

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.

The package `forarray` is © 2008 by Christian Schröppel, and distributed under the terms of the \LaTeX Project Public License (LPPL), version 1.3c.

Contents

1	Introduction	2
2	Files	3
3	Usage	3
3.1	Processing lists	3
3.1.1	The command <code>\ForEach</code>	3
3.1.2	The command <code>\ForEachX</code>	4
3.1.3	The command <code>\ForEachSublevel</code>	4
3.1.4	The command <code>\ForEachD</code>	5
3.1.5	The command <code>\ExitForEach</code>	5
3.2	Processing arrays	6
3.2.1	The command <code>\ForArray</code>	6
3.2.2	The command <code>\ExitForEachLevels</code>	7
3.3	Defining variables	8
3.3.1	The command <code>\DefineArrayVar</code>	8
3.3.2	The command <code>\DefineArrayVars</code>	9
3.3.3	The command <code>\DefineArrayDefault</code>	10
3.3.4	The command <code>\DefineArrayVarTo</code>	11
3.3.5	The command <code>\CommandForEach</code>	12
3.3.6	The command <code>\FunctionForEach</code>	12
3.4	Additional features	12

*Please send any comments or suggestions to christian@schroepel.com.

4 The implementation	12
4.1 Initial commands	12
4.2 Macros for processing lists	13
4.2.1 User commands	13
4.2.2 Defining the kernel macros	15
4.2.3 Auxilliary macros	18
4.3 Macros for processing arrays	23
4.3.1 User commands	23
4.3.2 Auxilliary macros	27
4.4 Macros for defining variables	27
4.5 Final commands	30
5 Test page	30
6 Copyright Notice	32
Index	33

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.

¹There 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 L^AT_EX 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 L^AT_EX 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 L^AT_EX installation will find it. This will often be `<your local texmf directory>/tex/latex/forarray`. Please consult the manuals for your L^AT_EX system for further information. In many cases, it will be possible to use a package manager provided with your L^AT_EX 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:

$$\text{\ForEach}\langle separator \rangle \langle function \rangle \langle list \rangle$$

<separator> This can be any character or control sequence. Often, a comma or a semicolon will be used as the separator.

<code>\thislevelitem</code>	<code>\function</code>
<code>\thislevelcount</code>	

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`

<pre> 1 \begin{itemize} 2 \ForEach 3 {,} 4 {\item This is item No.\ % 5 \the\thislevelcount.\ \ 6 It contains: ‘‘\thislevelitem’’ .} 7 {A \textbf{bold} word, 8 \textit{Some more words, written 9 in italics}, 10 \multiply\thislevelcount\thislevelcount 11 the number \the\thislevelcount, 12 Some more text} 13 \end{itemize} </pre>	<ul style="list-style-type: none"> • This is item No. 1. It contains: “A bold word”. • This is item No. 2. It contains: “<i>Some more words, written in italics</i>”. • 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:

$$\backslash\text{ForEachX}\{\langle separator \rangle\}\{\langle function \rangle\}\{\langle list token \rangle\}$$

- `<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:

$$\backslash\text{ForEachSublevel}\{\langle separator \rangle\}\{\langle function \rangle\}$$

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

1 \begin{enumerate}	
2 \ForEach	1. This is item No. 1.
3 {;}	(a) This is a nested item.
4 {	(b) Another nested item.
5 \item This is item No.\ %	
6 \the\thislevelcount.	
7 \begin{enumerate}	2. This is item No. 2.
8 \ForEachSublevel	(a) This item is, well, nested.
9 {,}	(b) A final item.
10 {\item \thislevelitem.}	
11 \end{enumerate}	
12 }	
13 {This is a nested item,	
14 Another nested item;	
15 {This item is, well, nested},	
16 A final item}	
17 \end{enumerate}	

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`.

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

Figure 3: Changing category codes within a list

```

1 \ForEachD                    * Huh? What is this?
2 {;}
3 {\thislevelitem\par\bigskip}    * I still have no clue!
4 * Huh? What is this?;
5 \catcode'*\active * I still have no clue!; Now I see ... This is a star!
6 \newcommand{*}{\textbf{star}}
7 Now I see \textellipsis\ This is a *!;      I remember, this is a star.
8 I remember, this is a *.;\endforeach

```

Figure 4: Usage of `\ExitForEach`

```

1 \newcommand{\LeaveThisPlace}    - This item says: "Please continue."
2 {Leave this place!}
3 \ForEach
4 {;}
5 {-- This item says:           - This item says: "Please continue."
6 '\thislevelitem'\par\medskip  Continuing to the next item ...
7 \ifx\thislevelitem\LeaveThisPlace
8 \ExitForEach Oops, I have to quit.\par - This item says: "Leave this place!"
9 \else
10 Continuing to the next item     Oops, I have to quit.
11 \textellipsis\par\bigskip
12 \fi}
13 {Please continue.;Please continue.;
14 Leave this place!;Where has he gone?}

```

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:

$$\text{\ForArray}\{\langle separator list \rangle\}[[\langle marker list separator \rangle]\langle marker list \rangle] \\ \{\langle sublevel token \rangle\}\{\langle function list separator \rangle\} \\ \{\langle function list \rangle\}\{\langle array \rangle\}$$

$\langle separator list \rangle$ 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 `\{ $\langle separator list \rangle$ \}`.

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

$\langle marker list \rangle$ 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*⟩.

⟨*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.

`\thislevelnr`

⟨*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 `\ForEach`.

Figure 5: Usage of `\ForEach`

(a) Basic usage

1	<code>\ForEach{;,}{*}{ }</code>	$[(A)(B)]$
2	<code>{[*]\par (*)}</code>	$[(C)(D)]$
3	<code>{A,B;C,D}</code>	

(b) Nested arrays

1	<code>\ForEach{;,}{*}{ }</code>	<table border="1"><tr><td>A</td><td>B</td></tr></table>	A	B		
A	B					
2	<code>{\framebox{*}\par\smallskip </code>					
3	<code>\framebox{\parbox[t]{3em}</code>	<table border="1"><tr><td>(a)(b)</td><td>D</td></tr><tr><td>(c)(d)</td><td></td></tr></table>	(a)(b)	D	(c)(d)	
(a)(b)	D					
(c)(d)						
4	<code>{\centering*}\ }</code>					
5	<code>{A,B;%</code>					
6	<code>\ForEach{;,}{*}{ }</code>					
7	<code>{\centering*\par (*)}</code>					
8	<code>{a,b;c,d},D}</code>					

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:

$$\backslash\text{ExitForEachLevels}\{\langle\textit{Start level}\rangle\}\{\langle\textit{Number of levels}\rangle\}$$

