Using array structures in LaTeX: The package forarray

Version 1.01 (2008/06/20)

Christian Schröppel[*]

June 20, 2008

Abstract

The package forarray provides functionality for processing lists and array structures in LaTeX. Arrays can contain characters as well as TeX and LaTeX commands, nesting of arrays is possible, and arrays are processed within the same brace level as their surrounding environment. Array levels can be delimited by characters or control sequences defined by the user. Practical uses of this package include data management, construction of lists and tables, and calculations based on the contents of lists and arrays.

Contents

---

[*] Please send any comments or suggestions to christian@schroeppel.com.

1 Introduction

TeX as well as LaTeX do not provide any native functionality for processing arrays. Command structures such as `\loop` ... `\repeat` cannot be easily nested, and processing list or array items that contain arbitrary TeX code, including braces that should be preserved during the execution, is difficult to implement.

The package forarray provides the commands `\ForEach` and `\ForArray`, as well as some additional commands, that simplify processing of array structures in LaTeX.

It offers the following features:

- Array data may contain arbitrary LaTeX code.[1]
- Array delimiters are chosen by the user, which is especially useful if you intend to process data provided in a format that you cannot or do not want to change.
- Arrays are being processed at the same brace level as their environment, so that any variables or commands defined while processing an array keep their values or definitions without using `\global`.
- While processing the array, braces supplied with the content of the array are being preserved.

When processing arrays, the macros provided by the package forarray create functions that are used for processing each array level "on the fly". This approach is essential for being able to nest arrays within each other. However, it creates a certain processing overhead. This will usually not matter much if you use the forarray package commands within a LaTeX document. Package writers, when defining commands that repeatedly process large amounts of data, should keep in mind that solutions tailor-made for specific data can avoid this overhead.

The name forarray indicates that this package offers functionality similar to the structures `for ... next` or `for ... each` that can be found in most programming languages.

---

[1] There are some restrictions, however: Category code changes of characters take effect only after a `\ForEach` or `\ForArray` command has been processed, and they do not take effect immediately while processing a `\ForEachD` command.

2 Files

The package forarray consists of the master file forarray.dtm, the file README.txt, and the derived files forarray.dtx, forarray.sty, forarray.pdf, forarray-test.tex, and forarray-test.pdf.

The file forarray.pdf contains the documentation for the package forarray.

The file README.txt contains some basic information about the package.

The file forarray-test.tex can be used to obtain a test page for the package. You can run this file with your LaTeX installation and compare the result with the file forarray-test.pdf to check if everything works well.

You can obtain the files forarray.dtx, forarray.sty, forarray.pdf, forarray-test.tex, and forarray-test.pdf, as well as README.txt by placing the files forarray.dtm, forarray.dts and the Bourne shell script forarray in a single directory and typing in `forarray` at the command prompt (after `cd` ⟨*your empty directory*⟩). The installation script forarray and the documentation style file forarray.dts are not part of the package forarray. Please note, however, that the provisions in the section "No warranty" of the LaTeX Project Public License (LPPL), version 1.3c, exempting the author and other parties from liability with regard to the work, apply to the contents of the package as well as to these files.

3 Usage

The package forarray provides commands for the following tasks:

Processing lists Use `\ForEach` or one of the other commands described in section 3.1.

Processing arrays Use `\ForArray`, described in section 3.2 on page 6.

Defining variables Use `\DefineArrayVar` or one of the other commands described in section 3.3 on page 8.

In order to use the package, you need to save the file forarray.sty to a directory where your LaTeX installation will find it. This will often be ⟨*your local texmf directory*⟩/tex/latex/forarray. Please consult the manuals for your LaTeX system for further information. In many cases, it will be possible to use a package manager provided with your LaTeX system to install the package.

As with most other packages, simply include the package in your file with the command `\usepackage{forarray}`. Currently, you cannot supply any options to the package forarray.

3.1 Processing lists

3.1.1 The command `\ForEach`

`\ForEach` The command `\ForEach` can be used to process the items of a list. It has the following syntax:

`\ForEach{`⟨*separator*⟩`}{`⟨*function*⟩`}{`⟨*list*⟩`}`

⟨*separator*⟩ This can be any character or control sequence. Often, a comma or a semicolon will be used as the separator.

`\thislevelitem`
`\thislevelcount`

⟨*function*⟩ This is the function that will be executed for every item in the list. You can use the token `\thislevelitem` to reference the contents of the item that is being processed.[2] The variable `\thislevelcount` contains the number of the position of the item within the list. This is a ⟨*count*⟩ token, so you must prefix it by `\the`, `\number`, `\romannumeral` or another appropriate command if you want to print out its content.

⟨*list*⟩ This argument contains the items of your list. Space characters of category code 10 will be ignored if they immediately precede or follow a separator.

Figure 1 illustrates the usage of `\ForEach`.

Figure 1: Usage of `\ForEach`

```
\begin{itemize}
\ForEach
{,}
{\item This is item No.\ %
\the\thislevelcount.\\
It contains: ``\thislevelitem''.}
{A \textbf{bold} word,
\textit{Some more words, written
in italics},
\multiply\thislevelcount\thislevelcount
the number \the\thislevelcount,
Some more text}
\end{itemize}
```

- This is item No. 1.
  It contains: “A **bold** word”.
- This is item No. 2.
  It contains: “*Some more words, written in italics*”.
- This is item No. 3.
  It contains: “the number 9”.
- This is item No. 4.
  It contains: “Some more text”.

3.1.2 The command `\ForEachX`

`\ForEachX`

The command `\ForEachX` processes the list of items in the same way as the command `\ForEach`. However, it expands its third argument, a token containing the actual list, before processing it. It has the following syntax:

`\ForEachX{`⟨*separator*⟩`}{`⟨*function*⟩`}{`⟨*list token*⟩`}`

⟨*separator*⟩ Same as in `\ForEach`.

⟨*function*⟩ Same as in `\ForEach`.

⟨*list token*⟩ This is a control sequence or an active character that expands to the list that is to be processed by `\ForEachX`.

3.1.3 The command `\ForEachSublevel`

`\ForEachSublevel`

The command `\ForEachSublevel` processes the the contents of a item of a surrounding list processing command. It is based on the command `\ForEachX`, but operates on the expansion of the token `\thislevelitem` taken from the surrounding list. It has the following syntax:

`\ForEachSublevel{`⟨*separator*⟩`}{`⟨*function*⟩`}`

---

[2] `\thislevelitem` is a `\long` macro that expands to the token list of the item. If you intend to compare its contents with the contents of other macros, please make sure that these macros are also defined as `\long` macros, e.g. by using `\newcommand`.

⟨***separator***⟩ Same as in `\ForEach`.

⟨***function***⟩ Same as in `\ForEach`.

With the command `\ForEachSublevel`, lists can be easily nested, as shown in Figure 2.

Figure 2: Nested Lists

```
\begin{enumerate}
\ForEach
{;}
{
\item This is item No.\ %
\the\thislevelcount.
\begin{enumerate}
\ForEachSublevel
{,}
{\item \thislevelitem.}
\end{enumerate}
}
{This is a nested item,
Another nested item;
{This item is, well, nested},
A final item}
\end{enumerate}
```

1. This is item No. 1.
    (a) This is a nested item.
    (b) Another nested item.
2. This is item No. 2.
    (a) This item is, well, nested.
    (b) A final item.

3.1.4 The command `\ForEachD`

`\ForEachD`

The command `\ForEachD` processes the list of items in the same way as the command `\ForEach`. However, it does not the contents of the list as its third argument, but parses the characters supplied after the second argument. These characters have to be delimited by the separator, followed by the control sequence `\endforeach`.[3] It has the following syntax:

`\ForEachD{`⟨*separator*⟩`}{`⟨*function*⟩`}`⟨*list*⟩⟨*separator*⟩
`\endforeach`

⟨***separator***⟩ Same as in `\ForEach`.

⟨***function***⟩ Same as in `\ForEach`.

