Citation Strings (`parse.dtx`)

Charles Duan

May 3, 2023

To specify the information needed to cite references in a document, this package provides a compact but flexible input syntax intended to capture the many types of information that may be included in a citation. This section first describes a data model for the information that goes into a citation, and then explains the syntax for input text that this package uses to receive that information. The input text syntax is used for all citation commands that this package accepts (`\sentence`, `\clause`, `\inline`, etc.).

1 Data Model

A *citation string* is an ordered list of one or more *citation items*.[1] Each citation item includes the following components:

- A signal (see `signals.dtx`).
- A reference name.
- A volume number.
- A pin cite (see `pages.dtx`).
- An "optional argument," used by some reference types to fine-tune the citation display.
- A list of parentheticals (see `parens.dtx`).

All of the components are optional other than the reference name. The syntax and meanings of signals, pin cites, and parentheticals are treated in other documentation sections; the reference name, volume number, and optional argument are freeform text that is used as-is with no processing.

[1] Multiple citation items are permitted even for citation command types like `\inline` that ought to accept only one item.

2 Input Syntax

Citation strings are given to the package using the following syntax:

⟨*citation-string*⟩ := ⟨*citation-item*⟩ (`;` ⟨*citation-item*⟩)*
⟨*citation-item*⟩ := [⟨*signal*⟩]
[⟨*volume*⟩]
⟨*reference-name*⟩
[`[` ⟨*opt-argument*⟩ `]`]
[`at` ⟨*pincite*⟩]
(`(` ⟨*parenthetical*⟩ `)`)*

More concretely, citation strings are one or more semicolon-separated items. The following cites three references:

```
ref-a; ref-b; ref-c
```

A complete citation item with every component might look like this:

```
see, e.g., 5 treatise[opt] at 35-60 (citing
references) (outdated)
```

This would cite volume 5, pages 35–60 of a reference named `treatise` with the signal `see, e.g.`, an optional argument `opt`, and two parentheticals.

More commonly, only a few components of a citation item will be included. An example citation string in ordinary use might look as follows:

```
statute at S 167; see also supporting-case at 189;
another-case at 465; cf. journal-article at 754
(discussing related applications of the statute); see
generally 7 treatise at S 56
```

Note that if the same reference is cited multiple times in a row, there is no facility for writing *id.* or the like in the input syntax. The reference must be fully identified and pin-cited on each use, and the package will automatically compute whether *id.* should be inserted as described in `state.dtx`. While this may seem like unnecessary work, there is a major advantage: The citation string input is fully independent of where in the document the citation string appears. As a result, writers can freely move blocks of text around a document without having to rewrite input citation strings. If *id.* or other shortcuts had been used in the document, then they likely would become unintelligible upon a document reorganization.

3 Implementation

Although the syntax is fairly straightforward, parsing that syntax in TEX is difficult. Citations are parsed with a state machine that consumes tokens from a citation string. The objective is to turn a citation string into this data structure:

```
{%
  \hi@pse@cite{%
    \hi@pse@svr{[sig vol ref]}%
    \hi@pse@opt{[opt]}%
    \hi@pse@page{[page]}%
    \hi@pse@paren{[paren]}%
    \hi@pse@paren{[paren]}%
    ...
  }%
  ...
}
```

To that end, we first define three components of infrastructure. First, we create a mini-parser for parsing the signal, volume, and reference name from a text string that appears after a signal in a citation. (This mini-parser is not used by the state machine, but is used internally by the formatting routine.) Second, we build a tokenizer that extracts components from the citation string. Third, we create an accumulator for storing the citation data structure as it is built.

3.1 Signal, Volume, and Reference

Separate the signal, volume, and reference name from a string of words. #1 is the text to split; #2 a callback which will receive, as arguments, the signal, volume, and reference.

```
\def\hi@pse@sigvolref#1#2{%
    \expandafter\find@try\expandafter\find@start\expandafter{%
        \hi@pse@findlist
    }{#1}{\hi@pse@sigvolref@withsig{}{#1}}{#2}%
}
```

Separates out the volume from the reference name. #1 is the signal, #2 the remaining text, #3 the callback.

```
\def\hi@pse@sigvolref@withsig#1#2#3{%
    \find@in{ }{#2}{%
        #3{#1}%
        % The two remaining arguments from \find@in will go to the callback
    }{%
        #3{#1}{}{#2}% No volume found; run the callback with blank and #2
    }%
}
```

3.2 Tokenizer

This is for converting a citation string into relevant tokens for the state machine.

