The `xt_capts` package*

Olaf Fricke
ofricke@cs.tu-berlin.de

1998/02/25

Abstract

This package allows to define language dependent text macros. They are called *extended captions*, because *caption* is the term that is used in LaTeX2e for denoting language dependent words.

Whenever a new language is selected, the appropriate macros are used, provided they have been defined. In addition, a default variant can be defined for *all* languages. And it is possible to use a fixed language for a caption, which is not affected by changing the current language.

1 Introduction

When doing some cooperative work within a team, it is often recommended to define some handy text commands, which everybody should use. In addition, it is sometimes necessary to use these commands in different languages. The command names should not change, but the represented text may vary from one language to the other. Similar ideas are used within a multilingual LaTeX2e, an examples is `\figurename`, which gives "Figure" in English and "Abbildung" in German. These commands are called *captions*, that is why this package is called `xt_capts`, an abbreviation for *extended captions*.

This package implements some commands, which can be used to define new language-specific captions. Most ideas (and even much of the documentation) were adapted from the LaTeX2e font encoding scheme, namely out of the `fontenc` package and its documentation file `ltoutenc.dtx`. Therefore it may not be very surprisingly, that the implementation is based on the two commands `\DeclareTextCommand` and `\ProvideTextCommand`.

1.1 Declaring captions

The syntax for declaring language-specific commands is as follows:

`\DeclareCaption{`⟨*caption*⟩`}{`⟨*language*⟩`}`
`[`⟨*number*⟩`][`⟨*default*⟩`]{`⟨*commands*⟩`}`
`\ProvideCaption{`⟨*caption*⟩`}{`⟨*language*⟩`}`
`[`⟨*number*⟩`][`⟨*default*⟩`]{`⟨*commands*⟩`}`

*This file has version number v1.1d, last revised 1998/02/25.

The first command acts like `\newcommand`, except that it defines a command which is specific to the given language, meaning it defines a new caption. The command accepts the same optional arguments as `\newcommand`, which allows somebody to define a caption with arguments. The resulting command is always robust, even if its definition is fragile.

The second command does nearly the same, except that nothing is done if the command is already defined for the given language. Additionally, the latter command is allowed inside the document, too, whereas the former one may be used only inside the preamble.

For example, to define the caption `\teapot` for English and German, the following code can be used:

```
\DeclareCaption{\teapot}{english}{\textbf{teapot}}
\DeclareCaption{\teapot}{german}{\textbf{Teekanne}}
```

1.2 Using captions

The defined captions can be used in two ways. Normally they are used by just calling the command after the appropriate language has been selected. However, sometimes it may be necessary to use a command in a language that is not selected (or even not installed). Therefore the `\UseCaption` command can be called:

`\UseCaption{`⟨*language*⟩`}{`⟨*caption*⟩`}`

For example, `\UseCaption{german}{\teapot}` has the same effect as:

```
{\selectlanguage{german}\teapot}
```

The first variant has the advantage, that it is much faster than changing the whole language with the `\selectlanguage` mechanism and that it can be used even if no language package is used at all.

After defining the above captions, `\teapot` gives either **teapot** or **Teekanne**, depending on the currently selected language.

Some more flexibility can be reached, if arguments are used in the declarations. Two examples are given in the following. The first one allows to use the caption in singular or in plural form, whereas the second example provides the possibility to change the leading letter.

```
\DeclareCaption{\teacup}{english}[1][\empty]{\textbf{cup#1 of tea}}
\DeclareCaption{\hotscones}{english}[1][h]{\textbf{#1ot scones}}
```

The use of `\teacup` leads to a **cup of tea**, whereas `\teacup[s]` gives some **cups of tea**. **Hot scones** are ordered with `\hotscones`[H], whereas **hot scones** are created with `\hotscones`.

Note that you can also use parameterized captions in conjunction with the `\UseCaption` command. This means, that if `\teacup` is defined for German like

```
\ProvideCaption{\teacup}{german}[1][\empty]{\textbf{Teetasse#1}}
```

you can say `\UseCaption{german}\teacup[n]` to typeset **Teetassen**. Please note that it is not necessary to put the caption command into braces.

1.3 Default captions

You can also declare a default definition for a caption command, which will be used if the current language has no appropriate definition. Such use will also set the definition for this command in the current language to equal this default definition; this makes subsequent uses of the command much faster.