⁴The ⟨*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`

1	<code>\makeatletter</code>	The capital city of Brazil is Brasília.
2	<code>\fontfamily{upl}\selectfont</code>	The capital city of Japan is Tōkyō.
3		
4	<code>\DefineArrayVar{Capital}{@}</code>	The capital city of the Republic of Korea
5	<code>{,}{Brazil,Japan,South Korea,Viet Nam}</code>	is Sōul.
6	<code>{,}{Bras\`{i}lia,T\={o}ky\={o},</code>	
7	<code>S\u{o}ul,H\^{a} N\^{o}i}</code>	The capital city of Việt Nam is Hà Nội.
8		
9	<code>\DefineArrayVar{Name}{@}</code>	
10	<code>{,}{Brazil,Japan,South Korea,Viet Nam}</code>	
11	<code>{,}{Brazil,Japan,the Republic of Korea,</code>	
12	<code>Vi\d\ecircumflex t Nam}</code>	
13		
14	<code>\newcommand{\CapitalCity}[1]</code>	
15	<code>{The capital city of</code>	
16	<code>\csname Name@#1\endcsname\ is</code>	
17	<code>\csname Capital@#1\endcsname.}</code>	
18		
19	<code>\ForEach{,}</code>	
20	<code>{\CapitalCity{\thislevelitem}</code>	
21	<code>\par\medskip}</code>	
22	<code>{Brazil,Japan,South Korea,Viet Nam}</code>	

3.3.2 The command `\DefineArrayVars`

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

$$\begin{aligned} \text{\DefineArrayVars}\{ & \langle variable\ list\ separator \rangle \} \\ & \{ \langle array\ definitions\ separator \rangle \} \\ & \{ \langle array\ name/content\ separator \rangle \} \\ & \{ \langle content\ list\ separator \rangle \} \\ & \{ \langle variable\ name\ separator \rangle \} \\ & \{ \langle variable\ list \rangle \} \\ & \{ \langle content\ definition\ list \rangle \} \end{aligned}$$

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`

1	<code>\makeatletter</code>	The capital city of Brazil is Brasília.
2	<code>\fontfamily{upl}\selectfont</code>	The capital city of Japan is Tōkyō.
3		
4	<code>\DefineArrayVars{,}{/}{:}{;}{@}</code>	The capital city of the Republic of Korea is
5	<code>{Brazil,Japan,South Korea,Viet Nam}</code>	Sōul.
6	<code>{Capital:</code>	
7	<code>Bras\'{i}lia;T\={o}ky\={o};</code>	The capital city of Việt Nam is Hà Nội.
8	<code>S\u{o}ul;H\^{a} N\^{o}i/</code>	
9	<code>Name:</code>	
10	<code>Brazil;Japan;the Republic of Korea;</code>	
11	<code>Vi\d\ecircumflex t Nam}</code>	
12		
13	<code>\newcommand{\CapitalCity}[1]</code>	
14	<code>{The capital city of</code>	
15	<code>\csname Name@#1\endcsname\ is</code>	
16	<code>\csname Capital@#1\endcsname.}</code>	
17		
18	<code>\ForEach{,}</code>	
19	<code>{\CapitalCity{\thislevelitem}</code>	
20	<code>\par\medskip}</code>	
21	<code>{Brazil,Japan,South Korea,Viet Nam}</code>	

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 according 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:

$$\begin{aligned} \text{\DefineArrayDefault}\{ & \langle \text{array list separator} \rangle \} \\ & \{ \langle \text{variable name separator} \rangle \} \\ & \{ \langle \text{index variable} \rangle \} \\ & \{ \langle \text{default variable} \rangle \} \\ & \{ \langle \text{array list} \rangle \} \end{aligned}$$

⁵Be 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 is 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`

1	<code>\makeatletter</code>	– You’ve asked for the capital city
2	<code>\fontfamily{upl}\selectfont</code>	of <i>Brazil</i> .
3		
4	<code>\DefineArrayVars{,}{/}{:}{;}{@}</code>	The capital city of Brazil is
5	<code>{Brazil,Japan,South Korea,Viet Nam,Unknown}</code>	Brasília .
6	<code>{Capital:</code>	
7	<code>Bras\’{i}lia;T\={o}ky\={o};</code>	– You’ve asked for the capital city
8	<code>S\u{o}ul;H^{a} N^{o}i;unknown/</code>	of <i>South Korea</i> .
9	<code>Name:</code>	
10	<code>Brazil;Japan;the Republic of Korea;</code>	The capital city of the Republic of
11	<code>Vi\d\ecircumflex t Nam;this country}</code>	Korea is Söul .
12		
13	<code>\DefineArrayDefault{,}{@}{\country}%</code>	– You’ve asked for the capital city
14	<code>{Unknown}{Capital,Name}</code>	of a <i>strange country</i> .
15		
16	<code>\newcommand{\WhatIsTheCapitalCity}</code>	The capital city of this country is
17	<code>{-- You’ve asked for the capital city of</code>	unknown .
18	<code>\textit{\country}.\par\medskip</code>	
19	<code>The capital city of \Name\ is</code>	
20	<code>\textbf{\Capital}.\par\bigskip}</code>	
21		
22	<code>\newcommand{\country}{Brazil}</code>	
23	<code>\WhatIsTheCapitalCity</code>	
24	<code>\renewcommand{\country}{South Korea}</code>	
25	<code>\WhatIsTheCapitalCity</code>	
26	<code>\renewcommand{\country}{a strange country}</code>	
27	<code>\WhatIsTheCapitalCity</code>	

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⟩}
```

\langle *variable list separator* \rangle Same as in `\DefineArrayVar`.
 \langle *variable name separator* \rangle Same as in `\DefineArrayVar`.
 \langle *array name* \rangle Same as in `\DefineArrayVar`.
 \langle *content* \rangle The content that is being assigned to the variables.
 \langle *variable list* \rangle 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:

$$\backslash\text{CommandForEach}\{\langle\textit{variable list separator}\rangle\}\{\langle\textit{command}\rangle\}\{\langle\textit{variable list}\rangle\}$$

\langle *list separator* \rangle Same as in `\ForEach`.
 \langle *command* \rangle This is the control sequence or token list that uses the respective item taken from the list as its argument.
 \langle *list* \rangle 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

```

1 %<*sty>
2 \ProvidesPackage{forarray}
3   [2008/06/20 Version 1.01 -- Using array structures in LaTeX]
4 \makeatletter
5 \def\fe@checkifdefined#1{%
6 \ifx#1\empty
7 \else
8   \expandafter\ifx\csname #1\endcsname\relax
9   \else
10    \PackageError{forarray}
11    {

```