⟨***list***⟩ Same as in `\ForEach`. The ⟨*list*⟩ must be supplied *without* surrounding braces.

The command `\ForEachD` allows to change the category codes of characters during the execution of the list. However, if the commands changing the category codes are supplied as part of the list, these changes only take effect after the current item has been processed. Figure 3 on the next page illustrates how category codes can be changed during the execution of `\ForEachD`.

3.1.5 The command `\ExitForEach`

`\ExitForEach`

The command `\ExitForEach` stops the execution of the list after the current item has been processed. It does not take any arguments. Figure 4 on the following page illustrates the usage of `\ExitForEach`.

---

[3] The token `\endforeach` is never actually expanded.

Figure 3: Changing category codes within a list

```
\ForEachD
{;}
{\thislevelitem\par\bigskip}
* Huh? What is this?;
\catcode`\*\active * I still have no clue!;
\newcommand{*}{\textbf{star}}
Now I see \textellipsis\ This is a *!;
I remember, this is a *.;\endforeach
```

* Huh? What is this?

* I still have no clue!

Now I see … This is a **star**!

I remember, this is a **star**.

Figure 4: Usage of `\ExitForEach`

```
\newcommand{\LeaveThisPlace}
{Leave this place!}
\ForEach
{;}
{-- This item says:
``\thislevelitem''\par\medskip
\ifx\thislevelitem\LeaveThisPlace
\ExitForEach Ooops, I have to quit.\par
\else
Continuing to the next item
\textellipsis\par\bigskip
\fi}
{Please continue.;Please continue.;
Leave this place!;Where has he gone?}
```

– This item says: “Please continue.”

Continuing to the next item …

– This item says: “Please continue.”

Continuing to the next item …

– This item says: “Leave this place!”

Ooops, I have to quit.

3.2 Processing arrays

3.2.1 The command `\ForArray`

`\ForArray` The command `\ForArray` processes the contents of an array. The first arguments supplied to this command specify the separators for each level of the array, a grouping marker for each array level, a token or active character that can be for executing a sublevel of the array, and the functions that the command `\ForArray` executes for each item of the array at the respective levels. The command `\ForArray` has the following syntax:

```
\ForArray{⟨separator list⟩}[[⟨marker list separator⟩]⟨marker list⟩]
         {⟨sublevel token⟩}{⟨function list separator⟩}
         {⟨function list⟩}{⟨array⟩}
```

⟨*separator list*⟩ This is a list of characters or control sequences that are used to separate the respective level of the array. For example, if you have an array with a semicolon as first level separator and a comma as second level separator, you would use `{;,}` as the `{`⟨*separator list*⟩`}`.

⟨*marker list separator*⟩ Optional argument. This is a control sequence or active character that separates the markers in the ⟨*marker list*⟩. It can only be used if a ⟨*marker list*⟩ is supplied as well. If the ⟨*marker list*⟩ is supplied without a ⟨*marker list separator*⟩, each character or control sequence of the marker list is taken as separate marker.

⟨*marker list*⟩ Optional argument. This is a list of characters or control sequences that are used as markers for the respective level of the array.

`\thislevelmarker`

You can use the token `\thislevelmarker` to reference the marker of the respective level.

⟨***sublevel token***⟩ This is a control sequence or an active character that expands to the function that processes the level of the array below the level that is being processed at the respective time. If you intend to process the lower levels of your array, you have to include this token in the ⟨*function list*⟩ as part of the functions for the higher levels of your array. The ⟨*sublevel token*⟩ expands to the content of the item if used in the function applied to the lowest level of the array.[4]

⟨***function list separator***⟩ This is a control sequence or active character that separates the functions in the ⟨*function list*⟩.

`\thislevelnr`

⟨***function list***⟩ This is a list of functions for the respective levels of the array, separated by the ⟨*function list separator*⟩. The variables `\thislevelitem` and `\thislevelcount` can be used in the same way as with the `\ForEach` command. In addition, the variable `\thislevelnr` can be used. It contains the number of level within the array. This is a ⟨*count*⟩ token, so you must prefix it by `\the`, `\number`, `\romannumeral` or another appropriate command if you want to print out its content.

⟨***array***⟩ This argument contains the items of your array. Space characters of category code 10 will be ignored if they immediately follow a separator.

Figure 5 illustrates the usage of `\ForArray`.

Figure 5: Usage of `\ForArray`

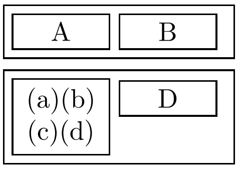
(a) Basic usage

```
\ForArray{;,}{*}{|}
{[*]\par|(*)}
{A,B;C,D}
```

[(A)(B)]
[(C)(D)]

(b) Nested arrays

```
\ForArray{;,}{*}{|}
{\framebox{*}\par\smallskip|
\framebox{\parbox[t]{3em}
{\centering*}}\ }
{A,B;%
\ForArray{;,}{*}{|}
{\centering*\par|(*)}
{a,b;c,d},D}
```



3.2.2 The command `\ExitForEachLevels`

`\ExitForEachLevels`

The command `\ExitForEachLevels` stops the execution of the array at the levels indicated in its arguments have been processed. It takes effect after the current item has been processed. Its syntax is as follows:

`\ExitForEachLevels{`⟨*Start level*⟩`}{`⟨*Number of levels*⟩`}`

---

[4] The ⟨*sublevel token*⟩ actually expands to `\fa@array@level`, which in turn expands to `\fa@array@`⟨*array level*⟩. The token `\fa@array@`⟨*array level*⟩ either expands to the function that processes the level of the array below the current level or, if used at the lowest level of the array, to `\thislevelitem`, the reference to the current item. Using `\thislevelitem` avoids this overhead.

⟨***Start level***⟩ This is the number of the lowest level that is exited after the command `\ExitForEachLevels` is being read. The number is relative to the current level, i.e. a value of 1 indicates that the current level is the lowest level that is being exited, a value of 2 indicates that the level above the current level is the lowest level that is being exited.

⟨***Number of levels***⟩ This is the number of levels that are being exited. The levels are being exited in consecutive order, beginning with the lowest level, which is being supplied as ⟨*Start level*⟩.

During the processing of an array item, you can supply multiple `\ExitForEachLevels` commands. `\ForArray` will exit from all levels indicated in the `\ExitForEachLevels` commands that have been supplied. The command `\ExitForEachLevels{1}{1}` has the same effect as the command `\ForEachExit`.

3.3 Defining variables

3.3.1 The command `\DefineArrayVar`

`\DefineArrayVar`

The command `\DefineArrayVar` defines a group of variables that are being named as ⟨*array name*⟩⟨*variable name separator*⟩⟨*variable name*⟩. Its syntax is as follows:

```
\DefineArrayVar{⟨array name⟩}
               {⟨variable name separator⟩}
               {⟨variable list separator⟩}
               {⟨variable list⟩}
               {⟨variable content list separator⟩}
               {⟨variable content list⟩}
```

⟨***array name***⟩ The names of all variables that are being defined by `\DefineArrayVar` start with the sequence of characters supplied as ⟨*array name*⟩. You can use any characters that TeX accepts as part of a `\csname` ... `\endcsname` construct.

⟨***variable name separator***⟩ This character or sequence of characters separates the ⟨*array name*⟩ and the ⟨*variable name*⟩. You can use any characters that TeX accepts as part of a `\csname` ... `\endcsname` construct.

⟨***variable list separator***⟩ The ⟨*variable list separator*⟩ separates the variable names supplied with the ⟨*variable list*⟩. It is a single control sequence or character.

⟨***variable list***⟩ The ⟨*variable list*⟩ contains the ⟨*variable name*⟩s that are used to construct the names of the variables in the array. You can use any characters that TeX accepts as part of a `\csname` ... `\endcsname` construct.

⟨***variable content list separator***⟩ The ⟨*variable content list separator*⟩ separates the contents of the variables supplied with the ⟨*variable content list*⟩. It is a single control sequence or character.

⟨***variable content list***⟩ The ⟨*variable content list*⟩ contains the contents of the variables in the array.

