset with `\aftertokcycle{}`, but is also reset as part of `\resettokcycle`. Unlike macros, the tokcycle environments are unaffected by `\aftertokcycle`, as they actually set it locally (within their scope) to accomplish their own purposes.

2.7 Accommodating catcode-1,2 changes: `\settcGrouping`

In order to avoid making the tokcycle parser overly complex, requiring multiple passes of the input stream, the package defaults to using catcode-1,2 braces `{ }` to bring about grouping in the output stream, regardless of what the actual cat-1,2 tokens are in the input stream. As long as their sole purpose in the token cycle is for grouping and scoping, this arrangement will produce the expected output.

However, if the actual character-code of these tokens is important to the result (e.g., when detokenized), there is one other option. The package allows the external specification of which cat-1,2 tokens should be used in the tokcycle output stream. The syntax is `\settcGrouping{{#1}}`, to use the standard braces for this purpose (default). If angle-brackets `< >` were to be the *new* grouping tokens, then, after their catcodes were changed, and `\bgroup` and `\egroup` were reassigned, one would invoke `\settcGrouping<<#1>>`. These will then be the grouping tokens in the tokcycle output stream until set to something else.

3 Usage Examples

See the adjunct file, `tokcycle-examples.pdf`, for an array of tokcycle examples.

4 Summary of known package limitations

The goal of this package is not to build the perfect token-stream parser. It is, rather, to provide the means for users to build useful token-processing tools for their TEX/LATEX documents.

What follows are the known limitations of the package. which arise, in part, from the single-pass parsing algorithm embedded in the package. Surely, there are more cases associated with arcane catcode-changing syntax that are not accounted for; I encourage you to bring them to my attention. If I can't fix them, I can at least disclaim and declaim them.

- One must inform the package (via `\settcGrouping`) of changes to the cat-1,2 tokens *if* there is a need to detokenize the output with the specified bracing group; however, grouping will still be handled properly (i.e., cat-1,2 tokens will be detected), even if the package is not notified. See section 2.7.
- Should one need to keep track of the *names* of implicit tokens, then only one named implicit cat-6 token (e.g., `\let\svhash#`) may appear in the input stream (though it can appear multiple times). There is no limit on explicit cat-6 toks. This limitation occurs because implicit cat-6 tokens are converted into the fixed-name implicit macro `\implicitsixtok` which contains the `\string` of the most recently used implicit cat-6 token name. In any event, all cat-6 tokens are trapped and flag `true` the `\ifcatSIX` condition.

Acknowledgments

In addition to Christian Tellechea, a contributor to this package, the author would like to thank Dr. David Carlisle for his assistance in understanding some of the nuances of token registers. Likewise, his explanation about how a space token is defined in TEX (see `https://tex.stackexchange.com/questions/64197/pgfparser-module-and-blank-spaces/64200#64200`) proved to be useful here. The `tex.stackexchange` site provides a wonderful opportunity to interact with the leading developers and practitioners of TEX and LATEX.

Source Code

`tokcycle.sty`

```
\input tokcycle.tex
\ProvidesPackage\tcname[\tcdate\space V\tcver\space Cycle through and transform
  a stream of tokens]
\endinput
```