Reads a word from the input and categorizes it into the following:

- Words/groups
- Space
- Semicolon
- Other punctuation
- EOF

#1 is the text to read; five arguments that follow are callbacks corresponding to the above list; the appropriate one is chosen. Each is given two arguments: the found text and the remaining text to parse. (The EOF callback receives nothing.)

```
\def\hi@pse@read#1{%
    %\hi@pse@debug{Reading #1}%
    \find@word{#1}%
        \hi@pse@read@char\hi@pse@read@grp\hi@pse@read@do@wd\hi@pse@read@eof
}
\def\hi@pse@read@char#1#2{%
    %\hi@pse@debug{Read char: #1}%
    \find@eq{;}{#1}{\hi@pse@read@do@sc}{%
```

```
        \find@eq{ }{#1}{\hi@pse@read@do@spc}{\hi@pse@read@do@punct}%
    }{#1}{#2}%
}
\def\hi@pse@read@grp#1#2{%
    %\hi@pse@debug{Read group: #1}%
    \hi@pse@read@do@wd{{#1}}{#2}%
}%
\def\hi@pse@read@eof#1#2#3#4#5{%
    %\hi@pse@debug{Read EOF}%
    #5%
}
```

In all of these, #1 is the matching text, #2 is the remaining text to be parsed, and #3-#7 are the callbacks described above.

```
\def\hi@pse@read@do@wd   #1#2#3#4#5#6#7{%
    %\hi@pse@debug{Read word: #1}%
    #3{#1}{#2}%
}
\def\hi@pse@read@do@spc  #1#2#3#4#5#6#7{%
    %\hi@pse@debug{Read space}%
    #4{#1}{#2}%
}
\def\hi@pse@read@do@sc   #1#2#3#4#5#6#7{%
    %\hi@pse@debug{Read semicolon}%
    #5{#1}{#2}%
}
\def\hi@pse@read@do@punct#1#2#3#4#5#6#7{%
    %\hi@pse@debug{Read punct: #1}%
    #6{#1}{#2}%
}
```

3.3 Accumulator

The purpose of this set of routines is to build up information as it is parsed from a citation string, to arrange them into the above structure. To do so, we define three accumulators:

- all: Holds the final list of citation structures
- cite: Holds the current citation structure being built
- work: Holds the working text being accumulated

The following command resets all the accumulators.

```
\def\hi@pse@acc@reset{%
    \let\hi@pse@acc@all\@empty
    \let\hi@pse@acc@cite\@empty
    \let\hi@pse@acc@work\@empty
}
```

First, helper functions. This macro saves save the work accumulator to the cite accumulator. #1 is the macro prefix (it must be a single-token macro).

```
\def\hi@pse@acc@savework#1{%
    \@expand{\hi@pse@acc@saveparam{#1}}\hi@pse@acc@work i%
}
%
% Save the given text under a macro in the citation structure directly. \#1 is
% the parameter; \#2 the text to be saved. It also clears out the work
% accumulator.
%
%    \begin{macrocode}
\def\hi@pse@acc@saveparam#1#2{%
    \addto@macro\hi@pse@acc@cite{#1{#2}}%
    \let\hi@pse@acc@work\@empty
}
```

This macro saves the working citation to the citation list, and clears both the work and cite accumulators.

```
\def\hi@pse@acc@savecite{%
    \expandafter\addto@macro\expandafter\hi@pse@acc@all\expandafter{%
        \expandafter\hi@pse@cite\expandafter{%
            \hi@pse@acc@cite
        }%
    }%
    \let\hi@pse@acc@work\@empty
    \let\hi@pse@acc@cite\@empty
}
```

At each state transition, one of a few things can happen, which are defined by the macros below.