```

12         Command #1 is already defined.\MessageBreak
13         This command is being used by the package "forarray" %
14         and must not be defined when the package is loaded
15     }
16     {No further immediate help available.}
17     \csname fi\endcsname\csname fi\endcsname\@gobblefour
18 \fi
19 \expandafter\fe@checkifdefined
20 \fi}
21 \fe@checkifdefined
22     {CommandForEach}{DefineArrayDefault}{DefineArrayVar}
23     {DefineArrayVars}{DefineArrayVarTo}{endforeach}
24     {ExitForEach}{ExitForEachLevels}{ForArray}{ForEach}
25     {ForEachD}{ForEachSublevel}{ForEachX}{FunctionForEach}
26     {thislevelcount}{thislevelitem}{thislevelmarker}
27     {thislevelnr}{}
28 \edef\fe@aux@endlinecharrestore{\the\endlinechar}
29 \endlinechar\m@ne
30 \newtoks\fe@toks
31 \newcount\fe@level
32 \newcount\fe@cnt@i
33 \newcount\fe@cnt@ii
34 \newcount\thislevelcount
35 \newcount\thislevelnr
36 \fe@level\z@
37 \chardef\fa@arraylevel\z@
38 \chardef\fe@toplevel\z@
39 \chardef\fe@count@abs@\@ne
40 \chardef\fe@relmax\z@
41 \chardef\fe@relmax@abs@\z@
42 \let\fe@item@abs@\empty
43 \let\fe@first@abs@\empty
44 \let\fe@last@abs@\empty
45 \let\fe@empty@abs@\empty
46 \let\fe@position@abs@\empty
47 \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.

```

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

```

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

```

51 \def\ForEach{\@ifnextchar(\ForEach@Arg\ForEach@NoArg}
52 \def\ForEach@Arg(#1){\ForEach@{#1\relax}}
53 \def\ForEach@NoArg{\ForEach@\fe@relmax}
54 \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.

```

55 \def\ForEachX{\@ifnextchar(\ForEachX@Arg\ForEachX@NoArg}
56 \def\ForEachX@Arg(#1){\ForEachX@{#1\relax}}
57 \def\ForEachX@NoArg{\ForEachX@\fe@relmax}
58 \long\def\ForEachX@#1#2#3#4
59 {
60 \def\fe@i{\ForEachD@{#1}{#2}{#3}}
61 \expandafter\fe@i#4#2\endforeach
62 }

```

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

```

63 \def\ForEachSublevel
64 {\@ifnextchar(\ForEachSublevel@Arg\ForEachSublevel@NoArg}
65 \def\ForEachSublevel@Arg(#1){\ForEachSublevel@{#1\relax}}
66 \def\ForEachSublevel@NoArg{\ForEachSublevel@\fe@relmax}
67 \long\def\ForEachSublevel@#1#2#3
68 {
69 \def\fe@i{\ForEachD@{#1}{#2}{#3}}
70 \expandafter\fe@i\thislevelitem#2\endforeach
71 }

```

`\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.

```

72 \def\ExitForEach
73 {
74 \expandafter\let
75 \csname fe@next@\romannumeral\fe@level\endcsname
76 \fe@ExitForEach@base
77 }
78 \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.

```

79 \long\def\ForEachD@#1#2#3
80 {
81 \let\fe@upperlevelrn\fe@levelrn
82 \advance\fe@level\@ne\relax
83 \expandafter\def\expandafter\fe@levelrn\expandafter
84 {\romannumeral\fe@level}
85 \ifnum\fe@level>\fe@toplevel
86 \expandafter
87 \ifx\csname fe@count@abs@\fe@levelrn\endcsname\relax
88 \expandafter\newcount
89 \csname fe@count@abs@\fe@levelrn\endcsname
90 \fi
91 \fe@define@position
92 \fi
93 \csname fe@count@abs@\fe@levelrn\endcsname\z@

```

```

94     \expandafter\chardef
95         \csname fe@first@abs@\fe@levelrn\endcsname\@ne
96     \expandafter\chardef
97         \csname fe@last@abs@\fe@levelrn\endcsname\z@
98     \expandafter\chardef
99         \csname fe@relmax@abs@\fe@levelrn\endcsname#1\relax
100    \expandafter\fe@define
101        \csname fe@relmax@abs@\fe@levelrn\endcsname\fe@levelvars
102    \expandafter\long\expandafter\def
103        \csname fe@function@\fe@levelrn\endcsname##1{#3}
104    \def\fe@emptytest{#2}
105    \ifx\fe@emptytest\empty
106        \fe@definelevel\empty{}
107    \else
108        \fe@definelevel#2{\expandafter#2}
109    \fi
110    \expandafter\expandafter\expandafter\fe@fnsl@
111        \expandafter\expandafter
112        \csname fe@nextcommandcode@\fe@levelrn\endcsname
113        \csname fe@check@\fe@levelrn\endcsname
114    }
115    \def\fe@levelvars{count,item,first,last,position}
116    \def\fe@definelevel#1#2
117    {
118        \ifnum\fe@level>\fe@toplevel
119            \fe@newlevel#1
120            \chardef\fe@toplevel\fe@level
121        \fi
122        \expandafter\let\csname fe@separator@\fe@levelrn\endcsname#1
123        \expandafter\expandafter\expandafter\long
124            \expandafter\expandafter\expandafter\def
125            \expandafter\expandafter
126            \csname fe@getitem@\fe@levelrn\endcsname
127            \expandafter##\expandafter1#2\expandafter
128            {\csname fe@setitem@\fe@levelrn\endcsname{##1}}
129    }

```

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.

```

130    \def\fe@newlevel#1
131    {
132        \fe@DefLevelVar{fe}{\fe@levelrn}
133        {
134            process,next,item@abs,count@abs,first@abs,
135            last@abs,position@abs,function,check,getitem,
136            nextcommandcode,aftergroup,

```

```

137         aftergroup@,firsttoken,space,separator,setitem
138     }
139 \fe@CollectLevelVar{base}
140     {check,getitem,next,nextcommandcode}
141 \fe@CollectLevelVar{i}
142     {firsttoken,aftergroup,aftergroup@,space,separator}
143 \fe@CollectLevelVar{ii}
144     {count@abs,first@abs}
145 \fe@CollectLevelVar{iii}
146     {process,function,item@abs,last@abs,setitem}
147 \expandafter\expandafter\expandafter\fe@newlevel@i
148     \expandafter\fe@level@base\fe@level@i
149 \expandafter\fe@newlevel@ii\fe@level@ii
150 \expandafter\expandafter\expandafter\fe@newlevel@iii
151     \expandafter\fe@level@base\fe@level@iii
152 }

```

`\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>`.

```

153 \long\def\fe@newlevel@i#1#2#3#4#5#6#7#8#9
154 {
155     \def#6{\fe@fnsl#4#8#7}
156     \def#7{
157         \ifx#9#4
158             \ifnum#8=\z@
159                 \fe@braces@ii#2#5{ }
160             \else
161                 \fe@braces@ii#2#5{ }
162             \fi
163         \else
164             \ifnum#8=\z@
165                 \fe@braces@i#2#5{ }
166             \else
167                 \fe@braces@i#2#5{ }
168             \fi
169         \fi
170         \fe@item@check@next
171     }
172 \def\fe@i##1
173 {
174     \fe@def@check
175     {
176         \long\def#5{##1}
177         \ifcat\noexpand#4\bgroup
178             \ifx#9\empty
179                 \expandafter\def\expandafter#3\expandafter
180                 {
181                     \expandafter#2\expandafter
182                     {\expandafter{#5}}
183                 }
184             \else
185                 \let#3#6
186             \fi
187         \else