tokcycle.tex

```
\def\tcname                     {tokcycle}
\def\tcver                        {1.11}
%
\def\tcdate                    {2020/02/04}
%
% Author     : Steven B Segletes, Christian Tellechea (contributor)
% Maintainer : Steven B Segletes
% License    : Released under the LaTeX Project Public License v1.3c
%              or later, see http://www.latex-project.org/lppl.txt
% Files      : 1) tokcycle.tex
%              2) tokcycle.sty
%              3) tokcycle-doc.tex
%              4) tokcycle-doc.pdf
%              5) tokcycle-examples.tex
%              6) tokcycle-examples.pdf
%              7) README
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MACRO FORM
\long\def\tokcycle#1#2#3#4#5{\tokcycraw{#1}{#2}{#3}{#4}#5\endtokcycraw}
% \expanded-ARGUMENT MACRO FORM
\long\def\expandedtokcycle#1#2#3#4#5{\cytoks{\tokcycraw{#1}{#2}{#3}{#4}}%
  \expandafter\the\expandafter\cytoks\expanded{#5}\endtokcycraw}
% ENVIRONMENT FORM
\long\def\tokencycle#1#2#3#4{\begingroup\let\endtokencycle\endtokcycraw
  \aftertokcycle{\the\cytoks\expandafter\endgroup\expandafter\tcenvscope
  \expandafter{\the\cytoks}}\tokcycraw{#1}{#2}{#3}{#4}}
% XPRESS-INTERFACE MACRO FORM
\long\def\tokcyclexpress#1{\tokcycrawxpress#1\endtokcycraw}
% XPRESS-INTERFACE \expanded-ARGUMENT MACRO FORM
\long\def\expandedtokcyclexpress#1{%
  \expandafter\tokcycrawxpress\expanded{#1}\endtokcycraw}
% XPRESS-INTERFACE ENVIRONMENT FORM
\def\tokencyclexpress{\begingroup\let\endtokencyclexpress\endtokcycraw
  \aftertokcycle{\the\cytoks\expandafter\endgroup\expandafter\tcenvscope
  \expandafter{\the\cytoks}}\tokcycrawxpress}
% INITIALIZATION & INTERNAL TOOLS
\def\tcenvscope{\cytoks}% CAN SET TO \global\cytoks TO OVERCOME SCOPE LIMITS
\edef\restorecatcode{\catcode\number`\@=\number\catcode`\@\relax}
\catcode`\@11
\newif\iftc@implicitgrp
\newif\if@argnext
\newtoks\tc@tok
\newcount\tc@depth
\def\tc@gobble#1{}
\def\tc@deftok#1#2{\let#1= #2\empty}
\tc@deftok\tc@sptoken{ }
\expandafter\def\expandafter\tc@absorbSpace\space{}
\def\tc@ifempty#1{\tc@testxifx{\expandafter\relax\detokenize{#1}\relax}}
\def\tc@defx#1#2{\tc@earg{\def\expandafter#1}{#2}}
\long\def\tc@earg#1#2{\expandafter#1\expandafter{#2}}
\long\def\tc@xarg#1#2{\tc@earg#1{\expanded{#2}}}
\long\def\tc@exfirst#1#2{#1}
\long\def\tc@exsecond#1#2{#2}
\long\def\tc@testxifx{\tc@earg\tctestifx}
\long\def\test@ifmacro#1{\tctestifcatnx#1\relax}
\def\tc@addtoks#1#2{\toks#1\expandafter{\the\toks#1#2}}
```

```
\def\addtc@depth{\advance\tc@depth 1}
\def\subtc@depth{\tc@depth=\numexpr\tc@depth-1\relax}
\def\tc@resetifs{\activetokfalse\implicittokfalse\tc@implicitgrpfalse
  \catSIXfalse\activecharfalse\activetokunexpandablefalse}
\long\def\count@stringtoks#1{\tc@earg\count@toks{\string#1}}
\long\def\count@toks#1{\the\numexpr-1\count@@toks#1.\tc@endcnt}
