

The `asciilist` package*

Richard Gay
`gay@mais.informatik.tu-darmstadt.de`

April 9, 2016

Abstract

This package provides two environments for *quickly typesetting nested lists* in \LaTeX without having to type the individual `\item` macros or opening/closing nested list environments.

1 Usage

We document the functionality of the package by examples in this section. The package provides two main environments: `AsciiList` and `AsciiDocList`.

The `AsciiList` environment (Section 1.1) provides a list environment in which a single character at the beginning of a line can be used to typeset an item at a particular list level. The mapping between these characters and list levels is fixed for the whole list.

The `AsciiDocList` environment (Section 1.2) provides a list environment in which a sequence of characters at the beginning of a line can be used to typeset an item at a particular list level. The mapping between individual characters and list environments is fixed for the whole list. This environment, thus, uses a syntax that is close to the asciidoc syntax.

1.1 Lists with Fixed Nesting Layout

AsciiList Use the `AsciiList[⟨environments⟩]{⟨item-chars⟩}` environment to quickly create \LaTeX lists, possibly nested ones, without too much \LaTeX interference. The following example might illustrate the environment best:

*This document corresponds to `asciilist` v2.1b, dated 2016/03/31. The package is available online at <http://www.ctan.org/pkg/asciilist> and <https://github.com/Ri-Ga/asciilist>.

```
\begin{AsciiList}{-,*,+,=}
- first item
  * first sub-item
  * second sub-item,
    which is multi-line
  + a sub-sub item
    = a sub-sub-sub item
- back up by three levels
  * end at a sub-item
\end{AsciiList}
```

- first item
 - first sub-item
 - second sub-item, which is multi-line
 - * a sub-sub item
 - a sub-sub-sub item
- back up by three levels
 - end at a sub-item

Note that in this example, the indentation with spaces is for demonstration purposes only. The package itself does not require a proper/uniform indentation of the items at the different levels. Item levels are solely recognized by the first character of a line.

`\AsciiListFromFile`

Instead of having the list content inline in the L^AT_EX file, one can also choose to put the list into a separate file. A list of such file can then be produced via the `\AsciiListFromFile[<environments>]{<item-chars>}{<file-name>}` macro. The first two parameters of the macro are the same as the parameters of the `AsciiList` environment. The *<file-name>* parameter specifies the name of the file to include.

```
\AsciiListFromFile{auto}{AsciiList.example}
```

- item
 - sub-item
- another item

In the above example, the used input file has the following content:

```
* item
  - sub-item
* another item
```

`\AsciiListFromFiles`

The `\AsciiListFromFiles` macro is the same as `\AsciiListFromFile`, except that a comma-separated list is accepted for the *<file-name>* parameter and that the files are input in the listed order.

1.1.1 Choosing List Environments

If you do not want to use the `itemize` environment for the lists, you can change it to `enumerate`, `compactitem` or other list-like environments (which should use `\item` for their items) you prefer by setting the optional *<environments>* parameter. For example, you can set the *<environments>* to `itemize,compactitem` in order to have the top level list as an `itemize` and the second as well as all deeper levels as `compactitem` lists.

The `AsciiList` environment allows you to use more than just list environments like `itemize`, `enumerate`, or `description`. You can even use sectioning command names (`chapter`, `section`, `section*`, `subsection`, `paragraph` etc.), which have been enabled already with the `\NewAsciiListEnv` macro. If you use such “environments”, every list entry will be produced using the respective `\chapter`,

`\section`, `\section*`, `\subsection`, `\paragraph`, etc. macro. Note, however, that only the first line of such an item will then be used for the name of the section/subsection/etc.

```
\begin{AsciiList}[section,subsection]{-,*}
- first section
  * a subsection
- second section
\end{AsciiList}
```

- 1 first section
 - 1.1 a subsection
- 2 second section

List environments with optional parameters are also supported, including their parameters. For instance, the `compactenum` environment of the `paralist` package has an optional parameter configuring the appearance of the individual items. The following example shows how to specify the optional parameters:

```
\begin{AsciiList}[compactenum<1.>,
                  compactenum<(a)>]{auto}
- item number 1.
  * item (a)
  * item (b)
- item number 2.
\end{AsciiList}
```

1. item number 1.
 - (a) item (a)
 - (b) item (b)
2. item number 2.

Care has to be taken when the optional argument contains a comma. In this case, the optional argument must be enclosed in parentheses such that \LaTeX does not confuse the comma with a comma that separates the list environments.