```



```

188             \expandafter\def\expandafter#3
189             \expandafter{\expandafter#2#5}
190         \fi
191     }
192     {#1#1}#2#3#4
193 }
194 }
195 \long\def\fe@braces@i#1#2#3
196 {
197     \expandafter\def\expandafter\fe@item@check@next\expandafter
198     {\expandafter#1\expandafter{#2}#3}
199 }
200 \long\def\fe@braces@ii#1#2#3
201 {
202     \expandafter\def\expandafter\fe@item@check@next\expandafter
203     {\expandafter#1\expandafter{\expandafter{#2}}#3}
204 }

```

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

```

205 \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`.

```

206 \long\def\fe@newlevel@iii#1#2#3#4#5#6#7#8#9
207 {
208     \fe@def@setitem#9#7#5
209     \fe@def@process#5#3#1#6#4#7#8
210 }

```

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

```

211 \long\def\fe@def@check#1#2#3#4#5#6#7
212 {
213     \long\def#2
214     {
215         \ifx#5\endforeach
216         \let#4\fe@endlevel
217         \else
218         \advance#6\@ne
219         \thislevelcount#6
220         \ifnum#6=\tw@
221         \chardef#7\z@
222         \fi
223         #1
224         \fi
225         #4
226     }
227 }

```

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

```

228 \long\def\fe@def@setitem#1#2#3
229 {

```

```

230     \long\def#1##1
231     {
232     \long\def#2{##1}
233     \let\thislevelitem#2
234     #3
235     }
236     }

```

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

```

237     \long\def\fe@def@process#1#2#3#4#5#6#7
238     {
239     \long\def#1
240     {
241     \ifx#5\endforeach
242     \chardef#7\@ne
243     \fi
244     \def#2{\fe@fnsl@#5#3}
245     #4#6
246     #2
247     }
248     }

```

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.

```

249     \def\fe@endlevel
250     {
251     \chardef\fe@count@total\thislevelcount
252     \advance\fe@level@m@ne
253     \expandafter\def\expandafter\fe@levelrn\expandafter
254     {\romannumeral\fe@level}
255     \expandafter\fe@define
256     \csname fe@relmax@abs@\fe@levelrn\endcsname\fe@levelvars
257     \expandafter\thislevelcount
258     \csname fe@count@abs@\fe@levelrn\endcsname
259     \expandafter\let\expandafter\thislevelitem
260     \csname fe@item@abs@\fe@levelrn\endcsname
261     }

```

`\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 `\futurenonspacel` from *The TeXBook*.⁶

```

262     \begingroup\def\{\global\let\fe@fnsl@stoken= }\ \endgroup
263     \def\fe@fnsl#1#2#3
264     {
265     \def\fe@fnsl@cs{#1}
266     \def\fe@fnsl@space{#2}

```

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

```

267     \def\fe@fnsl@next@ii{#3}
268     \expandafter\chardef\fe@fnsl@space\z@
269     \fe@fnsl@stepone
270   }
271 \def\fe@fnsl@stepone
272   {\expandafter\futurelet\fe@fnsl@cs\fe@fnsl@steptwo}
273 \def\fe@fnsl@steptwo
274   {
275   \expandafter\ifx\fe@fnsl@cs\fe@fnsl@stoken
276     \let\fe@fnsl@next@i=\fe@fnsl@stepthree
277   \else
278     \let\fe@fnsl@next@i\fe@fnsl@next@ii
279   \fi
280   \fe@fnsl@next@i
281   }
282 \def\fe@fnsl@stepthree
283   {
284   \expandafter\chardef\fe@fnsl@space\@ne
285   \afterassignment\fe@fnsl@stepone\let\fe@fnsl@next@i= %
286   }
287 \def\fe@fnsl@#1#2
288   {
289   \def\fe@fnsl@cs{#1}
290   \def\fe@fnsl@next@ii{#2}
291   \fe@fnsl@stepone@
292   }
293 \def\fe@fnsl@stepone@
294   {\expandafter\futurelet\fe@fnsl@cs\fe@fnsl@steptwo@}
295 \def\fe@fnsl@steptwo@
296   {
297   \expandafter\ifx\fe@fnsl@cs\fe@fnsl@stoken
298     \let\fe@fnsl@next@i=\fe@fnsl@stepthree@
299   \else
300     \let\fe@fnsl@next@i\fe@fnsl@next@ii
301   \fi
302   \fe@fnsl@next@i
303   }
304 \def\fe@fnsl@stepthree@
305   {
306   \afterassignment\fe@fnsl@stepone@\let\fe@fnsl@next@i= %
307   }

```

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

```

308 \def\endforeach
309   {
310   \PackageError{forarray}
311     {Tried to expand an \string\endforeach token. %
312     Something is wrong.\MessageBreak
313     The level of the current list is: %
314     \the\fe@level\MessageBreak
315     The content of the current item is: %
316     \expandafter\strip@prefix\meaning\thislevelitem
317     \MessageBreak
318     The position of the item is:

```

```

319         \the\thislevelcount}
320     {No further immediate help available. Sorry.}
321 }

```

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`.

```

322 \def\fe@@define@process#1,
323 {
324     \def\fe@@define@Item{#1}
325     \ifx\fe@@define@Item\empty
326         \let\fe@@define@process@next\relax
327     \else
328         \def\fe@@define@process@next
329         {
330             \expandafter\expandafter\expandafter\def
331             \expandafter\expandafter
332             \csname
333                 \fe@VarMacro @\fe@@define@Item @rel@
334                 \romannumeral\fe@cnt@ii
335             \endcsname
336             \expandafter
337             {
338                 \csname
339                     \fe@VarMacro @\fe@@define@Item @abs@
340                     \romannumeral\fe@cnt@i
341                 \endcsname
342             }
343             \fe@@define@process
344         }
345     \fi
346     \fe@@define@process@next
347 }

```

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

```

348 \def\fe@@define
349 {
350     \ifnum\fe@cnt@i>\m@ne
351         \ifnum\fe@cnt@ii>\fe@define@max
352             \let\fe@@define@next\relax
353         \else
354             \def\fe@@define@next{
355                 \expandafter\fe@@define@process\fe@Vars,{},
356                 \advance\fe@cnt@ii\@ne
357                 \advance\fe@cnt@i\m@ne
358                 \fe@@define
359             }
360         \fi
361     \else
362         \let\fe@@define@next\relax
363     \fi
364     \fe@@define@next

```

365 }

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