`\DeclareCaptionDefault{`⟨*caption*⟩`}{`⟨*definition*⟩`}`

There is a matching `\Provide` command which will not override an existing default definition:

`\ProvideCaptionDefault{`⟨*caption*⟩`}{`⟨*definition*⟩`}`

1.4 Language switching

The package uses the `\selectlanguage` mechanism to recognize a switching of the current language and to set its internal language definition, just like the `babel` package (and various other packages) does to redefine existing captions like `\chaptername`.

Some facts have to be known by the implementation concerning the language selection mechanism:

1. When using language packages like `babel` or `german`, you can switch to another (installed) language `XXX` with the `\selectlanguage{XXX}` command. During its execution, three language dependent macros are evaluated, namely: `\dateXXX`, `\captionsXXX`, and `\extrasXXX`. The second macro is used by our `xt_capts` package to realize new language dependent captions.

2. If no language is installed at all, the `\selectlanguage` command may be defined nevertheless, since it is possibly included into the format file. But luckily the `\languagename` macro is undefined, such that a test can be performed.

3. There are some difficulties in installing and initializing the new captions, since the package should work with *all* language packages, which uses the the macros `\selectlanguage` and `\captionsXXX`. An example is the `babel` system. In the current release v3.6h, the `\selectlanguage` command is deferred until the begin of the document. In other package like the `german` package (release v2.5d), the `\selectlanguage` command is called immediately. But the sequence in which the different packages are used should be of no interest. Therefore we must take care of the ordering of deferred commands.

2 The Macros

First of all, some useful commands are defined. They are used to affect the execution ordering of commands which have been deferred until the begin or the end of the document. After this prelude, the declaration and the implementation of captions is handled as well as the using a fixed language for a caption. It follows the code used for initialization and installation of a new caption language. The last section describes how the default language is set.

2.1 Some useful commands

1 ⟨*package⟩

\g@prependto@macro

In correspondence to the command \g@addto@macro, a new command with the name \g@prepend@macro is defined. As you may guess, this command is used to prepend something to an already defined macro. The \g@addto@macro is defined in ltclass.dtx as follows:

```
\long\def\g@addto@macro#1#2{%
  \begingroup
    \toks@\expandafter{#1#2}%
    \xdef#1{\the\toks@}%
  \endgroup}
```

Since it has to be possible to prepend more than a single command, it is not sufficient to simply exchange the two parameters when setting the token. Instead, the second parameter has to be bound to a temporary macro. Otherwise we would fail to jump over the second parameter to expand the macro itself. The \expandafter command would step into the code that is to be prepended. In the mentioned command \g@addto@macro this detour is not necessary, because there it is not necessary to jump over the second parameter.

The extension of a macro is done similarly to the one used in \g@addto@macro: the new contents of the macro is put into \toks@ (after expanding both parameters) and than the command mentioned in the first parameter is defined with the help of the token register. To expand both parameters, the \expandafter command is used. Note that the old contents of the macro is expanded first.

```
\long\def\g@prependto@macro#1#2{%
  \begingroup \def\reserved@a{#2}%
    \toks@\expandafter\expandafter\expandafter{%
      \expandafter\reserved@a#1}%
    \xdef#1{\the\toks@}%
    \endgroup}
```

\BeforeAtBeginDocument
\AfterAtBeginDocument

The recently defined command is used in the following definitions. Similar to the command \AtBeginDocument, the new command \BeforeAtBeginDocument can be used to execute something at the beginning of a document. But in contrast to the former command, the code will be *prepended* to \@begindocumenthook. This will allow package writers to define some code which is executed *before* the deferred code of all other packages is executed.

It may be useful to execute some code *after* all packages have done their initializations at the beginning of the document. Looking deeply into the \document macro shows that the do-list \@preamblecmds can be misused for this purpose. The list consists of \do-commands, each followed by the command to be disabled. It is evaluated after the \@begindocumenthook has been executed. Its original purpose is to disable commands from execution after the document has been started. But it is possible to (mis)use this list as hook as well.