1.1.2 Automatic Item Detection

If you want to be more flexible regarding the characters for the list items, you can use `auto` as the parameter to the environment. The `AsciiList` then makes an attempt to automatically identify the list items from a predefined list (preset to “-”, “*”, “+”). You can change this list using the `\AsciiListSetAutochars{<chars>}` macro, which expects `<chars>` to be a comma-separated list of chars. The ordering of this list does not matter. What matters is the ordering in which the characters appear in the environment.

`\AsciiListSetAutochars`

```
\AsciiListSetAutochars{+,*,-,=}
\begin{AsciiList}[enumerate,itemize,
                  compactitem]{auto}
- first item
  * first sub-item
  * second sub-item,
    which is multi-line
  + a sub-sub item
- back up by three levels
  * end at a sub-item
\end{AsciiList}
```

1. first item
 - first sub-item
 - second sub-item, which is multi-line
 - a sub-sub item
2. back up by three levels
 - end at a sub-item

1.1.3 Navigating Upwards

If you want to typeset a list in which a list item is continued after sub-items of that item, you can use the `\UP` and `\UPTO` macros. The `\UP[⟨N⟩]` macro goes back `⟨N⟩` levels (default: 1) without starting a new item at the resulting level. The `\UPTO{⟨N⟩}` macro goes back to level `⟨N⟩` (where the topmost level is 0) without starting a new item at the resulting level. The following example illustrates the use of the two macros.

```
\begin{AsciiList}{auto}
- first item
  * sub-item
  \UP[1]
  continuation one level higher
  * another sub-item
    + a sub-sub-item
  \UPTO{1}
  continuation on first-item level
\end{AsciiList}
```

- first item
 - sub-item
 continuation one level higher
 - another sub-item
 - * a sub-sub-item
 continuation on first-item level

1.2 Lists with AsciiDoc-Like Notation

Use the `AsciiDocList[⟨environments⟩]` environment to quickly create \LaTeX lists, possibly nested ones, without too much \LaTeX interference. The following example might illustrate the environment best:

```
\begin{AsciiDocList}
* first item (itemized)
  ** first sub-item (enumerated)
  ** second sub-item (enumerated),
    which is multi-line
  *** a sub-sub item
* back up by three levels
  ** end at a sub-item (itemized)
\end{AsciiDocList}
```

- first item (itemized)
 1. first sub-item (enumerated)
 2. second sub-item (enumerated), which is multi-line
 - a sub-sub item
- back up by three levels
 - end at a sub-item (itemized)

Note that in this example, the indentation with spaces is for demonstration purposes only. The package itself does not require a proper/uniform indentation of the items at the different levels.

By default, the `*` character maps to an `itemize` item, the `+` character maps to an `enumerate` item, and the `;` character maps to a description item (where the item label follows in the line after the `;` and the item text follows in the subsequent line).

Instead of having the list content inline in the \LaTeX file, one can also choose to put the list into a separate file. A list of such file can then be produced via the `\AsciiDocListFromFile[⟨environments⟩]{⟨file-name⟩}` macro. The optional parameter of the macro is the same as the parameter of the `AsciiDocList` environment. The `⟨file-name⟩` parameter specifies the name of the file to include.

```
\AsciiDocListFromFile{AsciiDocList.example}
```

- item
 1. sub-item
- another item

In the above example, the used input file has the following content:

```
* item
  *+ sub-item
* another item
```

`\AsciiDocListFromFiles`

The `\AsciiDocListFromFiles` macro is the same as `\AsciiDocListFromFile`, except that a comma-separated list is accepted for the `<file-name>` parameter and that the files are input in the listed order.

1.2.1 Choosing List Environments

If you do not want to use the pre-defined mapping of `*` to `itemize` etc., you can change the mapping to other list-like environments (which should use `\item` for their items) you prefer by setting the optional `<environments>` parameter. For example, you can set the `<environments>` to `*=itemize,-=compactitem,!=enumerate`.

The `AsciiDocList` environment allows you to use more than just list environments like `itemize`, `enumerate`, or `description`. The range of supported environments is for `AsciiDocList` is the same as for `AsciiList`.

```
\begin{AsciiDocList}[/=section,
                    *=subsection]
/ first section
/* first subsection
/* second subsection
/ second section
\end{AsciiDocList}
```

- 1 first section
 - 1.1 first subsection
 - 1.2 second subsection
- 2 second section

1.2.2 Navigating Upwards

If you want to typeset a list in which a list item is continued after sub-items of that item, you can use the `\UPTO` macro. The `\UPTO{<chars>}` macro goes back the level identified by `<chars>` without starting a new item at the resulting level. The following example illustrates the use of the macro.