If you supply less items in the ⟨*variable content list*⟩ than in the ⟨*variable list*⟩, the content of the remaining variables will be set to the token `\relax`. If you supply more items in the ⟨*variable content list*⟩ than in the ⟨*variable list*⟩, the remaining content items will be ignored. Figure 6 on the following page illustrates the usage of `\DefineArrayVar`.

Figure 6: Usage of `\DefineArrayVar`

```
\makeatletter
\fontfamily{upl}\selectfont

\DefineArrayVar{Capital}{@}
{,}{Brazil,Japan,South Korea,Viet Nam}
{,}{Bras\'{i}lia,T\={o}ky\={o},
S\u{o}ul,H\^{a} N\^{o}i}

\DefineArrayVar{Name}{@}
{,}{Brazil,Japan,South Korea,Viet Nam}
{,}{Brazil,Japan,the Republic of Korea,
Vi\d\ecircumflex t Nam}

\newcommand{\CapitalCity}[1]
{The capital city of
\csname Name@#1\endcsname\ is
\csname Capital@#1\endcsname.}

\ForEach{,}
{\CapitalCity{\thislevelitem}
\par\medskip}
{Brazil,Japan,South Korea,Viet Nam}
```

The capital city of Brazil is Brasília.

The capital city of Japan is Tōkyō.

The capital city of the Republic of Korea is Sŏul.

The capital city of Việt Nam is Hâ Nôi.

3.3.2 The command `\DefineArrayVars`

`\DefineArrayVars`

The command `\DefineArrayVars` defines a set of grouped variables that are being named as ⟨*array name*⟩⟨*variable name separator*⟩⟨*variable name*⟩. The ⟨*array name*⟩ is taken from a list of names supplied by the user. The syntax of `\DefineArrayVars` is as follows:

`\DefineArrayVars{`⟨*variable list separator*⟩`}`
`{`⟨*array definitions separator*⟩`}`
`{`⟨*array name/content separator*⟩`}`
`{`⟨*content list separator*⟩`}`
`{`⟨*variable name separator*⟩`}`
`{`⟨*variable list*⟩`}`
`{`⟨*content definition list*⟩`}`

A precise explanation of the functions of the respective arguments would probably be rather lengthy and somewhat confusing. Therefore, the command `\DefineArrayVar` will be explained with the help of an example. Figure 7 on the next page shows that the result already produced with the use of `\DefineArrayVar` (see Figure 6) can be obtained in a shorter and more structured way by using `\DefineArrayVars`.

In the example, the *first argument* of `\DefineArrayVars` is a comma. It separates the names of the sequences of character and/or tokens that are being used to construct the second part of the names of the array variables.

The *second argument* of `\DefineArrayVars` is a slash. It is used to delimit the arrays that you intend to define. In this case, only one slash is needed: It separates the array of "Capital" variables from the array of "Name" variables.

The *third argument* separates the array name from the list of contents. In this case, a colon is being used.

Figure 7: Usage of `\DefineArrayVars`

```
\makeatletter
\fontfamily{upl}\selectfont

\DefineArrayVars{,}{/}{:}{;}{@}
{Brazil,Japan,South Korea,Viet Nam}
{Capital:
Bras\'{i}lia;T\={o}ky\={o};
S\u{o}ul;H\^{a} N\^{o}i/
Name:
Brazil;Japan;the Republic of Korea;
Vi\d\ecircumflex t Nam}

\newcommand{\CapitalCity}[1]
{The capital city of
\csname Name@#1\endcsname\ is
\csname Capital@#1\endcsname.}

\ForEach{,}
{\CapitalCity{\thislevelitem}
\par\medskip}
{Brazil,Japan,South Korea,Viet Nam}
```

The capital city of Brazil is Brasília.

The capital city of Japan is Tōkyō.

The capital city of the Republic of Korea is Sŏul.

The capital city of Việt Nam is Hâ Nôi.

The *fourth argument*, which, in this example, is a semicolon, separates the items of the list of content definitions.

The *fifth argument* is the string of characters that is being used to separate the array name from the variable name. Here, an *at* sign is being used, and the resulting names of the variables are `\Capital@Brazil`, `\Capital@Japan`, ... `\Name@Viet␣Nam`.[5]

The *sixth argument* and the *seventh argument* contain the list of variable names and the list of array names and contents, respectively. Both have to be structured accordings to the specifications supplied with the preceding arguments.

3.3.3 The command `\DefineArrayDefault`

`\DefineArrayDefault`

The command `\DefineArrayDefault` can be used to access the contents of a group of variables that have been defined with `\DefineArrayVar` or `\DefineArrayVars`. It defines a new control sequence whose expansion depends on the value of another variable. If, for a particular content of the second variable, no specific expansion has been defined, the variable expands to a default content. The syntax of `\DefineArrayDefault` is as follows:

```
\DefineArrayDefault{⟨array list separator⟩}
                   {⟨variable name separator⟩}
                   {⟨index variable⟩}
                   {⟨default variable⟩}
                   {⟨array list⟩}
```

---

[5] Be aware that you cannot directly access control sequences whose names contain non-letter characters without changing the category codes of the respective characters.

⟨*array list separator*⟩ The ⟨*variable list separator*⟩ separates the array names supplied with the ⟨*array list*⟩. It is a single control sequence or character.

⟨*variable name separator*⟩ Same as in `\DefineArrayVar`.

⟨*index variable*⟩ The variable that expands to one of the names of the variables supplied with the ⟨*variable list*⟩ of `\DefineArrayVar` or `\DefineArrayVars`. If this variable does not expand to one of these names, the name of the default index variable is being used, resulting in an expansion to the default content.

⟨*default variable*⟩ The name of the default variable.

⟨*array list*⟩ This in a list of array names, for which the index variable and the default variable are being defined.

Figure 8 illustrates the usage of `\DefineArrayDefault`.

Figure 8: Usage of `\DefineArrayDefault`

```
\makeatletter
\fontfamily{upl}\selectfont

\DefineArrayVars{,}{/}{:}{;}{@}
{Brazil,Japan,South Korea,Viet Nam,Unknown}
{Capital:
Bras\'{i}lia;T\={o}ky\={o};
S\u{o}ul;H\^{a} N\^{o}i;unknown/
Name:
Brazil;Japan;the Republic of Korea;
Vi\d\ecircumflex t Nam;this country}

\DefineArrayDefault{,}{@}{\country}%
{Unknown}{Capital,Name}

\newcommand{\WhatIsTheCapitalCity}
{-- You've asked for the capital city of
\textit{\country}.\par\medskip
The capital city of \Name\ is
\textbf{\Capital}.\par\bigskip}

\newcommand{\country}{Brazil}
\WhatIsTheCapitalCity
\renewcommand{\country}{South Korea}
\WhatIsTheCapitalCity
\renewcommand{\country}{a strange country}
\WhatIsTheCapitalCity
```

– You've asked for the capital city of *Brazil*.

The capital city of Brazil is **Brasília**.

– You've asked for the capital city of *South Korea*.

The capital city of the Republic of Korea is **Sŏul**.

– You've asked for the capital city of *a strange country*.

The capital city of this country is **unknown**.

3.3.4 The command `\DefineArrayVarTo`

`\DefineArrayVarTo`

The command `\DefineArrayVarTo` assigns the same content to a group of variables. It has the following syntax:

`\DefineArrayVarTo{`⟨*variable list separator*⟩`}`
`{`⟨*variable name separator*⟩`}{`⟨*array name*⟩`}`
`{`⟨*content*⟩`}{`⟨*variable list*⟩`}`

⟨***variable list separator***⟩ Same as in `\DefineArrayVar`.

⟨***variable name separator***⟩ Same as in `\DefineArrayVar`.

⟨***array name***⟩ Same as in `\DefineArrayVar`.

⟨***content***⟩ The content that is being assigned to the variables.

⟨***variable list***⟩ Same as in `\DefineArrayVar`.

