Defining References (`refs.dtx`)

Charles Duan

August 16, 2024

A reference represents a single citable work of authority. What constitutes a single work may be uncertain, particularly for statutes and multi-authored works, as described previously. The general test for whether two items are two references or subdivisions of a single reference is whether, if they were cited immediately in sequence, *id.* should be used for the second item. In some cases this is a matter of the writer's taste, so this package generally gives writers the flexibility to define references however they see fit.

If a reference is used in a citation string but not defined, this package will issue an error, or it will keep track of the missing citation. See `draw.dtx` for more information on missing references.

1 Data Model

Each reference consists of:

- A reference type, such as a book, case, or statute. The complete list of reference types is given in Part **??**. The reference type determines how the reference is formatted and what parameters are necessary to define the reference.
- An identifier nickname that will be used to identify this reference in citations (and, on occasion, inside other references). The nickname must be unique and generally should not contain spaces.
- Parameters providing information about the reference. For example, a book uses parameters for the author, title, and publication year. The reference type determines which parameters are required and optional; this is documented in Part **??**. There are also parameters of general applicability to all reference types, described in `genparams.dtx`. The complete list of parameters is given in Part **??**.
- A Table of Authorities category, described below.

2 Input Syntax

The package provides its own syntax for writers to use when defining references. As an alternative, `hibib.dtx` describes a compatibility layer that accepts Biblatex reference definitions, so long as the parameter names correspond with those of this package.

Generally, references are defined as follows:

```
\def⟨reference type⟩ {⟨name⟩} {
    ⟨param1⟩ = ⟨value1⟩,
    ⟨param2⟩ = ⟨value2⟩,
    ...
}
```

For example, to define the historic case on copyrightability:

```
\defcase{baker}{
    parties=Baker v. Selden,
    cite=101 U.S. 99,
    year=1880,
}
```

The input follows TEX syntax and uses the `keyval` package, so if there are equal signs or commas in the parameter values, then the parameter should be surrounded with braces.

Entering the parameter names for every reference can be tedious, so for common reference types there are shortcuts described in `cparse.dtx`.

Creates a new reference definer macro. #1 is the type name (used to create a `\def`⟨*type*⟩ macro), #2 a textual description (used only for generating documentation), #3 the default TOA category, and #4 the macro code to be run to define the reference. Executing the reference definition command should cause the definition of the following formatting macros of the form `\`⟨*prefix*⟩`@`⟨*ref-name*⟩, with prefixes given in `draw.dtx`.

For convenience of parameter definition macros, `\@this@case` is set to the name of the reference being defined.

```
\def\hi@newcite#1#2#3#4{%
    \DefaultTOACategory{#1}{#3}%
    \global\@namedef{def#1}##1##2{%
        \@ifundefined{fc@##1}{}{%
            \PackageError\hi@pkgname{%
                Citation for ##1 already defined
            }{%
                The citation for ##1 has already been defined.\MessageBreak
                Please select another name.
            }%
        }%
        \begingroup
            \def\@this@case{##1}%
            \hi@toacat@setdefault{##1}{#1}%
            \hi@param@set{#1}{##2}%
            #4%
            % \hi@bib@output
        \endgroup
    }%
}
```

Aliases a cite definer macro to another.

```
\def\hi@alias@cite#1#2{\@namedef{def#1}{\@nameuse{def#2}}}
```

Within a reference definer macro, defines a new formatting macro of the form \⟨*prefix*⟩@⟨*name*⟩, where #1 is ⟨*prefix*⟩ and #2 is ⟨*name*⟩. Mainly, this checks to make sure that a full citation macro isn't being redefined (indicating that the writer has used a duplicate name).