\long\def\count@@toks#1#2\tc@endcnt{+1\tc@ifempty{#2}{\relax}{\count@@toks#2\tc@endcnt}}
\def\sv@hash{##}
% EXTERNAL TOOLS
\long\def\tctestifcon#1{#1\expandafter\tc@exfirst\else\expandafter\tc@exsecond\fi}
\long\def\tctestifcatnx#1#2{\tctestifcon{\ifcat\noexpand#1\noexpand#2}}
\long\def\tctestifx#1{\tctestifcon{\ifx#1}}
\long\def\tctestifnum#1{\tctestifcon{\ifnum#1\relax}}
\newif\ifstripgrouping
\def\stripimplicitgroupingcase#1{\edef\@implicitgroupingcase{\the\numexpr1-#1}}
\newif\ifcatSIX
\newif\ifimplicittok
\newif\ifactivetok
\newif\ifactivechar
\newif\ifactivetokunexpandable
\newtoks\cytoks
\long\def\tokcycleenvironment#1#2#3#4#5{\expandafter\def\expandafter#1%
  \expandafter{\expandafter\let\csname end\expandafter\tc@gobble
  \string#1\endcsname\endtokcycraw\tokencycle{#2}{#3}{#4}{#5}}}
\long\def\processtoks#1{\addtc@depth\@tokcycle#1\endtokcycraw }
\def\whennotprocessingparameter#1#2{\tctestifcon\if@argnext{\@argnextfalse\cytoks
  \expandafter{\the\cytoks###1}}{\tctestifcon\ifcatSIX{\@argnexttrue}{#2}}}
% ESSENTIAL METHOD: STREAMING MACRO WITH TERMINATOR:
% \tokcycraw{<Char>}{<Group>}{<Macro>}{<Space>}<input-stream>\endtokcycraw
\long\def\tokcycraw#1#2#3#4{\def\@chrT##1{#1}\long\def\@grpT##1{#2}%
  \long\def\@macT##1{#3}\def\@spcT##1{#4}\tokcycrawxpress}
% ENTRY POINT FOR XPRESS METHOD WITHOUT EXPLICIT ARGUMENTS
\def\tokcycrawxpress{\cytoks{}\tc@depth=1\relax\@tokcycle}
% CODE TO EXECUTE AT COMPLETION
\long\def\aftertokcycle#1{\def\@aftertokcycle{#1}}
\def\endtokcycraw{\subtc@depth\tctestifnum{\tc@depth=0}{\@aftertokcycle}{}}
% LOOP ENTRY POINT
\def\@tokcycle{\tc@resetifs\futurelet\tc@next\detect@CANTabsorb}
\def\detect@CANTabsorb{\tctestifx{\tc@next\tc@sptoken}{\stringify\@@@@@spcT}%
  {\tctestifx{\tc@next\bgroup}{\stringify\@@@@grpT}{\can@absorb}}}
% NON cat1,10 TOKENS
\long\def\can@absorb#1{\tc@tok{#1}\trapcatSIX{#1}\expandafter\can@absorb@
  \the\tc@tok}
\long\def\can@absorb@#1{\tctestifnum{\count@stringtoks{#1}>1}%
  {\tctestifx{\endtokcycraw#1}{#1}{\backslashcmds#1\@tokcycle}}%
  {\trapactives#1\tc@trapescape#1{\tc@escapecytoks}{\can@absorb@@#1}}}
\long\def\can@absorb@@#1{\let\@tmp=#1\test@ifmacro\@tmp{\implicittokfalse
  \@macT#1}{\trapimplicitegrp#1\implicitgrpfork#1}\@tokcycle}
%CONVERT NEXT (SPACE OR BEGIN-GROUP) TOKEN TO STRING
\def\stringify#1{\expandafter#1\string}% #1 WILL BE \@@@@@spcT or \@@@@grpT
%SPACE DECODE
\def\@@@@@spcT{\futurelet\tc@str\@@@@spcT}
\def\@@@@spcT{\tctestifx{\tc@str\tc@sptoken}%
 {\def\@tmp{\@@spcT{ }}\expandafter\@tmp\tc@absorbSpace}% EXPLICIT SPACE
 {\implicittoktrue\expandafter\@@@spcT\tc@gobble}}% IMPLICIT SPACE
\def\@@@spcT{\csmk{\expandafter\@@spcT\thecs}}
\def\@@spcT#1{\@spcT{#1}\@tokcycle}
```

```
% GROUP DECODE
\def\@@@@grpT{\futurelet\tc@str\@@@grpT}