3.3.5 The command `\CommandForEach`

`\CommandForEach`

The command `\CommandForEach` executes the same control sequence (or token list) for each item in a list. The items can consist of one or more characters or tokens. The command `\CommandForEach` has the following syntax:

`\CommandForEach{`⟨*variable list separator*⟩`}{`⟨*command*⟩`}`
`{`⟨*variable list*⟩`}`

⟨***list separator***⟩ Same as in `\ForEach`.

⟨***command***⟩ This is the control sequence or token list that uses the respective item taken from the list as its argument.

⟨***list***⟩ Same as in `\ForEach`.

3.3.6 The command `\FunctionForEach`

`\FunctionForEach`

The command `\FunctionForEach` works in nearly the same way and has the same syntax as the command `\CommandForEach`. The difference between the two commands is that `\FunctionForEach` encloses the respective list item in braces and supplies it as a single argument to the function. This command has the same syntax as `\CommandForEach`.

3.4 Additional features

The style file for this package contains some features which are not described in the documentation. In particular, some of the commands of the package can be used with optional arguments that are not included in the documentation. These features are still under development, and their usage may change in the future.

4 The implementation

4.1 Initial commands

```
%<*sty>
\ProvidesPackage{forarray}
   [2008/06/20 Version 1.01 -- Using array structures in LaTeX]
\makeatletter
\def\fe@checkifdefined#1{%
\ifx#1\empty
\else
   \expandafter\ifx\csname #1\endcsname\relax
   \else
      \PackageError{forarray}
         {
```

```
         Command #1 is already defined.\MessageBreak
         This command is being used by the package "forarray" %
         and must not be defined when the package is loaded
         }
         {No further immediate help available.}
      \csname fi\endcsname\csname fi\endcsname\@gobblefour
   \fi
   \expandafter\fe@checkifdefined
\fi}
\fe@checkifdefined
   {CommandForEach}{DefineArrayDefault}{DefineArrayVar}
   {DefineArrayVars}{DefineArrayVarTo}{endforeach}
   {ExitForEach}{ExitForEachLevels}{ForArray}{ForEach}
   {ForEachD}{ForEachSublevel}{ForEachX}{FunctionForEach}
   {thislevelcount}{thislevelitem}{thislevelmarker}
   {thislevelnr}{}
\edef\fe@aux@endlinecharrestore{\the\endlinechar}
\endlinechar\m@ne
\newtoks\fe@toks
\newcount\fe@level
\newcount\fe@cnt@i
\newcount\fe@cnt@ii
\newcount\thislevelcount
\newcount\thislevelnr
\fe@level\z@
\chardef\fa@arraylevel\z@
\chardef\fe@toplevel\z@
\chardef\fe@count@abs@\@ne
\chardef\fe@relmax\z@
\chardef\fe@relmax@abs@\z@
\let\fe@item@abs@\empty
\let\fe@first@abs@\empty
\let\fe@last@abs@\empty
\let\fe@empty@abs@\empty
\let\fe@position@abs@\empty
\let\fe@levelrn\empty
```

4.2 Macros for processing lists

4.2.1 User commands

`\ForEachD` The macro `\ForEachD` directly calls `\ForEachD@`. It does not take the list as an argument, so that catcodes can be changed during the execution.

```
\def\ForEachD{\@ifnextchar(\ForEachD@Arg\ForEachD@NoArg}
\def\ForEachD@Arg(#1){\ForEachD@{#1\relax}}
\def\ForEachD@NoArg{\ForEachD@\fe@relmax}
```

`\ForEach` The macro `\ForEach` absorbs the list immediately as its third mandatory argument.

```
\def\ForEach{\@ifnextchar(\ForEach@Arg\ForEach@NoArg}
\def\ForEach@Arg(#1){\ForEach@{#1\relax}}
\def\ForEach@NoArg{\ForEach@\fe@relmax}
\long\def\ForEach@#1#2#3#4{\ForEachD@{#1}{#2}{#3}#4#2\endforeach}
```

`\ForEachX` The macro `\ForEachX` expands its third mandatory argument, which contains the list.

```
\def\ForEachX{\@ifnextchar(\ForEachX@Arg\ForEachX@NoArg}
\def\ForEachX@Arg(#1){\ForEachX@{#1\relax}}
\def\ForEachX@NoArg{\ForEachX@\fe@relmax}
\long\def\ForEachX@#1#2#3#4
   {
   \def\fe@i{\ForEachD@{#1}{#2}{#3}}
   \expandafter\fe@i#4#2\endforeach
   }
```

`\ForEachSublevel` The macro `\ForEachSublevel` expands the current item of a surrounding list or array.

```
\def\ForEachSublevel
   {\@ifnextchar(\ForEachSublevel@Arg\ForEachSublevel@NoArg}
\def\ForEachSublevel@Arg(#1){\ForEachSublevel@{#1\relax}}
\def\ForEachSublevel@NoArg{\ForEachSublevel@\fe@relmax}
\long\def\ForEachSublevel@#1#2#3
   {
   \def\fe@i{\ForEachD@{#1}{#2}{#3}}
   \expandafter\fe@i\thislevelitem#2\endforeach
   }
```

`\ExitForEach` The macro `\ExitForEach` redefines the kernel macro `\fe@next@`⟨*current level*⟩ in order to stop the execution of the current list. The rest of the list is being supplied as an argument to `\fe@next@`⟨*current level*⟩, which just gobbles it.

```
\def\ExitForEach
   {
   \expandafter\let
   \csname fe@next@\romannumeral\fe@level\endcsname
   \fe@ExitForEach@base
   }
\def\fe@ExitForEach@base#1\endforeach{}
```

`\ForEachD@` The macro `\ForEachD@` defines several tokens used for processing a list or a level of an array. If it enters a level higher than any previous one, it calls `\fe@newlevel`, which defines the macros that are being used for processing lists as well as arrays for that level, i.e. the "kernel" macros.

```
\long\def\ForEachD@#1#2#3
   {
   \let\fe@upperlevelrn\fe@levelrn
   \advance\fe@level\@ne\relax
   \expandafter\def\expandafter\fe@levelrn\expandafter
      {\romannumeral\fe@level}
   \ifnum\fe@level>\fe@toplevel
      \expandafter
      \ifx\csname fe@count@abs@\fe@levelrn\endcsname\relax
         \expandafter\newcount
            \csname fe@count@abs@\fe@levelrn\endcsname
      \fi
      \fe@define@position
   \fi
   \csname fe@count@abs@\fe@levelrn\endcsname\z@
   \expandafter\chardef
      \csname fe@first@abs@\fe@levelrn\endcsname\@ne
   \expandafter\chardef
      \csname fe@last@abs@\fe@levelrn\endcsname\z@
   \expandafter\chardef
      \csname fe@relmax@abs@\fe@levelrn\endcsname#1\relax
   \expandafter\fe@define
      \csname fe@relmax@abs@\fe@levelrn\endcsname\fe@levelvars
   \expandafter\long\expandafter\def
      \csname fe@function@\fe@levelrn\endcsname##1{#3}
   \def\fe@emptytest{#2}
   \ifx\fe@emptytest\empty
      \fe@definelevel\empty{}
   \else
      \fe@definelevel#2{\expandafter#2}
   \fi
   \expandafter\expandafter\expandafter\fe@fnsl@
      \expandafter\expandafter
      \csname fe@nextcommandcode@\fe@levelrn\endcsname
      \csname fe@check@\fe@levelrn\endcsname
   }
\def\fe@levelvars{count,item,first,last,position}
\def\fe@definelevel#1#2
   {
   \ifnum\fe@level>\fe@toplevel
      \fe@newlevel#1
      \chardef\fe@toplevel\fe@level
   \fi
   \expandafter\let\csname fe@separator@\fe@levelrn\endcsname#1
   \expandafter\expandafter\expandafter\long
      \expandafter\expandafter\expandafter\def
      \expandafter\expandafter
      \csname fe@getitem@\fe@levelrn\endcsname
      \expandafter##\expandafter1#2\expandafter
      {\csname fe@setitem@\fe@levelrn\endcsname{##1}}
   }
```

4.2.2 Defining the kernel macros

The following macros define the kernel macros `\fe@check@`⟨*current level*⟩, `\fe@setitem@`⟨*current level*⟩ and `\fe@process@`⟨*current level*⟩. `\fe@check@`⟨*current level*⟩ checks the next list item and supplies it to `\fe@setitem@`⟨*current level*⟩, which defines the token that is being processed in turn by `\fe@process@`⟨*current level*⟩, which uses the function supplied by `\ForEachD@`.