Add to the work accumulator and then transition to the next state. #1 is the next state; #2 is the text to be added; #3 is the remaining text to parse.

```
\def\hi@pse@acc@add#1#2#3{\addto@macro\hi@pse@acc@work{#2}#1{#3}}
```

Add a space to the working accumulator. (This is used by states that consume a space that may be part of the citation text or may just be an ignorable separator.)

```
\def\hi@pse@acc@addspc{\addto@macro\hi@pse@acc@work{ }}
```

Like add, but ignores leading space.

```
\def\hi@pse@acc@addnonblank#1#2#3{%
    \ifx\hi@pse@acc@work\@empty
        \find@eq{ }{#2}{}{\addto@macro\hi@pse@acc@work{#2}}%
    \else
        \addto@macro\hi@pse@acc@work{#2}%
    \fi
    #1{#3}%
}
```

Ignore the current text and jump to the next state. #1 is the state, #2 the text to ignore, #3 the rest of the text to parse.

```
\def\hi@pse@acc@ignore#1#2#3{#1{#3}}%
```

Set the work accumulator to the found text, and transition to the next state. (This effectively discards whatever was in the work accumulator.)

```
\def\hi@pse@acc@addnext#1#2#3{\def\hi@pse@acc@work{#2}#1{#3}}
```

Place the work accumulator into the cite accumulator, prefixed by the given macro name. Ignore the found text and transition to the next state. #1 is the macro name for the working accumulator's text, #2 is the next state, #3 is the found text (to be ignored), and #4 is the remaining text.

```
\def\hi@pse@acc@setparam#1#2#3#4{%
    \hi@pse@acc@savework{#1}%
    #2{#4}%
}
```

Save the cite accumulator to the all accumulator, ignore the found text, and go to the `\hi@pse@state@nextcite` state. #1 is the found text and #2 the remaining text to parse.

```
\def\hi@pse@acc@setcite#1#2{%
    \hi@pse@acc@savecite
    \hi@pse@state@nextcite{#2}%
}
```

Does both setparam and setcite. #1 is the macro name, #2 the found text (to be ignored), #3 the rest of the text to parse.

```
\def\hi@pse@acc@setparamcite#1#2#3{%
    \hi@pse@acc@savework{#1}%
    \hi@pse@acc@setcite{#1}{#3}%
}
```

Handles an error during parsing, by printing an error message and setting the citation state to include an error.

```
\def\hi@pse@error#1#2{%
    \def\reserved@a{>>>#1<<<#2}%
    \PackageError\hi@pkgname{Citation parse error:
    \expandafter\strip@prefix\meaning\reserved@a
    }{Fix the erroneous citation}%
    \hi@acc@savecite
    \addto@macro\hi@pse@acc@all{\hi@pse@cite{\hi@pse@svr{???}}}%
}
```

3.4 State Machine

Parsing a citation is a matter of running a state machine as components of the citation are read and stored in accumulators. Having defined the tokenizer and the accumulator systems above, this portion of the program defines the states and their transitions.

This macro initializes the accumulators and starts the state machine. #1 is the citation text, and #2 the callback.

```
\def\hi@pse@parse#1#2{%
    %\hi@pse@debug{Starting parse: #1}%
    \hi@pse@acc@reset
    \hi@pse@state@nextcite{#1}%
    \@expand{#2}\hi@pse@acc@all i%
}
```

Each state accepts one argument, the portion of the citation remaining to be read. (All other state for the parser is maintained in the accumulators.) The state executes `\hi@pse@read` to extract a token, and then provides five callbacks in compliance with `\hi@pse@read`. Each callback other than the EOF one will take two arguments: the tokenized portion of the citation string read, and the remaining text. The callback should update any accumulators and then jump to the next appropriate state. For the EOF callback, each state except for nextcite will treat it as if a semicolon had been read (nextcite will do nothing, thereby ending parsing).

The following are the states:

At the beginning of a citation or after a semicolon. Wait for the first substantive token and go to svr.

```
\def\hi@pse@state@nextcite#1{%
    %\hi@pse@debug{State nextcite; reading #1}%
    \hi@pse@read{#1}%
        {\hi@pse@acc@addnext\hi@pse@state@svr}% word
        {\hi@pse@acc@ignore\hi@pse@state@nextcite}% space
        {\hi@pse@acc@ignore\hi@pse@state@nextcite}% semicolon
        {\hi@pse@acc@addnext\hi@pse@state@svr}% other punctuation
        {}% EOF--do nothing; all done
}
```