\def\@@@grpT#1{\tctestifnum{\number\catcode`#1=1}%
  {\expandafter\@@grpT\expandafter{\iffalse}\fi}% {
  {\implicittoktrue\tc@implicitgrptrue%
    \tctestifnum{`#1=92}% WORKS EVEN IF CAT-0 HAS CHANGED
    {\csmk{\expandafter\backslashcmds\thecs\@tokcycle}}% \bgroup
    {\begingroup\catcode`#1=\active \xdef\@tmp{\scantokens{#1\noexpand}}\endgroup
      \expandafter\implicitgrpfork\@tmp\@tokcycle}% ACTIVE CHAR \bgroup
  }}
\def\@@grpT#1{\tctestifcon\ifstripgrouping{\@grpT{#1}}%
  {\groupedcytoks{\@grpT{#1}}}\@tokcycle}
% \ COMMANDS (MACROS AND IMPLICITS)
\long\def\backslashcmds#1{%
  \test@ifmacro#1{\tctestifcon\ifcatSIX{\implicittoktrue\@chrT#1}{\@macT#1}}%
    {\implicittoktrue\trapimplicitegrp#1\implicitgrpfork#1}}
% FORK BASED ON IMPLICIT GROUP TREATMENT
\def\implicitgrpfork#1{\tctestifcon{\iftc@implicitgrp}{\ifcase
  \@implicitgroupingcase\or\addcytoks{#1}\or\@chrT{#1}\fi}{\@chrT#1}}
% SET UP ESCAPE MECHANISM
\def\settcEscapechar#1{\let\@tcEscapeptr#1%
  \def\tc@escapecytoks##1#1{\addcytoks{##1}\@tokcycle}}
\def\tc@trapescape#1{\tctestifx{\@tcEscapeptr#1}}
% TRAP CAT-6
\long\def\trapcatSIX#1{\tctestifcatnx#1\relax{}{\trapcatSIXb#1}}
\def\trapcatSIXb#1{\tc@earg\tctestifcatnx\sv@hash#1{\catSIXtrue\trapcatSIXc#1}{}}
\def\trapcatSIXc#1{\tctestifnum{\count@stringtoks{#1}>1}{\tc@defx\six@str{\string#1}%
  \global\let\implicitsixtok\six@str\tc@tok{\implicitsixtok}}%
  {\tc@tok\expandafter{\string#1}\tctestifnum{\number\catcode`#1=6}%
  {}{\activetoktrue\implicittoktrue}}}
% DIRECTIVES FOR HANDLING GROUPS RECURSIVELY; DEFINE tokcycle GROUPING CHARS
\def\@defgroupedcytoks#1{\long\def\groupedcytoks##1{%
  \begingroup\cytoks{}##1\expandafter\endgroup\expandafter
  \addcytoks\expandafter{\expandafter#1}}}
\def\settcGrouping#1{\def\@tmp##1{#1}\tc@defx\@@tmp{\@tmp{\the\cytoks}}%
  \tc@earg\@defgroupedcytoks{\@@tmp}}
% FAUX TOKENIZATION OF COMMAND NAME (WHEN cat0 TOKEN HAS BEEN MADE cat12)
\def\csmk#1{\def\csaftermk{#1}\toks0{}\@csmkA}
\def\@csmkA{\futurelet\@tmp\@csmkB}
\def\@csmkB{\tctestifx{\@tmp\tc@sptoken}%
  {\toks0{ }\expandafter\@csmkF\tc@absorbSpace}{\@csmkCA}}
\def\@csmkCA#1{\tc@addtoks0{#1}\tctestifnum{\number\catcode`#1=11}%
  {\futurelet\@tmp\@csmkD}{\@csmkF}}
\def\@csmkC#1{\tctestifnum{\number\catcode`#1=11}
  {\tc@addtoks0{#1}\futurelet\@tmp\@csmkD}{\@csmkE#1}}
\def\@csmkD{\tctestifcatnx 0\@tmp\@csmkC\@csmkE}
\def\@csmkE{\tctestifx{\@tmp\tc@sptoken}%
  {\expandafter\@csmkF\tc@absorbSpace}{\@csmkF}}
\def\@csmkF{\tc@defx\thecs{\csname\the\toks0\endcsname}\csaftermk}
% TRAP IMPLICIT END GROUP TOK (e.g., \egroup); SET \iftc@implicitgrp
\def\trapimplicitegrp#1{\tctestifx{#1\egroup}{%
  \implicittoktrue\tc@implicitgrptrue}{}}
% TRAP ACTIVE TOK
\def\trapactives#1{\trapactivechar{#1}\trapactivetok{#1}}