```
366        \def\fe@define#1
367        {
368        \chardef\fe@define@max#1\relax
369        \ifnum\fe@define@max>\z@
370        \expandafter\fe@define@
371        \else
372        \expandafter\@gobble
373        \fi
374        }
375        \def\fe@define@#1
376        {
377        \def\fe@VarMacro{fe}
378        \let\fe@Vars#1
379        \fe@cnt@i\fe@level
380        \fe@cnt@ii\@ne
381        \fe@@define
382        }
```

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

```
383        \def\fe@ProcessList#1,
384        {
385        \def\fe@ProcessList@check{#1}
386        \ifx\fe@ProcessList@check\empty
387        \let\fe@ProcessList@next\relax
388        \else
389        \def\fe@ProcessList@next
390        {
391        \fe@ProcessList@act{#1}
392        \fe@ProcessList
393        }
394        \fi
395        \fe@ProcessList@next
396        }
```

`\fe@DefLevelVar` The macro `\fe@DefLevelVar` redefines the expansion of the pointers `<fe|fa>@<pointer name>@level`.

```
397        \def\fe@DefLevelVar#1#2#3
398        {
399        \def\fe@ProcessList@act##1
400        {
401        \expandafter\expandafter\expandafter\def
402        \expandafter\expandafter
403        \csname #1@##1@level\endcsname
404        \expandafter
405        {\csname #1@##1@#2\endcsname}
406        }
407        \fe@ProcessList#3,{},
408        }
```

`\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'.

```

409 \def\fe@define@position
410 {
411   \ifnum\fe@level=\@ne
412     \expandafter\expandafter\expandafter
413     \def\expandafter\expandafter
414     \csname fe@position@abs@\fe@levelrn\endcsname
415     \expandafter
416     {
417       \csname fe@position@abs@\fe@upperlevelrn\endcsname
418       \number\thislevelcount
419     }
420   \else
421     \expandafter\expandafter\expandafter
422     \def\expandafter\expandafter
423     \csname fe@position@abs@\fe@levelrn\endcsname
424     \expandafter
425     {
426       \csname fe@position@abs@\fe@upperlevelrn\endcsname
427       -\number\thislevelcount
428     }
429   \fi
430 }

```

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

```

431 \def\fe@AddToTokensX#1#2
432 {
433   \expandafter\expandafter\expandafter#1
434   \expandafter\expandafter\expandafter
435   {\expandafter\the\expandafter#1#2}
436 }

```

`\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`.

```

437 \def\fe@CollectLevelVar@#1#2#3
438 {
439   \expandafter\def\csname #1@level@#2\endcsname{}
440   \def\fe@ProcessList@act##1{
441     \expandafter\fe@AddToTokensX\expandafter
442     \fe@toks\csname #1@##1@level\endcsname
443   }
444   \fe@toks{}
445   \fe@ProcessList#3,{},
446   \expandafter\expandafter\expandafter\def
447   \expandafter\expandafter
448   \csname #1@level@#2\endcsname\expandafter{\the\fe@toks}
449 }

```

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

```

450 \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.

```
451 \def\ForArray
452 {
453   \fe@aux@advancechardef\fa@arraylevel\@ne
454   \expandafter\def\expandafter\fa@arraylevelrn\expandafter
455     {\romannumeral\fa@arraylevel}
456   \@ifnextchar(\fa@FA@WRelMax\fa@FA@WoRelMax
457   }
458 \def\fa@FA@WRelMax(#1)
459 {
460   \expandafter\chardef
461     \csname fa@relmax@\fa@arraylevelrn\endcsname#1\relax
462   \ForArray@
463   }
464 \def\fa@FA@WoRelMax
465 {
466   \expandafter\chardef
467     \csname fa@relmax@\fa@arraylevelrn\endcsname\fe@relmax
468   \ForArray@
469   }
470 \def\ForArray@
471   {\@ifnextchar[\fa@FA@WSepListSep\fa@FA@WoSepListSep}
472 \def\fa@FA@WSepListSep[#1]{\fa@FA@SepList#1}
473 \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.

```
474 \def\fe@aux@advancechardef#1#2
475 {
476   \count@#1
477   \advance\count@#2
478   \chardef#1\count@
479   }
480 \def\fa@FA@SepList#1#2
481 {
482   \fe@DefLevelVar{fa}{\fa@arraylevelrn}
483   {
484     separatorcount,oldcatcode,baselevel,level,olddef,
485     array,restore,next,separator,orientation,nextlevel,
486     oldnextlevel,oldlowernextlevel
487   }
488   \expandafter\chardef
489     \csname fa@level@\fa@arraylevelrn\endcsname\z@
490   \ForEach
```

```

491     {#1}
492     {
493         \expandafter\expandafter\expandafter\def
494         \expandafter\expandafter
495         \csname
496             fa@separator@\fa@arraylevelrn @
497             \romannumeral\thislevelcount
498         \endcsname
499         \expandafter
500         {\thislevelitem}
501     }
502     {#2}
503     \expandafter\chardef\fa@separatorcount@level\fe@count@total
504     \fa@FA@MarkerList
505     }

```

`\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.

```

506     \def\fa@FA@MarkerList
507     {\@ifnextchar[\fa@FA@WMarkerList\fa@FA@SublevelToken}
508     \def\fa@FA@WMarkerList[
509     {\@ifnextchar[\fa@FA@WMarkerListSep\fa@FA@WoMarkerListSep}
510     \def\fa@FA@WMarkerListSep[#1]#2]{\fa@FA@MarkerList@{#1}-{#2}}
511     \def\fa@FA@WoMarkerListSep#1]{\fa@FA@MarkerList@{#1}}

```

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

```

512     \def\fa@FA@MarkerList@#1#2
513     {
514         \ForEach
515         {#1}
516         {
517             \expandafter\expandafter\expandafter\def
518             \expandafter\expandafter
519             \csname
520                 fa@orientation@\fa@arraylevelrn @
521                 \romannumeral\thislevelcount
522             \endcsname
523             \expandafter
524             {\thislevelitem}
525         }
526         {#2}
527         \fa@FA@SublevelToken
528     }

```

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

```

529     \def\fa@FA@SublevelToken#1
530     {
531         \expandafter\fa@SublevelToken\fa@array@level#1
532         \fa@FA@Process
533     }

```


`\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.