`\fe@newlevel` The macro `\fe@newlevel` sets up the tokens that are being used to define the kernel for the new level.

```
\def\fe@newlevel#1
   {
   \fe@DefLevelVar{fe}{\fe@levelrn}
      {
      process,next,item@abs,count@abs,first@abs,
      last@abs,position@abs,function,check,getitem,
      nextcommandcode,aftergroup,
      aftergroup@,firsttoken,space,separator,setitem
      }
   \fe@CollectLevelVar{base}
      {check,getitem,next,nextcommandcode}
   \fe@CollectLevelVar{i}
      {firsttoken,aftergroup,aftergroup@,space,separator}
   \fe@CollectLevelVar{ii}
      {count@abs,first@abs}
   \fe@CollectLevelVar{iii}
      {process,function,item@abs,last@abs,setitem}
   \expandafter\expandafter\expandafter\fe@newlevel@i
      \expandafter\fe@level@base\fe@level@i
   \expandafter\fe@newlevel@ii\fe@level@ii
   \expandafter\expandafter\expandafter\fe@newlevel@iii
      \expandafter\fe@level@base\fe@level@iii
   }
```

`\fe@newlevel@i`

The macro `\fe@newlevel@i` defines two auxilliary macros for the kernel macros and then assembles the first tokens of the macro `\fe@def@check`, which defines the kernel macro `\fe@check@`⟨*current level*⟩.

```
\long\def\fe@newlevel@i#1#2#3#4#5#6#7#8#9
   {
   \def#6{\fe@fnsl#4#8#7}
   \def#7{
      \ifx#9#4
         \ifnum#8=\z@
            \fe@braces@ii#2#5{}
         \else
            \fe@braces@ii#2#5{ }
         \fi
      \else
         \ifnum#8=\z@
            \fe@braces@i#2#5{}
         \else
            \fe@braces@i#2#5{ }
         \fi
      \fi
      \fe@item@check@next
      }
   \def\fe@i##1
      {
      \fe@def@check
         {
         \long\def#5{##1}
            \ifcat\noexpand#4\bgroup
               \ifx#9\empty
                  \expandafter\def\expandafter#3\expandafter
                     {
                     \expandafter#2\expandafter
                        {\expandafter{#5}}
                     }
               \else
                  \let#3#6
               \fi
            \else
```

```
               \expandafter\def\expandafter#3
                  \expandafter{\expandafter#2#5}
            \fi
         }
         {#1##1}#2#3#4
      }
   }
\long\def\fe@braces@i#1#2#3
   {
   \expandafter\def\expandafter\fe@item@check@next\expandafter
      {\expandafter#1\expandafter{#2}#3}
   }
\long\def\fe@braces@ii#1#2#3
   {
   \expandafter\def\expandafter\fe@item@check@next\expandafter
      {\expandafter#1\expandafter{\expandafter{#2}}#3}
   }
```

`\fe@newlevel@ii`

The macro `\fe@newlevel@ii` actually calls the macro `\fe@def@check`, which defines the kernel macro `\fe@check@`⟨*current level*⟩.

```
\long\def\fe@newlevel@ii#1#2{\fe@i{##1}#1#2}
```

`\fe@newlevel@iii`

The macro `\fe@newlevel@iii` calls the macros `\fe@def@setitem` and `\fe@def@process`, which define the kernel macros `\fe@setitem@`⟨*current level*⟩ and `\fe@process@`⟨*current level*⟩.

```
\long\def\fe@newlevel@iii#1#2#3#4#5#6#7#8#9
   {
   \fe@def@setitem#9#7#5
   \fe@def@process#5#3#1#6#4#7#8
   }
```

`\fe@def@check`

The macro `\fe@def@check` defines the kernel macro `\fe@check@`⟨*current level*⟩.

```
\long\def\fe@def@check#1#2#3#4#5#6#7
   {
   \long\def#2
      {
      \ifx#5\endforeach
         \let#4\fe@endlevel
      \else
         \advance#6\@ne
         \thislevelcount#6
         \ifnum#6=\tw@
            \chardef#7\z@
         \fi
         #1
      \fi
      #4
      }
   }
```

`\fe@def@setitem`

The macro `\fe@def@setitem` defines the kernel macro `\fe@setitem@`⟨*current level*⟩.

```
\long\def\fe@def@setitem#1#2#3
   {
```

```
   \long\def#1##1
      {
      \long\def#2{##1}
      \let\thislevelitem#2
      #3
      }
   }
```

`\fe@def@process` The macro `\fe@def@process` defines the kernel macro `\fe@process@`⟨*current level*⟩.

```
\long\def\fe@def@process#1#2#3#4#5#6#7
   {
   \long\def#1
      {
      \ifx#5\endforeach
         \chardef#7\@ne
      \fi
      \def#2{\fe@fnsl@#5#3}
      #4#6
      #2
      }
   }
```

4.2.3 Auxilliary macros

`\fe@endlevel` The macro `\fe@endlevel` is being called after a list or array level has been processed, i.e. when `\fe@check@`⟨*current level*⟩ identifies an `\endforeach` token. It resets the pointers `\thislevelitem` and `\thislevelcount`. It also resets the expansions of some variables, most notably the pointers to the kernel macros, so that they refer to the contents associated with the previous level.

```
\def\fe@endlevel
   {
   \chardef\fe@count@total\thislevelcount
   \advance\fe@level\m@ne
   \expandafter\def\expandafter\fe@levelrn\expandafter
      {\romannumeral\fe@level}
   \expandafter\fe@define
      \csname fe@relmax@abs@\fe@levelrn\endcsname\fe@levelvars
   \expandafter\thislevelcount
      \csname fe@count@abs@\fe@levelrn\endcsname
   \expandafter\let\expandafter\thislevelitem
      \csname fe@item@abs@\fe@levelrn\endcsname
   }
```

`\fe@fnsl` The macro `\fe@fnsl` discards any implicit or explicit space tokens that intervene before a nonspace is scanned, and provides information about the next token and the presence of spaces before this token. It is a modified version of the macro `\futurenonspacelet` from *The TeXBook*.[6]

```
\begingroup\def\\{\global\let\fe@fnsl@stoken= }\\ \endgroup
\def\fe@fnsl#1#2#3
   {
   \def\fe@fnsl@cs{#1}
   \def\fe@fnsl@space{#2}
```

---

[6] See D. Knuth, *The TeXBook*, Reading: Addison-Wesley, 20th printing, rev., p. 376.

```
   \def\fe@fnsl@next@ii{#3}
   \expandafter\chardef\fe@fnsl@space\z@
   \fe@fnsl@stepone
   }
\def\fe@fnsl@stepone
   {\expandafter\futurelet\fe@fnsl@cs\fe@fnsl@steptwo}
\def\fe@fnsl@steptwo
   {
   \expandafter\ifx\fe@fnsl@cs\fe@fnsl@stoken
      \let\fe@fnsl@next@i=\fe@fnsl@stepthree
   \else
      \let\fe@fnsl@next@i\fe@fnsl@next@ii
   \fi
   \fe@fnsl@next@i
   }
\def\fe@fnsl@stepthree
   {
   \expandafter\chardef\fe@fnsl@space\@ne
   \afterassignment\fe@fnsl@stepone\let\fe@fnsl@next@i= %
   }
\def\fe@fnsl@#1#2
   {
   \def\fe@fnsl@cs{#1}
   \def\fe@fnsl@next@ii{#2}
   \fe@fnsl@stepone@
   }
\def\fe@fnsl@stepone@
   {\expandafter\futurelet\fe@fnsl@cs\fe@fnsl@steptwo@}
\def\fe@fnsl@steptwo@
   {
   \expandafter\ifx\fe@fnsl@cs\fe@fnsl@stoken
      \let\fe@fnsl@next@i=\fe@fnsl@stepthree@
   \else
      \let\fe@fnsl@next@i\fe@fnsl@next@ii
   \fi
   \fe@fnsl@next@i
   }
\def\fe@fnsl@stepthree@
   {
   \afterassignment\fe@fnsl@stepone@\let\fe@fnsl@next@i= %
   }
```

`\endforeach` The token `\endforeach` is used as a list delimiter. If executed, it expands to an error message.

```
\def\endforeach
   {
   \PackageError{forarray}
      {Tried to expand an \string\endforeach token. %
      Something is wrong.\MessageBreak
      The level of the current list is: %
         \the\fe@level\MessageBreak
      The content of the current item is: %
         \expandafter\strip@prefix\meaning\thislevelitem
         \MessageBreak
      The position of the item is:
         \the\thislevelcount}
      {No further immediate help available. Sorry.}
   }
```

The following auxilliary macros redefine several variables when entering or exiting a nesting level.