If a non-full form is being defined and it already exists, then the existing definition is not overwritten.

```
\def\hi@newcite@form#1#2{%
    \@ifundefined{#1@#2}{%
        \expandafter\protected@xdef\csname#1@#2\endcsname
    }{%
        \find@eq{fc}{#1}{%
            \PackageWarning\hi@pkgname{Refusing to redefine #1@#2}%
        }{}%
        \hi@newcite@form@gobble
    }%
}
\make@find@eq{fc}
\def\hi@newcite@form@gobble#1#{\@gobble}
```

3 Implementation of Parameters

This implementation documentation now turns to how parameters are stored and set internally. For every defined parameter, there is a macro \KV@hi@⟨*param*⟩ as created by the `keyval` package. There is also a macro \hi@kv@⟨*param*⟩ defined to hold the value that the parameter is set to, which is used by most but not all parameters. By default, the \hi@kv@⟨*param*⟩ variable is given a value that will generate an error if used in a citation, alerting the user to the parameter being necessary but missing.

This is a list of all the parameter registers. Each parameter is stored in the form \do{⟨*param*⟩}{⟨*description*⟩}.

```
\def\hi@params{}
```

Parenthetical parameters. We make a special token list of parenthetical creator commands, to be executed presumably upon the full citation.

```
\newtoks\hi@param@parens
```

Adds a parenthetical to the list. #1 is the parenthetical priority, #2 the text.

```
\def\hi@param@addparen#1#2{%
    \push@toks\hi@param@parens{%
        \hi@parens@add#1{#2}%
    }%
}
%
% Special token to set undefined parameters to.
%
%    \begin{macrocode}
\DeclareRobustCommand\hi@param@undef[1]{%
    \PackageError\hi@pkgname{%
        Citation parameter \noexpand#1 missing in \@this@case
    }{Include the missing citation parameter.}%
    \textbf{???}%
}

\def\hi@undefine#1{\def#1{\hi@param@undef{\string#1}}}
```

Set all parameters to empty. This will be called after all the parameters have been created. Since reference definitions are executed inside groups, these initial settings should carry over into all definitions.

```
\def\hi@param@clear{%
    \def\do##1##2{\expandafter\hi@undefine\csname hi@kv@##1\endcsname}%
    \hi@params
    \hi@param@parens{}%
}
\AtEndOfPackage{\hi@param@clear}
```

Parameter definers. #1 is the parameter, #2 is an explanation, and anything that follows should be arguments to `\define@key` excluding the keyset and parameter name.

```
\def\hi@defparam#1#2{%
    \g@addto@macro\hi@params{\do{#1}{#2}}%
    \define@key{hi}{#1}%
}
\def\hi@newparam#1#2{%
    \@expand{\hi@defparam{#1}{#2}}{%
        \expandafter\def\csname hi@kv@#1\endcsname{##1}%
```

```
    }i%
}
\def\hi@defparam@noval#1#2{%
    \hi@defparam{#1}{#2}[#1]%
}
\def\hi@newparam@noval#1#2{%
    \@expand{\hi@defparam@noval{#1}{#2}}{%
        \expandafter\def\csname hi@kv@#1\endcsname{##1}%
    }i%
}
\def\hi@param@alias#1#2{%
    \g@addto@macro\hi@params{\do{#1}{Alias for #2}}%
    \edef\reserved@a{%
        \noexpand\def\expandafter\noexpand\csname KV@hi@#1\endcsname{%
            \expandafter\noexpand\csname KV@hi@#2\endcsname
        }%
    }\reserved@a
}
```

3(a) Setting Parameters

The following macro is used when executing a reference definer, to read and set all the parameters in preparation for constructing the citation macros. It looks for a parseable formatted citation (either by finding a semicolon in the parameters, or determining that the parameters contain no equal sign), and otherwise uses `keyval` to read the parameters. It also executes any post-parameter hooks.