At the beginning of the substantive text of a citation, which should be the signal, volume, and reference name. Read tokens until reaching a square bracket, a parenthetical, or ” at ”.

```
\def\hi@pse@state@svr#1{%
    %\hi@pse@debug{State svr; reading #1}%
    \hi@pse@read{#1}%
        {\hi@pse@acc@add\hi@pse@state@svr}% word
        {\hi@pse@acc@ignore\hi@pse@state@svrspc}% space
        {\hi@pse@acc@setparamcite\hi@pse@svr}% ;
        {\hi@pse@state@svr@testpunct{}}% punct
        {\hi@pse@acc@setparamcite\hi@pse@svr{}{}}% eof
}
```

This state is reached if a space is found while reading the signal, volume, and reference name. (The space has not been added to the accumulator yet.) The special case is if the text to parse begins with ”at ” since that means we start reading a page number. Otherwise, we use `\hi@pse@read` as usual.

```
\def\hi@pse@state@svrspc#1{%
    %\hi@pse@debug{State svrspc; reading #1}%
    \find@start{at }{#1}{%
        % The trailing argument of the \find@start callback is the remaining
        % text to be parsed. Thus, \hi@pse@acc@setparam receives four arguments
        % as expected.
        \hi@pse@acc@setparam\hi@pse@svr\hi@pse@state@page{}%
    }{%
        \hi@pse@read{#1}%
            {\hi@pse@acc@addspc\hi@pse@acc@add\hi@pse@state@svr}% Word
            {\hi@pse@acc@ignore\hi@pse@state@svrspc}% space
            {\hi@pse@acc@setparamcite\hi@pse@svr}% ;
            {\hi@pse@state@svr@testpunct{ }}% punct
            {\hi@pse@acc@setparamcite\hi@pse@svr{}{}}% eof
    }%
}
```

In the svr or svrspc states, if we identify a punctuation mark, we have to test to see if it starts an option or parenthetical. #1 is either empty or a space depending on what preceded this call.

```
\def\hi@pse@state@svr@testpunct#1#2#3{%
    %\hi@pse@debug{svr@testpunct{#1}{#2}{#3}}%
    \find@try\find@eq{%
```

```
        {[}{\hi@pse@acc@setparam\hi@pse@svr\hi@pse@state@opt{#2}{#3}}%
        {(}{\hi@pse@acc@setparam\hi@pse@svr\hi@pse@state@paren{#2}{#3}}%
    }{#2}{\hi@pse@acc@add\hi@pse@state@svr{#1#2}{#3}}%
}
```

The state where a square bracket was read, indicating parsing of the optional argument. Anything other than a square bracket is made part of the optional argument.

```
\def\hi@pse@state@opt#1{%
    %\hi@pse@debug{State opt; reading #1}%
    \find@in{]}{#1}\hi@pse@state@opt@set{%
        \hi@pse@error{EOF}{ before option parsing finished}%
    }%
}
\def\hi@pse@state@opt@set#1#2{%
    \hi@pse@acc@saveparam\hi@pse@opt{#1}%
    \hi@pse@state@afteropt{#2}%
}
```

The state after the closing square bracket of an option was read. Anything is an error other than a semicolon, parenthetical, or page number signal "at".

```
\def\hi@pse@state@afteropt#1{%
    %\hi@pse@debug{State afteropt; reading #1}%
    \find@try\find@start{%
        % The trailing argument of the \find@start callback is the remaining
        % text to be parsed, thus passed to the next state. Accumulators were
        % cleared at state@opt so they are in the right status.
        {at }{\hi@pse@state@page}%
        { }{\hi@pse@state@afteropt}%
        {;}{\hi@pse@acc@setcite{}}% Extra arg needed for acc@setcite
        {(}{\hi@pse@state@paren}%
    }{#1}{%
        \ifstrempty{#1}{\hi@pse@acc@setcite{}{}}{%
            \hi@pse@error{Invalid text after optional arg}{#1}%
        }%
    }%
}
```

The state where we're reading a page number. We read until reaching a semicolon (next cite) or a space followed by a parenthesis.

```
\def\hi@pse@state@page#1{%
    %\hi@pse@debug{State page; reading #1}%
    \hi@pse@read{#1}%
        {\hi@pse@acc@add\hi@pse@state@page}% word
        {\hi@pse@acc@ignore\hi@pse@state@pagespc}% Space
        {\hi@pse@acc@setparamcite\hi@pse@page}% semicolon
        {\hi@pse@acc@add\hi@pse@state@page}%
        {\hi@pse@acc@setparamcite\hi@pse@page{}{}}%
}
```