`\fe@@define@process` The macro `\fe@@define@process` processes a list of variables supplied by `\fe@@define`.

```
\def\fe@@define@process#1,
   {
   \def\fe@@define@Item{#1}
   \ifx\fe@@define@Item\empty
      \let\fe@@define@process@next\relax
   \else
      \def\fe@@define@process@next
         {
         \expandafter\expandafter\expandafter\def
         \expandafter\expandafter
         \csname
            \fe@VarMacro @\fe@@define@Item @rel@
            \romannumeral\fe@cnt@ii
         \endcsname
         \expandafter
            {
            \csname
               \fe@VarMacro @\fe@@define@Item @abs@
               \romannumeral\fe@cnt@i
            \endcsname
            }
         \fe@@define@process
         }
   \fi
   \fe@@define@process@next
   }
```

`\fe@@define` The macro `\fe@@define` processes a list of variables supplied by `\fe@define`, iterating through the current nesting levels.

```
\def\fe@@define
   {
   \ifnum\fe@cnt@i>\m@ne
      \ifnum\fe@cnt@ii>\fe@define@max
         \let\fe@@define@next\relax
      \else
         \def\fe@@define@next{
            \expandafter\fe@@define@process\fe@Vars,{},
            \advance\fe@cnt@ii\@ne
            \advance\fe@cnt@i\m@ne
            \fe@@define
            }
      \fi
   \else
      \let\fe@@define@next\relax
   \fi
   \fe@@define@next
   }
```

`\fe@define` The macro `\fe@define` supplies the names of variables that are being redefined when entering or exiting a nesting level.

```
\def\fe@define#1
   {
   \chardef\fe@define@max#1\relax
   \ifnum\fe@define@max>\z@
      \expandafter\fe@define@
   \else
      \expandafter\@gobble
   \fi
   }
\def\fe@define@#1
   {
   \def\fe@VarMacro{fe}
   \let\fe@Vars#1
   \fe@cnt@i\fe@level
   \fe@cnt@ii\@ne
   \fe@@define
   }
```

`\fe@ProcessList` The macro `\fe@ProcessList` processes a list of variables supplied by `\fe@DefLevelVar`.

```
\def\fe@ProcessList#1,
   {
   \def\fe@ProcessList@check{#1}
   \ifx\fe@ProcessList@check\empty
      \let\fe@ProcessList@next\relax
   \else
      \def\fe@ProcessList@next
         {
         \fe@ProcessList@act{#1}
         \fe@ProcessList
         }
   \fi
   \fe@ProcessList@next
   }
```

`\fe@DefLevelVar` The macro `\fe@DefLevelVar` redefines the expansion of the pointers ⟨`fe`|`fa`⟩`@`⟨*pointer name*⟩`@level`.

```
\def\fe@DefLevelVar#1#2#3
   {
   \def\fe@ProcessList@act##1
      {
      \expandafter\expandafter\expandafter\def
      \expandafter\expandafter
      \csname #1@##1@level\endcsname
      \expandafter
      {\csname #1@##1@#2\endcsname}
      }
   \fe@ProcessList#3,{},
   }
```

`\fe@define@position` The macro `\fe@define@position` defines a variable that contains the absolute position of the current item in a list, an array, or a system of nested lists and/or arrays. At this point, it is not being used by the package 'forarray'.

```
\def\fe@define@position
   {
   \ifnum\fe@level=\@ne
      \expandafter\expandafter\expandafter
         \def\expandafter\expandafter
         \csname fe@position@abs@\fe@levelrn\endcsname
         \expandafter
         {
         \csname fe@position@abs@\fe@upperlevelrn\endcsname
         \number\thislevelcount
         }
   \else
      \expandafter\expandafter\expandafter
         \def\expandafter\expandafter
         \csname fe@position@abs@\fe@levelrn\endcsname
         \expandafter
         {
         \csname fe@position@abs@\fe@upperlevelrn\endcsname
         -\number\thislevelcount
         }
   \fi
   }
```

`\fe@AddToTokensX` The macro `\fe@AddToTokensX` adds the contents of a variable to a token list.

```
\def\fe@AddToTokensX#1#2
   {
   \expandafter\expandafter\expandafter#1
   \expandafter\expandafter\expandafter
   {\expandafter\the\expandafter#1#2}
   }
```

`\fe@CollectLevelVar@` The macro `\fe@CollectLevelVar@` takes a list of variable names and assembles a corresponding list of pointer variables ⟨`fe`|`fa`⟩@⟨*pointer name*⟩@level.

```
\def\fe@CollectLevelVar@#1#2#3
   {
   \expandafter\def\csname #1@level@#2\endcsname{}
   \def\fe@ProcessList@act##1{
      \expandafter\fe@AddToTokensX\expandafter
      \fe@toks\csname #1@##1@level\endcsname
      }
   \fe@toks{}
   \fe@ProcessList#3,{},
   \expandafter\expandafter\expandafter\def
      \expandafter\expandafter
      \csname #1@level@#2\endcsname\expandafter{\the\fe@toks}
   }
```

`\fe@CollectLevelVar` The macro `\fe@CollectLevelVar` calls `\fe@CollectLevelVar@`.

```
\def\fe@CollectLevelVar{\fe@CollectLevelVar@{fe}}
```

4.3 Macros for processing arrays

4.3.1 User commands

This section contains the macros `\ForArray` and `\ExitForEachLevels`, as well as some macros associated with `\ForArray` that directly read information from the token stream following the call to `\ForArray`.

`\ForArray` When being called, the macro `\ForArray` checks whether a limit for the creation of pointers to levels relative to the current list or array level has been supplied, and whether an optional argument containing a separator for the list of separators is present.

```
\def\ForArray
   {
   \fe@aux@advancechardef\fa@arraylevel\@ne
   \expandafter\def\expandafter\fa@arraylevelrn\expandafter
      {\romannumeral\fa@arraylevel}
   \@ifnextchar(\fa@FA@WRelMax\fa@FA@WoRelMax
   }
\def\fa@FA@WRelMax(#1)
   {
   \expandafter\chardef
      \csname fa@relmax@\fa@arraylevelrn\endcsname#1\relax
   \ForArray@
   }
\def\fa@FA@WoRelMax
   {
   \expandafter\chardef
      \csname fa@relmax@\fa@arraylevelrn\endcsname\fe@relmax
   \ForArray@
   }
\def\ForArray@
   {\@ifnextchar[\fa@FA@WSepListSep\fa@FA@WoSepListSep}
\def\fa@FA@WSepListSep[#1]{\fa@FA@SepList#1}
\def\fa@FA@WoSepListSep{\fa@FA@SepList{}}
```