\def\trapactivechar#1{\tctestifnum{\number\catcode`#1=13}{\activechartrue}{}}
\def\trapactivetok#1{\tctestifcatnx~#1{\activetoktrue}{\trapactivetokunexpandable#1}}
%% WILL ALSO TRAP ACTIVE \let TO PRIMITIVES AS IMPLICIT; UNDO LATER IN \can@absorb@@
```

```
\def\trapactivetokunexpandable#1{\tctestifcon{\expandafter\if
  \detokenize{#1}#1}{}{\activetoktrue\activetokunexpandabletrue\implicittoktrue}}
% EXPRESS-INTERFACE - ALLOWS TO EXTERNALLY DEFINE DIRECTIVES
\def\Characterdirective{\def\@chrT##1}
\def\Groupdirective{\long\def\@grpT##1}
\def\Macrodirective{\long\def\@macT##1}
\def\Spacedirective{\def\@spcT##1}
% EXPRESS-INTERFACE - DEFAULT DIRECTIVES
\def\resetCharacterdirective{\Characterdirective{\addcytoks{##1}}}
\def\resetGroupdirective{\Groupdirective{\processtoks{##1}}}
\def\resetMacrodirective{\Macrodirective{\addcytoks{##1}}}
\def\resetSpacedirective{\Spacedirective{\addcytoks{##1}}}
\def\resettokcycle{\resetCharacterdirective\resetGroupdirective
  \resetMacrodirective\resetSpacedirective\aftertokcycle{}%
  \stripgroupingfalse\stripimplicitgroupingcase{0}}
% SUPPORT MACROS FOR TOKENIZED OUTPUT: \addcytoks[<expansion level>]{<arg>}
% (CONTRIBUTED BY CHRISTIAN TELLECHEA)
\def\addcytoks{\futurelet\nxttok\addcytoks@A}
\long\def\tc@addtotoks#1{\cytoks\expandafter{\the\cytoks#1}}
\def\addcytoks@A{\tctestifx{[\nxttok}\addcytoks@B\tc@addtotoks}
\long\def\addcytoks@B[#1]#2{\tc@ifempty{#1}\tc@addtotoks
  {\tctestifx{x#1}{\tc@xarg\tc@addtotoks}{\addcytoks@C{#1}}}{#2}}
\def\addcytoks@C#1{\tctestifnum{#1>0}{\tc@earg\addcytoks@C
  {\the\numexpr#1-1\expandafter}\expandafter}\tc@addtotoks}
% SET INITIAL PARAMETERS
\settcGrouping{{#1}}%          E.G. <<#1>> IF cat-1,2 SET TO < AND >
\settcEscapechar{|}%           BYPASS TOKCYCLE PROCESSING BETWEEN |...|
\resettokcycle%                WHICH ALSO CONTAINS THE FOLLOWING 3 RESETS:
%  \stripimplicitgroupingcase{0}% DEFAULT RETAIN UNALTERED \b/e-groups
%  \stripgroupingfalse%           DEFAULT RETAIN UNALTERED {} GROUPING
%  \aftertokcycle{}% NO DEFAULT CODE EXECUTED AFTER EACH TOKCYCLE INVOCATION
\restorecatcode
\endinput

EDIT HISTORY
v1.0  2019/8/21
      - Initial release

v1.1  2019/9/27
      - Introduced \ifactivechar, \ifactivetokunexpandable
      - Tightened up consistent definition of implicit (to exclude primitives)
      - Rewrote active token trapping logic, to differentiate between active
        token vs. active character code, in the event that an earlier tokenized
        token no longer shares the current characteristics of the character code
      - Added ability to handle active-implicit grouping tokens
      - Added ability to handle active-implicit cat-6 tokens

v1.11 2020/02/04
      - Fixed bug in \can@absorb@@ macro, which prevented the proper absorption/
        handling of the = token.
```