#1 is the reference type name, and #2 is the parameter set.

```
\def\hi@param@set#1#2{%
    \let\hi@param@set@hooks\@empty
    \def\hi@kv@kind{#1}%
    \hi@param@read{#2}%
    \hi@param@set@hooks
}
\def\hi@param@read#1{%
    \find@in{;}{#1}{%
        \hi@param@read@{#1}%
    }{\hi@param@read@{#1}{#1}{}}%
}
\def\hi@param@read@#1#2#3{%
    \find@in{=}{#2}{%
        \@firstofthree{\setkeys{hi}{#1}}%
    }{%
        \chop@space@then@run{#2}\hi@cparse
        \ifstrempty{#3}{}{\setkeys{hi}{#3}}%
    }%
}
\make@find@in{=}
\make@find@in{;}
```

Adds a post-parameter-setting hook. #1 is code to be run.

```
\def\hi@param@addhook#1{%
    \addto@macro\hi@param@set@hooks{#1}%
}
```

3(b) Testing Parameters in Citation Macros

Commands for testing parameters in citation macros.

```
\def\hi@ifset#1{%
    \expandafter\hi@ifset@#1.\@stop
}
\def\hi@ifset@#1#2\@stop{%
    \ifx\hi@param@undef#1\expandafter\@secondoftwo
    \else               \expandafter\@firstoftwo   \fi
}

\def\hi@ifeitherset#1#2{%
    \hi@ifset#1{\@firstoftwo}{\hi@ifset#2}%
}
```

4 Citation Groups

Often there will be several references that follow a standard conventional form, so it would be convenient to define all of them in one go. The `\defcitegroup` macro enables this to an extent. It takes four arguments:

- (optional) A prefix for the reference nicknames to be defined, default blank
- A reference definer macro (`\def`⟨*type*⟩)
- Initial text for the reference definition body
- A comma-separated list of texts used both as the nickname and as a suffix to the reference definition

For example, say you want to define references for several sections of the Patent Act. That could be achieved with the following three definitions:

```
\defstatcode{101}{35 U.S.C. S 101}
\defstatcode{102}{35 U.S.C. S 102}
\defstatcode{103}{35 U.S.C. S 103}
```

This works (using the reference parser for statutes) but is repetitive. Instead, you can write:

```
\defcitegroup\defstatcode{35 U.S.C. S }{101, 102, 103}
```

Each of the items in the comma-separated list (e.g., `101`) becomes both a reference nickname and the end of the reference definition.

With the optional argument, the reference names can be given a prefix:

```
\defcitegroup[17usc]\defstatcode{17 U.S.C. S }{101,
102, 103}
```

would define `17usc101` as the statute "17 U.S.C. § 101" and so on.

```
\newcommand\defcitegroup[4][]{%
    \forcsvlist{\hi@citegroup@{#1}{#2}{#3}}{#4}%
}
\def\hi@citegroup@#1#2#3#4{#2{#1#4}{#3#4}}
```

5 Table of Authorities Categories

Each reference type is associated with a category, used to organize a Table of Authorities as described in `toa.dtx`. The category can be altered using the reference definition parameter toacat or by the `\SetTOACategory` macro. The categories are used to assemble the headings for the table of authorities, as described in `toa.dtx`.

Specifically, every category has three properties:

1. A name for internal use.
2. A "supercategory" that associates the category with others that this category should be joined with in the Table of Authorities. For example, the categories for statutes and regulations should have the same supercategory, so that they are placed in a single section with the Table of Authorities. Supercategories are presented in alphabetical order.

3. A textual description of the category, for use in constructing the Table of Authorities heading. Because both the singular and plural forms are needed, the textual description consists of two to three parts, separated by slashes:

   (a) The base text common to both forms.

   (b) [*Optional*] Text to append to the singular form only.

   (c) Text to append to the plural form only.

A new TOA category may be defined with `\NewTOACategory`. The macro takes three parameters, corresponding to the three properties of categories above.