`\fa@FA@SepList` The macro `\fa@FA@SepList` reads the list of separators and defines some variables used for processing the array.

```
\def\fe@aux@advancechardef#1#2
   {
   \count@#1
   \advance\count@#2
   \chardef#1\count@
   }
\def\fa@FA@SepList#1#2
   {
   \fe@DefLevelVar{fa}{\fa@arraylevelrn}
      {
      separatorcount,oldcatcode,baselevel,level,olddef,
      array,restore,next,separator,orientation,nextlevel,
      oldnextlevel,oldlowernextlevel
      }
   \expandafter\chardef
      \csname fa@level@\fa@arraylevelrn\endcsname\z@
   \ForEach
```

```
      {#1}
      {
         \expandafter\expandafter\expandafter\def
            \expandafter\expandafter
            \csname
               fa@separator@\fa@arraylevelrn @
               \romannumeral\thislevelcount
            \endcsname
            \expandafter
            {\thislevelitem}
      }
      {#2}
   \expandafter\chardef\fa@separatorcount@level\fe@count@total
   \fa@FA@MarkerList
   }
```

`\fa@FA@MarkerList` The macro `\fa@FA@MarkerList` checks for an optional argument containing a list of markers, possibly with an additional optional argument that contains a separator for this list.

```
\def\fa@FA@MarkerList
   {\@ifnextchar[\fa@FA@WMarkerList\fa@FA@SublevelToken}
\def\fa@FA@WMarkerList[
   {\@ifnextchar[\fa@FA@WMarkerListSep\fa@FA@WoMarkerListSep}
\def\fa@FA@WMarkerListSep[#1]#2]{\fa@FA@MarkerList@{#1}{#2}}
\def\fa@FA@WoMarkerListSep#1]{\fa@FA@MarkerList@{}{#1}}
```

`\fa@FA@MarkerList@` The macro `\fa@FA@MarkerList@` reads the list of markers.

```
\def\fa@FA@MarkerList@#1#2
   {
   \ForEach
      {#1}
      {
         \expandafter\expandafter\expandafter\def
            \expandafter\expandafter
            \csname
               fa@orientation@\fa@arraylevelrn @
               \romannumeral\thislevelcount
            \endcsname
            \expandafter
            {\thislevelitem}
      }
      {#2}
   \fa@FA@SublevelToken
   }
```

`\fa@FA@SublevelToken` The macro `\fa@FA@SublevelToken` reads the token that is used to access sublevels and calls `\fa@SublevelToken`.

```
\def\fa@FA@SublevelToken#1
   {
   \expandafter\fa@SublevelToken\fa@array@level#1
   \fa@FA@Process
   }
```

`\fa@FA@Process` The macro `\fa@FA@Process` reads the functions applied to the respective levels of the array and processes the content of the array.

```
\long\def\fa@FA@Process#1#2#3
   {
   \ForEach
      {#1}
      {
         \expandafter\expandafter\expandafter\def
            \expandafter\expandafter
            \csname
               fa@function@\fa@arraylevelrn @
               \romannumeral\thislevelcount
            \endcsname
            \expandafter
            {\thislevelitem}
      }
      {#2}
   \long\def\thislevelitem{#3}
   \fe@cnt@i\fe@level
   \advance\fe@cnt@i\@ne
   \expandafter\chardef\fa@baselevel@level\fe@cnt@i
   \expandafter\def\fa@array@level
      {
      \ifnum\fa@level@level=\fa@separatorcount@level\relax
         \expandafter\def\fa@next@level{\thislevelitem}
      \else
         \expandafter\let\fa@next@level\fa@next@level@

      \fi
      \fa@next@level
      }
   \fa@array@level
   \fa@restore@level
   \fe@aux@advancechardef\fa@arraylevel\m@ne
   \expandafter\def\expandafter\fa@arraylevelrn\expandafter
      {\romannumeral\fa@arraylevel}
   }
\def\fa@next@level@
   {
   \fa@SetLevelVars\tw@
   \expandafter\expandafter\expandafter\def
      \expandafter\expandafter\expandafter\fa@i
      \expandafter\expandafter\expandafter
      {
      \csname
      fa@separator@\fa@arraylevelrn @
      \romannumeral\fa@level@level
      \endcsname
      }
   \expandafter\expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter
      \ForEachSublevel@
      \expandafter\expandafter\expandafter\expandafter
      \expandafter\expandafter
      \csname
```

```
         fa@relmax@\fa@arraylevelrn
      \endcsname
      \expandafter
      \fa@i
      \expandafter
      {
      \csname
         fa@function@\fa@arraylevelrn @
         \romannumeral\fa@level@level
      \endcsname
      }
   \fa@SetLevelVars\@ne
   }
\def\fa@SetLevelVars#1
   {
   \fe@cnt@i\fe@level
   \advance\fe@cnt@i-\fa@baselevel@level
   \advance\fe@cnt@i#1
   \expandafter\chardef\fa@level@level\fe@cnt@i
   \thislevelnr\fa@level@level
   \expandafter\let\expandafter\thislevelmarker
      \csname
         fa@orientation@\fa@arraylevelrn @
         \romannumeral\fa@level@level
      \endcsname
   }
```

`\ExitForEachLevels` The macro `\ExitForEachLevels` redefines the kernel macros `\fe@next@`⟨*start level*⟩ ... `\fe@next@`⟨*start level plus number of levels*⟩ of the current position of the array that is being processed. It essentially works in the same way as `\ExitForEach`.

```
\def\ExitForEachLevels#1#2
   {
   \fe@cnt@i\fe@level
   \fe@cnt@ii\z@
   \advance\fe@cnt@i\@ne
   \advance\fe@cnt@i-#1\relax
   \def\fe@exitforeach
      {
      \ifnum\fe@cnt@ii<#2\relax
         \def\fe@exitforeach@next
            {
            \expandafter\let
               \csname fe@next@\romannumeral\fe@level\endcsname
               \fe@ExitForEach@base
            \advance\fe@cnt@i\m@ne
            \advance\fe@cnt@ii\@ne
            }
      \else
         \let\fe@exitforeach@next\relax
      \fi
      \fe@exitforeach@next
      }
   \fe@exitforeach
   }
```

4.3.2 Auxilliary macros

`\fa@SublevelToken` The macro `\fa@SublevelToken` assigns a pointer to the control sequence or active character that is being used to access sublevels. It also defines how the expansion of this token is being reset after exiting a level of an array or nested list. The code at the end of this macro is taken from the inputenc package.[7]

```
\def\fa@SublevelToken#1#2
   {
   \expandafter\if\noexpand#2\relax
      \expandafter\let\fa@olddef@level#2
      \def#2{#1}
      \expandafter\def\fa@restore@level
         {\expandafter\let#2\fa@olddef@level}
   \else
      \chardef\fa@oldcatcode@level\catcode‘#2\relax
      \ifnum\fa@oldcatcode@level=\active
         \expandafter\let\fa@olddef@level#2
      \else
         \catcode‘#2\active
      \fi
      \expandafter\def\fa@restore@level
         {
         \ifnum\fa@oldcatcode@level=\active
            \expandafter\expandafter\expandafter
            \fa@SublevelToken
            \expandafter\expandafter\expandafter
            {\fa@olddef@level}{#2}
         \else
            \catcode‘#2\fa@oldcatcode@level
         \fi
         }
         \bgroup
            \uccode‘\~‘#2\relax
            \uppercase{
         \egroup
               \def~{#1}
               }
   \fi
   }
```

4.4 Macros for defining variables

`\DefineArrayVar` The macro `\DefineArrayVar` first collects the items from the content list in an array of numbered variables and then assigns the content of these variables to the new variables.

```
\def\DefineArrayVar#1#2#3#4#5#6
   {
   \ForEach{#5}
      {
      \expandafter\expandafter\expandafter
         \def\expandafter\expandafter
         \csname
            fe@item@nr@\number\thislevelcount
```