`\UPTO`

```
\begin{AsciiDocList}
; description
beginning of description
;+ item
;+ sub-item
\UPTO{;}
continuation of description
\end{AsciiDocList}
```

- description** beginning of description
1. item
 - sub-item
- continuation of description

2 Customizing List Environments

`\AsciiListRegisterEnv` You can register additional environment names for use with `AsciiList` or `AsciiDocList` by using the `\AsciiListRegisterEnv{<envname>}{<begin>}{<end>}{<item>}` macro, by which the `<begin>`, `<end>`, and `<item>` code can be specified. If you want to use a command for the `<item>` code that takes a single parameter (e.g., `\section`), then you can use `\AsciiListEndArg{<command>}` for the `<item>` to pass the whole line of the item to `<command>`. If the `<item>` takes a single *optional* parameter (like `\item`), then use `\AsciiListEndOArg{<command>}` instead of `\AsciiListEndArg`. For example to register an environment `TTEnum` for enumerating items in typewriter font, you can use the following command:

```
\AsciiListRegisterEnv{TTEnum}
  {\enumerate}{\endenumerate}
  {\AsciiListEndArg{\item\texttt}}
\begin{AsciiList}[TTEnum]{-}
- items are typewriter
  (though only in the first line)
- and enumerated
\end{AsciiList}
```

1. items are typewriter (though only in the first line)
2. and enumerated

2.1 Customizing AsciiList

`\AsciiListSetEnvironments` You can also change the default list environments for `AsciiList` from `itemize` to a comma-separated list of `<environments>`. This allows you to omit the optional parameter to the `AsciiList` environment for such default configurations. For this, use the `\AsciiListSetEnvironments{<environments>}` macro.

```
\AsciiListSetEnvironments{enumerate,
                           compactitem<!>}
\begin{AsciiList}{-,*}
- first
  * sub
- second
\end{AsciiList}
```

1. first
 - ! sub
2. second

`\NewAsciiListEnv` If particular kinds of nested lists occur several times in your document, you can also define abbreviation environments, as demonstrated in the following (which also shows how to use `description` environments in an `AsciiList`):

```
\NewAsciiListEnv[description,compactitem]
  {auto}{TopicIdeas}
\begin{TopicIdeas}
+ food:
- cheese
- nuts
+ beverages:
- water
- milk
\end{TopicIdeas}
```

```
food:      • cheese
           • nuts
beverages: • water
           • milk
```

2.2 Customizing AsciiDocList

`\AsciiDocListSetEnvironments` You can also change the default list environments for `AsciiDocList` from `*=itemize,+=enumerate` to another comma-separated list of *environments*. This allows you to omit the optional parameter to the `AsciiDocList` environment for such default configurations. For this, use the `\AsciiDocListSetEnvironments{environment}` macro.

```
\AsciiDocListSetEnvironments{?=enumerate,
                             !=compactitem<!=>}
\begin{AsciiDocList}
? item?
?! sub!
? second
\end{AsciiDocList}
```

1. item?
- ! sub!
2. second

`\NewAsciiDocListEnv` The analogous macro to `\NewAsciiListEnv` for the `AsciiDocList` environment is `\NewAsciiDocListEnv`. Its use is as the following example illustrates:

```
\NewAsciiDocListEnv[!=Description,
                    *=compactitem]{TopicDoc}
\begin{TopicDoc}
! food:
!* cheese
!* nuts
! beverages:
!* water
!* milk
\end{TopicDoc}
```

food:

- cheese
- nuts

beverages:

- water
- milk

3 Noteworthy Features

- This package can be used together with `SyncTeX`. That is, for a point in the `AsciiList` of a generated PDF, one can obtain the position in the `AsciiList`'s source code.
- Some initial tests show that `AsciiList` can be used in a p-column of a tabular environment.

4 Known Limitations

- Currently, the `AsciiList` and `AsciiDocList` environments cannot be used in a “moving argument”, such as in a `\footnote`. If you really want such lists in footnotes, you might consider using `\AsciiListFromFile` and friends.
- Not all chars can directly be used as item chars. This probably is some character code issue that so far has not been investigated further.