Note that both commands may be used only in the preamble.

```
\def\BeforeAtBeginDocument{\g@prependto@macro\@begindocumenthook}
\def\AfterAtBeginDocument#1{\g@prependto@macro\@preamblecmds{#1}}
\@onlypreamble\BeforeAtBeginDocument
\@onlypreamble\AfterAtBeginDocument
```

\BeforeAtEndDocument

Perhaps somebody wants to extend \@enddocumenthook at its head, too. This can be done by prepending some code to that hook. Unfortunately, there is no simple way to implement the analogous \AfterAtEndDocument command.

```
\def\BeforeAtEndDocument#1{\g@prependto@macro\@enddocumenthook{#1}}
```

2.2 Declaring language dependent captions and their defaults

This section describes the implementation of the essential commands belonging to this package. They all serve the purpose to define and use language dependent captions. The name space \xtc@ is used for internal macros.

\DeclareCaption
\ProvideCaption

The implementation is based on \DeclareTextCommand, a LaTeX2e kernel command, which is used for declaring encoding-specific commands. Some tricky declarations make sure that the same command can be used for language-specific commands, too.

The \xtc@init command is called to fulfill the initialization stuff. Note that \DeclareCaption can only be used in the preamble, but that the \Provide version is allowed anywhere.

```
\newcommand{\DeclareCaption}[2]{%
  \xtc@init{#2}%
  \DeclareTextCommand#1{#2}}
\newcommand{\ProvideCaption}[2]{%
  \xtc@init{#2}%
  \ProvideTextCommand#1{#2}}
\@onlypreamble\DeclareCaption
%\@onlypreamble\ProvideCaption
```

\DeclareCaptionDefault
\ProvideCaptionDefault

These macros define commands with our default language lang?. Note that \DeclareCaptionDefault can only be used in the preamble, but that the \Provide version is allowed anywhere.

```
\newcommand{\DeclareCaptionDefault}[1]{%
  \DeclareTextCommand#1{lang?}}
\newcommand{\ProvideCaptionDefault}[1]{%
  \ProvideTextCommand#1{lang?}}
\@onlypreamble\DeclareCaptionDefault
%\@onlypreamble\ProvideCaptionDefault
```

These macros require \lang?-cmd to be initialized as \xtc@changed@lang.

```
\expandafter\let\csname lang?-cmd\endcsname\xtc@changed@lang
```

2.3 Implementation of the language mechanism

When executing \DeclareCaption{\foo}{XXX}..., the standard kernel command \DeclareTextCommand is called with exactly the same arguments. During that call, \foo is defined to be \XXX-cmd \foo \XXX\foo, where \XXX\foo is ***one*** control sequence, not two. In addition, \newcommand is called to define \XXX\foo accordingly to what follows in the dots.

The documentation of `ltoutenc.dtx` states that all the tricky work is done when executing `\XXX-cmd \foo \XXX\foo`. That's fairly true. Since we are juggling with language switching and with protected commands, we must respect the current values of `\protect` and the actual language. Unfortunately, the macro `\languagename` is not always defined. Therefore we define `\xtc@lang` to contain the current language name for the captions and take an approach similar to the font encoding scheme. The approach is as follows:

- If `\protect` is not `\@typeset@protect`, then we execute `\noexpand\foo`. This make `\foo` robust, so that it will survive all things which may happen to it, even if it is written to a file or something else.
- If `\protect` is `\@typeset@protect` and `\xtc@lang` is `XXX`, then `\XXX\foo` is executed. Like in the font encoding scheme, this is the normal behavior, and it is optimized for speed.
- If `\protect` is `\@typeset@protect`, `\xtc@lang` is (say) `YYY`, and `\YYY\foo` is defined, then we execute `\YYY\foo`.
- If `\protect` is `\@typeset@protect`, `\xtc@lang` is (say) `YYY`, and `\YYY\foo` is undefined, then we define `\YYY\foo` to be equal to the default value of `\foo`, and execute `\YYY\foo`.
- In all cases, if `\protect` is `\@typeset@protect` and we are in math mode, then an additional warning is issued.

So after all that, we will either execute the appropriate definition of `\foo` for the current language, or we will execute `\noexpand\foo`.

The default value of `\foo` is `\lang?\foo` if it is defined, and an error message otherwise.