---

[7] See A. Jeffrey and F. Mittelbach, inputenc.sty, v1.1b, May 5th, 2006.

```
         \endcsname
         \expandafter{\thislevelitem}
      }
      {#6}
   \ForEach{#3}
      {
      \expandafter\ifx
         \csname
            fe@item@nr@\number\thislevelcount
         \endcsname
         \relax
         \fe@DefineArrayVar@Warning{#4}{#5}
         \ExitForEach
      \else
         \expandafter\expandafter\expandafter\def
         \expandafter\expandafter\expandafter
         \fa@ArrayVarContent
         \expandafter\expandafter\expandafter
            {
            \csname
               fe@item@nr@\number\thislevelcount
            \endcsname
            }
         \expandafter\ifx
            \csname
               fe@item@nr@\number\thislevelcount
            \endcsname
            \empty
            \typeout
               {Content of   \expandafter\string
               \csname #1#2\thislevelitem\endcsname\space
               is set to nothing.}
         \else
            \typeout
               {Content of   \expandafter\string
               \csname #1#2\thislevelitem\endcsname\space
               is set to %
               \expandafter\strip@prefix
               \meaning\fa@ArrayVarContent.}
         \fi
         \expandafter\expandafter\expandafter\def
            \expandafter\expandafter
            \csname #1#2\thislevelitem\endcsname\expandafter
            {\fa@ArrayVarContent}
      \fi
      }
      {#4}
   }
\def\fe@DefineArrayVar@Warning#1#2
   {
   \PackageWarning
      {fornext}
      {
      No more items available while %
         defining pointers!\MessageBreak
      Pointers: #1\MessageBreak
      Items:\space\space\space #2
      }
   }
```

`\DefineArrayVars` The macro `\DefineArrayVars` first defines a function that passes information read from its parameters to `\DefineArrayVar`, and then calls this function within a `\ForEach` loop.

```
\def\DefineArrayVars#1#2#3#4#5#6#7
   {
   \typeout{}\typeout{Defining Array Variables...}
   \def\fe@DefineArrayVar@##1#3##2#3##3#2
      {
      \typeout{-- Initializing new variable array: ##1}
      \DefineArrayVar
         {##1}{#5}{#1}{##3}{#4}{##2}
      }
   \ForEach
      {#2}
      {\expandafter\fe@DefineArrayVar@\thislevelitem#3#6#2}
      {#7}
   }
```

`\DefineArrayDefault` The macro `\DefineArrayDefault` assigns an `\ifx` … `else` … `\fi` structure to the pointers supplied with its last argument.

```
\def\DefineArrayDefault#1#2#3#4#5
   {
   \ForEach
      {#1}
      {
      \expandafter\edef\csname\thislevelitem\endcsname
         {
         \noexpand\expandafter\noexpand\ifx
         \noexpand\csname
            \thislevelitem #2\noexpand#3
         \noexpand\endcsname
         \noexpand\relax
            \noexpand\csname
               \thislevelitem #2#4
            \noexpand\endcsname
         \noexpand\else
            \noexpand\csname
               \thislevelitem #2\noexpand#3
            \noexpand\endcsname
         \noexpand\fi
         }
      }
      {#5}
   }
```

`\DefineArrayVarTo` The macro `\DefineArrayVarTo` assigns the same value to the variables supplied with its last argument.

```
\def\DefineArrayVarTo#1#2#3#4#5
   {
   \ForEach
```

```
      {#1}
      {\expandafter\expandafter\expandafter\def\expandafter
         \csname #3#2\thislevelitem\endcsname{#4}}
      {#5}
   }
```

`\CommandForEach`

The macro `\CommandForEach` places the items of the list immediately after the token list that is being supplied as the command.

```
\def\CommandForEach#1#2#3
   {\ForEach#1{\expandafter#2\thislevelitem}{#3}}
```

`\FunctionForEach`

The macro `\FunctionForEach` expands the items of the list inside braces and immediately places the token group after the token list that is being supplied as the function.

```
\def\FunctionForEach#1#2#3
   {\ForEach#1{\expandafter#2\expandafter{\thislevelitem}}{#3}}
```

4.5 Final commands

```
\endlinechar\fe@aux@endlinecharrestore\relax
\makeatother
%</sty>
```

5 Test page

The file `forarray-test.tex` contains the following code that generates a test page for the package forarray.

```
%<*test>
\documentclass[10pt,a4paper]{article}
\usepackage[latin1]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[margin=2cm]{geometry}
\usepackage[dvips, pdfborder={0 0 0}, pdfstartview={FitH},
   pdfpagelayout={OneColumn}, bookmarks=false, pdfnewwindow,
   unicode=true]{hyperref}
\urlstyle{same}
\usepackage{examplep}
\usepackage{forarray}
\author
   {
   Christian Schr\"{o}ppel%
   \footnote%
      {
      Please send any comments or suggestions to %
      \protect\href{mailto:christian@schroeppel.com}%
{\mbox{\fontfamily{cmss}\selectfont christian@schroeppel.com}}.%
      }
   }
\title
   {
   Test page for the %
      {\fontfamily{cmss}\selectfont forarray} %
      package\\ [.5ex]\Large Version 1.01 (2008/06/20)
   }
\def\ShowExample{
   \PexaShowBoth{
      yalign=b,
      allowbreak=yes,
      srcstyle=leftnumcol,
      }
   }
\arrayrulewidth=0pt
\begin{document}
\errorcontextlines=20\relax
\maketitle
\thispagestyle{empty}
\small
\section{Test of \texttt{ForEach} (Simple List)}
\begin{WBoth}
\begin{itemize}
\ForEach{,}
   {\item Item No.\ %
      \the\thislevelcount\ is:
      ``\thislevelitem''\\{\footnotesize
      \meaning\thislevelitem}}
   {
   {Hello, World!},  Sec{ond},
      {Thi}rd,{\bf Last}   item%
   }
\end{itemize}
\end{WBoth}
\ShowExample
\section{Test of \texttt{ForEach} (Nested)}
\begin{WBoth}
\begin{itemize}
\ForEach{;}
   {\item[\the\thislevelcount)]
      \raggedright\thislevelitem
   \begin{itemize}
   \ForEachX{,}
      {\item Item No.\ %
      \the\thislevelcount\ %
      is: \thislevelitem}
      {\thislevelitem}
   \end{itemize}}
   {$\alpha$,$\beta$,$\gamma$;
   {\Large A Large Item},
   Transparency \it test,
   Ends \rm here.}
\end{itemize}
\end{WBoth}
\ShowExample
\section{Test of \texttt{ForArray}}
\makeatletter
\begin{WBoth}
\parindent=0pt
\def\MyArray{\ForArray(3){;,}{*}{|}}
\MyArray
   {*\par\vskip 3ex|\parbox{8em}{*}}
```

```
  {
  A,B,C;
  \textit{A nested array:}\par
  \MyArray{[*]\par|(*)}{1,2;3,4},b,c;
  $\alpha$,$\beta$,$\gamma$
  }
\end{WBoth}
\ShowExample
\end{document}
%</test>
```

6 Copyright Notice

The package forarray is © 2008 by Christian Schröppel. It may be distributed and/or modified under the conditions of the LaTeX Project Public License (LPPL), version 1.3c. This licence allows you to distribute unmodified copies of the package, as long as you include all components of the package in your distribution. It also allows modification of the package under certain conditions. Please read the licence if you intend to modify any of the contents of this package.

If any later version of the LPPL replaces this version, the package may be distributed and/or modified under the conditions of the current version of the LPPL at that time. The latest version of the LPPL is available at www.latex-project.org/lppl.txt.

The Author of the package is Christian Schröppel. You can contact the author at christian@schroeppel.com.

This package has the LPPL maintenance status "maintained". The Current Maintainer is Christian Schröppel.

The package forarray consists of the master file forarray.dtm, the file README.txt, and the derived files forarray.dtx, forarray.sty, forarray.pdf, forarray-test.tex, and forarray-test.pdf.

The installation script forarray and the documentation style file forarray.dts are not part of the package forarray. Please note, however, that the provisions in the section "No warranty" of the LaTeX Project Public License (LPPL), version 1.3c, exempting the author and other parties from liability with regard to the work, apply to the contents of the package as well as to these files.

Index

Bold numbers refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

*** END OF INDEX ***