```

534 \long\def\fa@FA@Process#1#2#3
535 {
536   \ForEach
537     {#1}
538     {
539       \expandafter\expandafter\expandafter\def
540         \expandafter\expandafter
541         \csname
542           fa@function@\fa@arraylevelrn @
543           \romannumeral\thislevelcount
544         \endcsname
545         \expandafter
546         {\thislevelitem}
547     }
548   {#2}
549   \long\def\thislevelitem{#3}
550   \fe@cnt@i\fe@level
551   \advance\fe@cnt@i\@ne
552   \expandafter\chardef\fa@baselevel@level\fe@cnt@i
553   \expandafter\def\fa@array@level
554     {
555       \ifnum\fa@level@level=\fa@separatorcount@level\relax
556         \expandafter\def\fa@next@level{\thislevelitem}
557       \else
558         \expandafter\let\fa@next@level\fa@next@level@
559
560       \fi
561       \fa@next@level
562     }
563   \fa@array@level
564   \fa@restore@level
565   \fe@aux@advancechardef\fa@arraylevel\m@ne
566   \expandafter\def\expandafter\fa@arraylevelrn\expandafter
567     {\romannumeral\fa@arraylevel}
568   }
569   \def\fa@next@level@
570     {
571       \fa@SetLevelVars\tw@
572       \expandafter\expandafter\expandafter\def
573         \expandafter\expandafter\expandafter\fa@i
574         \expandafter\expandafter\expandafter
575         {
576           \csname
577             fa@separator@\fa@arraylevelrn @
578             \romannumeral\fa@level@level
579           \endcsname
580         }
581       \expandafter\expandafter\expandafter\expandafter
582         \expandafter\expandafter\expandafter
583       \ForEachSublevel@
584         \expandafter\expandafter\expandafter\expandafter
585       \expandafter\expandafter
586       \csname

```

```

587         fa@relmax@\fa@arraylevelrn
588     \endcsname
589     \expandafter
590     \fa@i
591     \expandafter
592     {
593     \csname
594         fa@function@\fa@arraylevelrn @
595         \romannumeral\fa@level@level
596     \endcsname
597     }
598     \fa@SetLevelVars\@ne
599     }
600 \def\fa@SetLevelVars#1
601     {
602     \fe@cnt@i\fe@level
603     \advance\fe@cnt@i-\fa@baselevel@level
604     \advance\fe@cnt@i#1
605     \expandafter\chardef\fa@level@level\fe@cnt@i
606     \thislevelnr\fa@level@level
607     \expandafter\let\expandafter\thislevelmarker
608     \csname
609         fa@orientation@\fa@arraylevelrn @
610         \romannumeral\fa@level@level
611     \endcsname
612     }

```

`\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`.

```

613 \def\ExitForEachLevels#1#2
614     {
615     \fe@cnt@i\fe@level
616     \fe@cnt@ii\z@
617     \advance\fe@cnt@i\@ne
618     \advance\fe@cnt@i-#1\relax
619     \def\fe@exitforeach
620     {
621     \ifnum\fe@cnt@ii<#2\relax
622     \def\fe@exitforeach@next
623     {
624     \expandafter\let
625     \csname fe@next@\romannumeral\fe@level\endcsname
626     \fe@ExitForEach@base
627     \advance\fe@cnt@i\m@ne
628     \advance\fe@cnt@ii\@ne
629     }
630     \else
631     \let\fe@exitforeach@next\relax
632     \fi
633     \fe@exitforeach@next
634     }
635     \fe@exitforeach
636     }

```

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

```
637 \def\fa@SublevelToken#1#2
638 {
639   \expandafter\if\noexpand#2\relax
640     \expandafter\let\fa@olddef@level#2
641     \def#2{#1}
642     \expandafter\def\fa@restore@level
643       {\expandafter\let#2\fa@olddef@level}
644   \else
645     \chardef\fa@oldcatcode@level\catcode'#2\relax
646     \ifnum\fa@oldcatcode@level=\active
647       \expandafter\let\fa@olddef@level#2
648     \else
649       \catcode'#2\active
650     \fi
651     \expandafter\def\fa@restore@level
652       {
653         \ifnum\fa@oldcatcode@level=\active
654           \expandafter\expandafter\expandafter
655             \fa@SublevelToken
656           \expandafter\expandafter\expandafter
657             {\fa@olddef@level}{#2}
658         \else
659           \catcode'#2\fa@oldcatcode@level
660         \fi
661       }
662     \bgroup
663       \uccode'\~'#2\relax
664       \uppercase{
665         \egroup
666           \def~{#1}
667         }
668     \fi
669 }
```

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.

```
670 \def\DefineArrayVar#1#2#3#4#5#6
671 {
672   \ForEach{#5}
673   {
674     \expandafter\expandafter\expandafter
675       \def\expandafter\expandafter
676       \csname
677         fe@item@nr@\number\thislevelcount
```

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

```

678         \endcsname
679         \expandafter{\thislevelitem}
680     }
681     {#6}
682 \ForEach{#3}
683 {
684     \expandafter\ifx
685         \csname
686             fe@item@nr@\number\thislevelcount
687         \endcsname
688         \relax
689         \fe@DefineArrayVar@Warning{#4}{#5}
690         \ExitForEach
691     \else
692         \expandafter\expandafter\expandafter\def
693         \expandafter\expandafter\expandafter
694         \fa@ArrayVarContent
695         \expandafter\expandafter\expandafter
696         {
697             \csname
698                 fe@item@nr@\number\thislevelcount
699             \endcsname
700         }
701         \expandafter\ifx
702             \csname
703                 fe@item@nr@\number\thislevelcount
704             \endcsname
705             \empty
706             \typeout
707                 {Content of \expandafter\string
708                 \csname #1#2\thislevelitem\endcsname\space
709                 is set to nothing.}
710         \else
711             \typeout
712                 {Content of \expandafter\string
713                 \csname #1#2\thislevelitem\endcsname\space
714                 is set to %
715                 \expandafter\strip@prefix
716                 \meaning\fa@ArrayVarContent.}
717         \fi
718         \expandafter\expandafter\expandafter\def
719         \expandafter\expandafter
720         \csname #1#2\thislevelitem\endcsname\expandafter
721         {\fa@ArrayVarContent}
722     \fi
723 }
724 {#4}
725 }
726 \def\fe@DefineArrayVar@Warning#1#2
727 {
728     \PackageWarning
729     {fornext}
730     {
731         No more items available while %
732         defining pointers!\MessageBreak
733         Pointers: #1\MessageBreak

```

```

734         Items:\space\space\space #2
735     }
736 }

```

\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.

```

737 \def\DefineArrayVars#1#2#3#4#5#6#7
738 {
739     \typeout{\typeout{Defining Array Variables...}}
740     \def\fe@DefineArrayVar@##1#3##2#3##3#2
741     {
742         \typeout{-- Initializing new variable array: ##1}
743         \DefineArrayVar
744             {##1}{#5}{#1}{##3}{#4}{##2}
745     }
746     \ForEach
747         {#2}
748         {\expandafter\fe@DefineArrayVar@\thislevelitem#3#6#2}
749         {#7}
750 }

```

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