TODO: The underlying implementation here could be improved to rely less on marker symbols. Also, categories should have a precedence number indicating the order in which they should appear in the TOA heading. Finally, this section should probably be in the TOA chapter, not here.

```
\def\NewTOACategory#1#2#3{%
    \@namedef{hi@cats@#1}{#2:#3}%
    \AddToList{hi@toacat@supercats}{#2}%
}
```

5.1 Setting and Getting Categories

Each reference type may have a default category, set with the `\DefaultTOACategory` macro. The parameters are #1 the reference type, and #2 the category name. If the category name is blank, then there will be no default category. This default category will be used unless another category is specified for a particular reference, overriding the default category.

```
\def\DefaultTOACategory#1#2{%
    \ifstrempty{#2}{%
        \cslet{hi@catd@#1}\relax
    }{%
        \@ifundefined{hi@cats@#2}{%
            \PackageError{%
                TOA category #2 does not exist, in setting the\MessageBreak
                default for reference type #1%
            }{Define TOA category #2 first with \string\NewTOACategory}%
        }{%
            \@namedef{hi@catd@#1}{#2}%
        }%
    }%
}
```

The `\SetTOACategory{`⟨*reference*⟩`}{`⟨*category*⟩`}` macro sets the TOA category for ⟨*reference*⟩. This macro should be called within the reference definer macros to set the category, or it is invoked by the toacat parameter. If it is called multiple times, then the last category will prevail. If it is not called, then the reference will not appear in the table. If ⟨*category*⟩ is empty, then any previously set category will be removed.

```
\def\SetTOACategory#1#2{%
    \ifstrempty{#2}{%
        \global\cslet{hi@cat@#1}\relax
    }{%
        \@ifundefined{hi@cats@#2}{%
            \PackageError\hi@pkgname{%
                Invalid Table of Authorities category #1,\MessageBreak
                given for reference #2%
```

```
            }{%
                Please enter a valid Table of Authorities category.
            }%
        }{%
            \global\csletcs{hi@cat@#1}{hi@cats@#2}%
        }%
    }
}
```

Retrieves the TOA category for a reference #1, and executes a callback #2 with the category text as an argument. If the reference has no category, then the callback will not be executed.

```
\def\hi@toacat@get#1#2{%
    \@ifundefined{hi@cat@#1}{}{%
        \@expand{#2}{\csname hi@cat@#1\endcsname}{ii}%
    }%
}
```

Transfers a reference's category to another reference. #1 is the reference with the already-defined category; #2 is the reference to receive the category. The category for #1 is deleted.

```
\def\hi@toacat@transfer#1#2{%
    \global\csletcs{hi@cat@#2}{hi@cat@#1}%
    \global\cslet{hi@cat@#1}\relax
}
```

Sets a reference's TOA category to the reference type's default category. #1 is the reference name, and #2 the reference type. The category will not be changed if it is already set, or if the reference type has no default category.

```
\def\hi@toacat@setdefault#1#2{%
    \@ifundefined{hi@cat@#1}{%
        \@ifundefined{hi@catd@#2}{}{%
            \SetTOACategory{#1}{\csname hi@catd@#2\endcsname}%
        }%
    }{}%
}
```

5.2 Supercategories

Supercategories are stored in a sorted list (see `sortlist.dtx`), which is sorted alphabetically.

```
\NewSortedList{hi@toacat@supercats}{\SortEasyAlpha}{\StripForAlpha{#1}{\def#2}}
\ListElementsMustBeUnique{hi@toacat@supercats}
```

5.3 Predefined Categories

Definitions of the categories follow in the source code, but they are used to produce a table in the documentation, so please consult the package file for the code.

The following are the TOA categories, with their supercategories and display texts as described above. Note the choice of supercategory names, which coincidentally sort alphabetically in the correct order. Because the actual supercategory names are never displayed, any new supercategories can be named to fit into the existing ordering.

Category	Supercategory	Text
other	other	Other Source/s
const	const	Constitutional Provision/s
found	const	Foundational Document/s
statute	edict	Statute/s
regulation	edict	Regulation/s
rule	edict	Rule/s
case	case	Case/s
admin	case	Administrative Decision/s
treaty	edict	Treat/y/ies

This macro will perform a function for each of the Table of Authorities groups, in the proper order. The function should be given as a macro definition in the first argument (absorbed later), the macro definition accepting one argument which will be the supercategory name.

```
\def\hi@toacat@each{%
    \ShowList{hi@toacat@supercats}%
}
```