`\xtc@use@lang` `\xtc@lang`

If the language is changed from `XXX` to `YYY`, `\XXX-cmd` is defined to be an alias to `\xtc@changed@lang`, whereas `\YYY-cmd` is defined to be an alias to `\xtc@current@lang`. This means that the test for what the current language is can be performed quickly. More precisely, there is no need to make any test at all, because the right command is called on the fly. The language change is done by calling `\xtc@use@lang`.

```
\def\xtc@use@lang#1{%
  \expandafter\let\csname\xtc@lang -cmd\endcsname\xtc@changed@lang
  \edef\xtc@lang{#1}%
  \expandafter\let\csname\xtc@lang -cmd\endcsname\xtc@current@lang}
```

In addition, the macro `\xtc@lang` (denoting the current language) is initialized to be `lang?`. This has to be done, because otherwise the macro would be undefined if the first language is selected.

```
\def\xtc@lang{lang?}
```

`\xtc@current@lang` `\xtc@changed@lang` `\xtc@unavailable`

The next macro treats the case that a caption uses the current language. It simply tests if we are in protected modus. If not, the math warning is raised optionally, just before the next command will be executed. Remember that this command is the `\XXX\foo` one. If we are in protected modus, we do a `\noexpand\foo` and remove the next command by means of the `\@gobble` command.

```
\def\xtc@current@lang#1{%
   \ifx\protect\@typeset@protect
      \@inmathwarn#1%
   \else
      \noexpand#1\expandafter\@gobble
   \fi}
```

Things differ a bit, if the language has changed. First of all, we remove the `\XXX\foo` and save it in the second parameter. Note that we do not use it any more. The test if we are in protected modus is the same, the actions differ. Since we have already removed the next command, we simply do our `\noexpand\foo` in case of protection. Otherwise we possibly raise our math warning, before processing the command.

We have to take care of undefined commands. If the desired is undefined for the current language, it is let to be an alias to the default command. If the default command is undefined, it is defined to give an error message. The last step of our processing is to evaluate the desired command.

```
\def\xtc@changed@lang#1#2{%
   \ifx\protect\@typeset@protect
      \@inmathwarn#1%
      \expandafter\ifx\csname\xtc@lang\string#1\endcsname\relax
         \expandafter\ifx\csname lang?\string#1\endcsname\relax
            \expandafter\def\csname lang?\string#1\endcsname{%
               \xtc@unavailable#1%
            }%
         \fi
         \global\expandafter\let
               \csname\xtc@lang \string#1\expandafter\endcsname
               \csname lang?\string#1\endcsname
      \fi
      \csname\xtc@lang\string#1%
         \expandafter\endcsname
   \else
      \noexpand#1%
   \fi}
```

Now we define the error message `\xtc@unavailable`. Note that the mentioned warning `\@inmathwarn` is used out of the font encoding code as it is.

```
\gdef\xtc@unavailable#1{%
  \@latex@error{%
    Command \protect#1 unavailable in language \xtc@lang%
    }\@eha}
```

2.4 A command for using a fixed language

`\UseCaption` The `\UseCaption` command can be used even for captions which have parameters. It is important to make the command robust, since it can be use inside a `\section` command to prevent a heading from being typeset in the wrong language.

Our first implementation approach set the desired language before evaluating the caption. It finished with restoring the language to its old value. The coding was straight forward:

```
\declare@robustcommand{\UseCaption}[2]{%
```

```
\let\xtc@saved@lang\xtc@lang
\xtc@use@lang{#1}%
#2%
\xtc@use@lang\xtc@saved@lang}
```

Unfortunately, this was not the very best solution, because an appended `\xspace` was misleaded. Therefore the implementation was changed. Now nothing will follow the expanded caption.

The implementation takes an approach very similar to the `\xtc@changed@lang` command. Since the `\UseCaption` command is defined to be robust, we need not to check the value of `\protect`. Instead, we only check if the desired language command is defined or not. If not, we have a look at the default command. If the latter is not defined as well, it is defined to give our error. Now we have the default command available and let the desired command be an alias to the default command. Finally, we evaluate the desired command.

```
\declare@robustcommand\UseCaption[2]{%
  \@inmathwarn#2%
  \expandafter\ifx\csname #1\string#2\endcsname\relax
    \expandafter\ifx\csname lang?\string#2\endcsname\relax
      \expandafter\def\csname lang?\string#2\endcsname{%
        \xtc@unavailable#2%
        }%
    \fi
    \global\expandafter\let
    \csname #1\string#2\expandafter\endcsname
    \csname lang?\string#2\endcsname
  \fi
  \csname #1\string#2\endcsname
  }
```

2.5 Initialization and installation stuff

`\xtc@init` `\xtc@install`

Each time a caption is declared or provided, the command `\xtc@init` is called with the desired language as parameter, say `XXX`. The purpose of this call is to initialize a new language if called the first time with `XXX` and to do nothing otherwise.