```

751 \def\DefineArrayDefault#1#2#3#4#5
752 {
753     \ForEach
754         {#1}
755         {
756             \expandafter\edef\csname\thislevelitem\endcsname
757             {
758                 \noexpand\expandafter\noexpand\ifx
759                 \noexpand\csname
760                     \thislevelitem #2\noexpand#3
761                 \noexpand\endcsname
762                 \noexpand\relax
763                 \noexpand\csname
764                     \thislevelitem #2#4
765                 \noexpand\endcsname
766                 \noexpand\else
767                 \noexpand\csname
768                     \thislevelitem #2\noexpand#3
769                 \noexpand\endcsname
770                 \noexpand\fi
771             }
772         }
773         {#5}
774 }

```

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

```

775 \def\DefineArrayVarTo#1#2#3#4#5
776 {
777     \ForEach

```

```

778         {#1}
779         {\expandafter\expandafter\expandafter\def\expandafter
780          \csname #3#2\thislevelitem\endcsname{#4}}
781         {#5}
782     }

```

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

```

783 \def\CommandForEach#1#2#3
784   {\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.

```

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

```

4.5 Final commands

```

787 \endlinechar\fe@aux@endlinecharrestore\relax
788 \makeatother
789 %</sty>

```

5 Test page

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

```

790 %<*test>
791 \documentclass[10pt,a4paper]{article}
792 \usepackage[latin1]{inputenc}
793 \usepackage[T1]{fontenc}
794 \usepackage[margin=2cm]{geometry}
795 \usepackage[dvips, pdfborder={0 0 0}, pdfstartview={FitH},
796   pdfpagelayout={OneColumn}, bookmarks=false, pdfnewwindow,
797   unicode=true]{hyperref}
798 \urlstyle{same}
799 \usepackage{examplep}
800 \usepackage{forarray}
801 \author
802   {
803     Christian Schr\{"o}ppel%
804     \footnote%
805     {
806       Please send any comments or suggestions to %
807       \protect\href{mailto:christian@schroeppe.com}%
808     {\mbox{\fontfamily{cmss}\selectfont christian@schroeppe.com}}}.%
809     }
810   }
811 \title
812   {
813     Test page for the %
814     {\fontfamily{cmss}\selectfont forarray} %
815     package\ \ [.5ex]\Large Version 1.01 (2008/06/20)

```

```

816     }
817 \def\ShowExample{
818     \PexaShowBoth{
819         yalign=b,
820         allowbreak=yes,
821         srcstyle=leftnumcol,
822     }
823 }
824 \arrayrulewidth=0pt
825 \begin{document}
826 \errorcontextlines=20\relax
827 \maketitle
828 \thispagestyle{empty}
829 \small
830 \section[Test of \texttt{ForEach} (Simple List)]
831 \begin{WBoth}
832 \begin{itemize}
833 \ForEach{,}
834     {\item Item No.\ %
835         \the\thislevelcount\ is:
836         ‘\thislevelitem’\{\footnotesize
837         \meaning\thislevelitem}}
838     {
839     {Hello, World!}, Sec{ond},
840     {Thi}rd,{\bf Last} item%
841     }
842 \end{itemize}
843 \end{WBoth}
844 \ShowExample
845 \section[Test of \texttt{ForEach} (Nested)]
846 \begin{WBoth}
847 \begin{itemize}
848 \ForEach{;}
849     {\item[\the\thislevelcount]}
850     \raggedright\thislevelitem
851     \begin{itemize}
852     \ForEachX{,}
853         {\item Item No.\ %
854             \the\thislevelcount\ %
855             is: \thislevelitem}
856         {\thislevelitem}
857     \end{itemize}}
858     {\$\alpha$, $\beta$, $\gamma$;
859     {\Large A Large Item},
860     Transparency \it test,
861     Ends \rm here.}
862 \end{itemize}
863 \end{WBoth}
864 \ShowExample
865 \section[Test of \texttt{ForArray}]
866 \makeatletter
867 \begin{WBoth}
868 \parindent=0pt
869 \def\MyArray{\ForArray(3){,}{*}{|}}
870 \MyArray
871     {*\par\vskip 3ex|\parbox{8em}{*}}

```

```

872     {
873     A,B,C;
874     \textit{A nested array:}\par
875     \MyArray{[*]\par|(*)}{1,2;3,4},b,c;
876     $\alpha$, $\beta$, $\gamma$
877     }
878 \end{WBoth}
879 \ShowExample
880 \end{document}
881 %</test>

```

6 Copyright Notice

The package `forarray` is © 2008 by Christian Schröppel. It may be distributed and/or modified under the conditions of the L^AT_EX 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@schroepel.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 L^AT_EX 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.