The state where a space within a page number was read. If the next token is a parenthesis or semicolon, then go on to the next appropriate state. Otherwise, add the space back to the page number and continue parsing the page number.

```
\def\hi@pse@state@pagespc#1{%
    %\hi@pse@debug{State pagespc; reading #1}%
    \find@try\find@start{%
        {;}{\hi@pse@acc@setparamcite\hi@pse@page{}}%
        {(}{\hi@pse@acc@setparam\hi@pse@page\hi@pse@state@paren{}}%
    }{#1}{%
        \ifstrempty{#1}{\hi@pse@acc@setparamcite\hi@pse@page{}{}}{%
            \hi@pse@acc@addspc\hi@pse@state@page{#1}%
        }%
    }%
}
```

The state where the opening parenthesis of a parenthetical was read. The only tokens of interest are parentheses. We use a separate parsing system here, because we need to keep track of how many parentheses have been seen; this is done with a counter system of tokens.

```
\def\hi@pse@state@paren{\hi@pse@state@paren@count{i}{}}%
```

#1 is the set of counter tokens; #2 the accumulated parenthetical; #3 the text to be parsed.

```
\def\hi@pse@state@paren@count#1#2#3{%
    %\hi@pse@debug{State paren@count; reading #3}%
    \find@next{#3}%
        \hi@pse@state@paren@add
        \hi@pse@state@paren@addgrp
```

```
        \hi@pse@state@paren@test
        {\hi@pse@error{EOF}{ before parenthetical parsing finished}}%
    {#1}{#2}%
}
```

#1 is the found text, #2 the rest of the string to parse, #3 the counter tokens, and #4 the accumulated parenthetical.

```
\def\hi@pse@state@paren@add#1#2#3#4{%
    \hi@pse@state@paren@count{#3}{#4#1}{#2}%
}
\def\hi@pse@state@paren@addgrp#1#2#3#4{%
    \hi@pse@state@paren@count{#3}{#4{#1}}{#2}%
}
\def\hi@pse@state@paren@test#1#2#3#4{%
    \find@try\find@eq{%
        {(}{\hi@pse@state@paren@add{#1}{#2}{#3i}{#4}}%
        {)}{\hi@pse@state@paren@dec{#1}{#2}{#4}#3\@stop}%
    }{#1}{\hi@pse@state@paren@add{#1}{#2}{#3}{#4}}%
}
```

Handles a close parenthesis. #1 is the found text (the parenthesis), #2 the rest of the string to parse, #3 the accumulated parenthetical, and #4#5 the counter tokens.

```
\def\hi@pse@state@paren@dec#1#2#3#4#5\@stop{%
    \ifstrempty{#5}{%
        \hi@pse@acc@saveparam\hi@pse@paren{#3}%
        \hi@pse@state@afterparen{#2}%
    }{%
        \hi@pse@state@paren@add{#1}{#2}{#5}{#3}%
    }%
}
```

After a parenthetical has been closed. This can be spaces, a new parenthetical, a semicolon, or EOF.

```
\def\hi@pse@state@afterparen#1{%
    %\hi@pse@debug{State afterparen; reading #1}%
    \ifstrempty{#1}{%
        \hi@pse@acc@setcite{}{}% EOF; treat like a semicolon
    }{%
        \find@try\find@start{%
            {(}{\hi@pse@state@paren}%
            {;}{\hi@pse@acc@setcite{}}%
            { }{\hi@pse@state@afterparen}%
        }{#1}{\hi@pse@error{Invalid text after parenthetical}{#1}}%
    }%
}
```

Finders used throughout this code.

```
\make@find@in{ }
\make@find@in{]}
\make@find@start{at }
\make@find@start{(}
\make@find@start{;}
\make@find@start{ }
\make@find@eq{[}
\make@find@eq{(}
\make@find@eq{)}
\make@find@eq{;}
\make@find@eq{ }
```

Tests.

```
\def\hi@pse@debug#1{\immediate\write16{\detokenize{#1}}}
```