Therefore the command `\xtc@XXX` is tested. If this command is undefined, the initialization happens, otherwise the call returns. When performing the test, it must be taken into account, that `\csname` lets an undefined command to `\relax`. If undefined, it is defined to set the new language. In addition, the `\XXX-cmd` is let to `\xtc@changed@lang` and the new `\xtc@XXX` is installed.

Since the initialization command may be called by `\ProvideCaption` after the document has started, it is tested, if `\@begindocumenthook` is defined. If not, we are inside the document and can install `\xtc@XXX` immediately. If defined, we are still in the preamble and have to defer the installation until the beginning of the document.

The deferring has to be done because the command `\xtc@XXX` is to be appended to the command `\captionsXXX`. And the latter command may be undefined by now, because the use of packages like `babel` or `german` may happen later than the use of our package `xt_caps`.

```
\newcommand{\xtc@init}[1]{%
```

```
\expandafter\ifx\csname xtc@#1\endcsname\relax
  \expandafter\def\csname xtc@#1\endcsname{%
    \xtc@use@lang{#1}}%
  \expandafter\let\csname #1-cmd\endcsname\xtc@changed@lang
  \ifx\@begindocumenthook\@undefined
    \xtc@install{#1}%
  \else
    \AtBeginDocument{\xtc@install{#1}}%
  \fi
\fi}
```

The following macro will be called at the beginning of the document. Its purpose is to install the extended caption mechanism for a given language `XXX`. First of all, it is tested if there exists any support for the language. This is done by looking at the existence of the command `\captionsXXX`. If no support is given, a warning is raised. Otherwise we append our `\xtc@XXX` command to `\captionsXXX`. Some care must be taken, because the `\csname` command lets an unknown command to be `\relax`. We use a group to restore the command to its previous value.

```
\newcommand{\xtc@install}[1]{%
  \begingroup\expandafter\endgroup
  \expandafter\ifx\csname captions#1\endcsname\@undefined
    \PackageWarningNoLine{xt_capts}{%
      You haven't defined the language #1\space yet.\MessageBreak
      Nevertheless, \string\UseCaption{#1}{\string\foo} will work}
  \else
    \expandafter\g@addto@macro\csname captions#1\expandafter\endcsname
      \csname xtc@#1\endcsname
  \fi}
```

2.6 Default language

To set the default language for the captions, we must check if different languages are supported. If support is given, the current language is used for the captions as well. If not, `english` is used. The code for doing this selection is deferred until all other packages have executed their deferred code. This is necessary, because language package may execute `\selectlanguage` at the beginning of the document, too. It is important for the caption implementation, that the default language of the captions is defined after the actual language has been selected. Unfortunately, the `\selectlanguage` command for a given language may be executed *before* `\xtc@XXX` has been installed or even before our package is loaded. One candidate for this behavior is the `german` package, which does not defer its `\selectlanguage` call. Therefore, we put our piece of code somewhere *after* the `\@begindocumenthook`.

```
\AfterAtBeginDocument{%
  \ifx\languagename\@undefined
    \xtc@use@lang{english}%
  \else
    \xtc@use@lang\languagename
  \fi}
```

To prevent the default language `english` from producing an undesired warning message, another piece of code it put into the `\@begindocumenthook`. This piece

lets `\captionsenglish` to be an empty macro, if it is undefined and no language package is used at all. Because we want to be sure that this code is executed *before* any caption language is installed, we put our piece of code at the head of the hook.

```
\BeforeAtBeginDocument{%
  \ifx\languagename\@undefined
    \ifx\captionsenglish\@undefined
      \let\captionsenglish\@empty
    \fi
  \fi}
```

⟨/package⟩