C	
<code>\CommandForEach</code>	12 , <u>783</u>
D	
<code>\DefineArrayDefault</code> . . .	10 , <u>751</u>
<code>\DefineArrayVar</code>	8 , <u>670</u> , 743
<code>\DefineArrayVars</code>	9 , <u>737</u>
<code>\DefineArrayVarTo</code>	11 , <u>775</u>
E	
<code>\endforeach</code>	54, 61, 70, 78, 215, 241, <u>308</u>
<code>\ExitForEach</code>	5 , <u>72</u> , 690
<code>\ExitForEachLevels</code>	7 , <u>613</u>
F	
<code>\fa@array@level</code> . . .	531, 553, 563
<code>\fa@arraylevel</code>	37, 453, 455, 565, 567
<code>\fa@arraylevelrn</code>	454, 461, 467, 482, 489, 496, 520, 542, 566, 577, 587, 594, 609
<code>\fa@ArrayVarContent</code>	694, 716, 721
<code>\fa@baselevel@level</code> . . .	552, 603
<code>\fa@FA@MarkerList</code>	504, <u>506</u>
<code>\fa@FA@MarkerList@</code>	510, 511, <u>512</u>
<code>\fa@FA@Process</code>	532, <u>534</u>
<code>\fa@FA@SepList</code>	472, 473, <u>474</u>
<code>\fa@FA@SublevelToken</code>	507, 527, <u>529</u>
<code>\fa@FA@WMarkerList</code>	507, 508
<code>\fa@FA@WMarkerListSep</code> . .	509, 510
<code>\fa@FA@WoMarkerListSep</code> .	509, 511
<code>\fa@FA@WoRelMax</code>	456, 464
<code>\fa@FA@WoSepListSep</code> . . .	471, 473
<code>\fa@FA@WRelMax</code>	456, 458
<code>\fa@FA@WSepListSep</code>	471, 472
<code>\fa@i</code>	573, 590
<code>\fa@level@level</code>	555, 578, 595, 605, 606, 610
<code>\fa@next@level</code>	556, 558, 561
<code>\fa@next@level@</code>	558, 569
<code>\fa@oldcatcode@level</code>	645, 646, 653, 659
<code>\fa@olddef@level</code>	640, 643, 647, 657
<code>\fa@restore@level</code> .	564, 642, 651
<code>\fa@separatorcount@level</code>	503, 555
<code>\fa@SetLevelVars</code> . .	571, 598, 600
<code>\fa@SublevelToken</code>	531, <u>637</u>
<code>\fe@@define</code>	<u>348</u> , 381
<code>\fe@@define@Item</code>	324, 325, 333, 339
<code>\fe@@define@next</code>	352, 354, 362, 364
<code>\fe@@define@process</code> . . .	<u>322</u> , 355
<code>\fe@@define@process@next</code>	326, 328, 346
<code>\fe@AddToTokensX</code>	<u>431</u> , 441
<code>\fe@aux@advancechardef</code>	453, 474, 565
<code>\fe@aux@endlinecharrestore</code>	28, 787
<code>\fe@braces@i</code>	165, 167, 195
<code>\fe@braces@ii</code>	159, 161, 200
<code>\fe@checkifdefined</code>	5, 19, 21
<code>\fe@cnt@i</code>	32, 340, 350, 357, 379, 550–552, 602–605, 615, 617, 618, 627
<code>\fe@cnt@ii</code>	33, 334, 351, 356, 380, 616, 621, 628
<code>\fe@CollectLevelVar</code>	139, 141, 143, 145, <u>450</u>
<code>\fe@CollectLevelVar@</code> . .	<u>437</u> , 450
<code>\fe@count@abs@</code>	39
<code>\fe@count@total</code>	251, 503
<code>\fe@def@check</code>	174, <u>211</u>
<code>\fe@def@process</code>	209, <u>237</u>
<code>\fe@def@setitem</code>	208, <u>228</u>
<code>\fe@define</code>	100, 255, <u>366</u>
<code>\fe@define@</code>	370, 375
<code>\fe@define@max</code>	351, 368, 369
<code>\fe@define@position</code>	91, <u>409</u>
<code>\fe@DefineArrayVar@</code> . . .	740, 748
<code>\fe@DefineArrayVar@Warning</code>	689, 726
<code>\fe@definelevel</code> . . .	106, 108, 116
<code>\fe@DefLevelVar</code> . . .	132, <u>397</u> , 482
<code>\fe@empty@abs@</code>	45
<code>\fe@emptytest</code>	104, 105
<code>\fe@endlevel</code>	216, <u>249</u>
<code>\fe@exitforeach</code>	619, 635
<code>\fe@ExitForEach@base</code>	76, 78, 626
<code>\fe@exitforeach@next</code>	622, 631, 633
<code>\fe@first@abs@</code>	43
<code>\fe@fnsl</code>	155, <u>262</u>
<code>\fe@fnsl@</code>	110, 244, 287
<code>\fe@fnsl@cs</code>	265, 272, 275, 289, 294, 297
<code>\fe@fnsl@next@i</code>	276, 278, 280, 285, 298, 300, 302, 306
<code>\fe@fnsl@next@ii</code>	267, 278, 290, 300
<code>\fe@fnsl@space</code>	266, 268, 284
<code>\fe@fnsl@stepone</code> . .	269, 271, 285
<code>\fe@fnsl@stepone@</code> .	291, 293, 306

<code>\fe@fnsl@stepthree</code> 276, 282	<code>\ForEach</code> 3 , 51 , 490, 514, 536, 672, 682, 746, 753, 777, 784, 786, 833, 848
<code>\fe@fnsl@stepthree@</code>	... 298, 304	<code>\ForEach@</code> 52-54
<code>\fe@fnsl@steptwo</code> 272, 273	<code>\ForEach@Arg</code> 51, 52
<code>\fe@fnsl@steptwo@</code> 294, 295	<code>\ForEach@NoArg</code> 51, 53
<code>\fe@fnsl@stoken</code>	... 262, 275, 297	<code>\ForEachD</code> 5 , 48
<code>\fe@i</code> 60, 61, 69, 70, 172, 205	<code>\ForEachD@</code>	.. 49, 50, 54, 60, 69, 79
<code>\fe@item@abs@</code> 42	<code>\ForEachD@Arg</code> 48, 49
<code>\fe@item@check@next</code>	170, 197, 202	<code>\ForEachD@NoArg</code> 48, 50
<code>\fe@last@abs@</code> 44	<code>\ForEachSublevel</code> 4 , 63
<code>\fe@level</code> 31, 36, 75, 82, 84, 85, 118, 120, 252, 254, 314, 379, 411, 550, 602, 615, 625	<code>\ForEachSublevel@</code>	... 65-67, 583
<code>\fe@level@base</code> 148, 151	<code>\ForEachSublevel@Arg</code>	... 64, 65
<code>\fe@level@i</code> 148	<code>\ForEachSublevel@NoArg</code>	.. 64, 66
<code>\fe@level@ii</code> 149	<code>\ForEachX</code> 4 , 55 , 852
<code>\fe@level@iii</code> 151	<code>\ForEachX@</code> 56-58
<code>\fe@levelrn</code> 47, 81, 83, 87, 89, 93, 95, 97, 99, 101, 103, 112, 113, 122, 126, 128, 132, 253, 256, 258, 260, 414, 423	<code>\ForEachX@Arg</code> 55, 56
<code>\fe@levelvars</code> 101, 115, 256	<code>\ForEachX@NoArg</code> 55, 57
<code>\fe@newlevel</code> 119, 130	<code>\FunctionForEach</code> 12 , 785
<code>\fe@newlevel@i</code> 147, 153	M	
<code>\fe@newlevel@ii</code> 149, 205	<code>\MyArray</code> 869, 870, 875
<code>\fe@newlevel@iii</code> 150, 206	P	
<code>\fe@position@abs@</code> 46	<code>\PackageError</code> 10, 310
<code>\fe@ProcessList</code>	... 383 , 407, 445	<code>\PackageWarning</code> 728
<code>\fe@ProcessList@act</code>	391, 399, 440	T	
<code>\fe@ProcessList@check</code>	.. 385, 386	<code>\thislevelcount</code> 4 , 34, 219, 251, 257, 319, 418, 427, 497, 521, 543, 677, 686, 698, 703, 835, 849, 854
<code>\fe@ProcessList@next</code>	387, 389, 395	<code>\thislevelitem</code> 4 , 70, 233, 259, 316, 500, 524, 546, 549, 556, 679, 708, 713, 720, 748, 756, 760, 764, 768, 780, 784, 786, 836, 837, 850, 855, 856
<code>\fe@relmax</code>	. 40, 50, 53, 57, 66, 467	<code>\thislevelmarker</code> 7 , 607
<code>\fe@relmax@abs@</code> 41	<code>\thislevelnr</code> 7 , 35, 606
<code>\fe@toks</code> 30, 442, 444, 448		
<code>\fe@toplevel</code>	... 38, 85, 118, 120		
<code>\fe@upperlevelrn</code>	... 81, 417, 426		
<code>\fe@VarMacro</code> 333, 339, 377		
<code>\fe@Vars</code> 355, 378		
<code>\ForArray</code> 6 , 451 , 869		
<code>\ForArray@</code> 462, 468, 470		

*** END OF INDEX ***