5 Related Packages

The following L^AT_EX packages provide similar functionalities to the `asciilist` package.

easylist: This package is probably closest to `asciilist`. The package “is designed for typesetting lists of numbered items [...] with a single active character acting as the only command” (cited from the package documentation). The package provides a lot of possibilities for configuring the appearance of items at the individual levels. Compared to `asciilist`, the package differs in two main points. Firstly, the package uses a single character for the items. This character can be used in the middle of a line to begin a new item and, hence, must not occur in the text itself. The character must be repeated for indicating the depth of the item, rather than having separate characters for different levels. Secondly, the package is focused on list environments and does not support using sectioning macros for items at selected levels. Finally, the package does not provide convenience functionality similar to that described in Section 2 of this documentation.

iitem: This package provides macros `\iitem`..`\ivtem`, which can be used within `enumerate` and `itemize` environments for changing list levels without explicitly starting nested `enumerate` and `itemize` environments. The list type (`enumerate`, `itemize`, ...) is the same for all nested list levels. Compared to `asciilist`, the package has the main drawback that the list type is the same for all levels. Moreover, the package uses L^AT_EX-ish enumeration macros rather than nice symbols. The package does not provide convenience functionality similar to that described in Section 2 of this documentation. Finally, the package has the limitation that items below the first level must be placed within a single line.

outlines: This package provides an `outline` environment, in which multiple list levels can be accessed. The individual levels can be reached via `\1`..`\4` macros, which substitute the `\item` of the respective level. The list environments that `outline` uses for the individual levels are configurable. Compared to `asciilist`, the package has the slight drawback that it uses L^AT_EX-ish enumeration macros rather than nice symbols. Moreover, the package does not offer convenience functionality similar to that described in Section 2 of this documentation.

6 Implementation

The `etoolbox` package is used for processing comma-separated lists easily.

```
1 \RequirePackage{etoolbox}
```

The `trimspaces` package is used for trimming leading spaces in a robust manner.

```
2 \RequirePackage{trimspaces}
```

6.1 The `AsciiList` Environment

`AsciiList` The `AsciiList` [*environments*] {*item-chars*} creates an environment in which nested lists can be typeset without much L^AT_EX interference. The *item-chars* must specify of comma-separated list of characters. The special value `auto` makes the environment try to auto-detect the *item-chars*. The *environments* specifies an optional, comma-separated list of environments to be used to create the lists at the individual levels.

```
3 \newenvironment{AsciiList}[2] []{%
4   \bgroup
```

Setup the environment by storing the list *environments* to use and – most importantly – setting up the newline character such that it scans for the *item-chars* to find new list items.

```
5   \ifstrepty{#1}%
6     {\let\asclst@listenv=\asclst@defaultenvs}%
7     {\def\asclst@listenv{#1}}%
```

Convert the given *item-chars* to an internal list of `etoolbox`, because this makes working with the list easier.

```
8   \gdef\asclst@itemchars{%
9     \ifstrequal{#2}{auto}%
10    {\asclst@autocharstrue}%
11    {\asclst@autocharsfalse\forcsvlist{\listadd\asclst@itemchars}{#2}}%
12    \asclst@nlsetup\asclst@newline
```

Initialize to nesting level 0. And make the macro `\UP` available for switching to an upper level.

```
13   \global\asclst@curlevel=0\relax%
14   \let\UP=\asclst@levelsup%
15   \let\UPT0=\asclst@levelsupto%
```

Ensure that the scanning for an item char starts with the very first line of the environment.

```
16   \asclst@ifnextnewline{}\{\asclst@newline}%
17 }{%
```

Ensure that all remaining open list environments are closed before the end of the `AsciiList` environment.

```
18   \asclst@changelistlevel{\asclst@curlevel}{0}%
19   \asclst@restorenewline%
20   \egroup}
```

`\AsciiListFromFile` The `\AsciiListFromFile[environments]{item-chars}{file-name}` macro produces a result like the `AsciiList` environment does, but takes the content of the list from *file-name*.

```
21 \newcommand\AsciiListFromFile[3] []{%
22   \AsciiList[#1]{#2}%
```

The following is a bit of a hack to ensure that there is some parameter to `\asclst@newline` even if the input file ends with a newline.

```
23   \everyeof{\relax}%
```

Now we input the file using the `\@@input` primitive, because this primitive can be expanded via `\expandafter` such that the `\asclst@newline` can parse the first line of the file already.

```
24   \expandafter\asclst@newline\@@input #3\relax
25   \endAsciiList}
```

`\AsciiListFromFiles` The `\AsciiListFromFiles[environments]{item-chars}{file-list}` macro produces a result like the `AsciiList` environment does, but takes the content of the list from the comma-separated *file-list*.

```
26 \newcommand\AsciiListFromFiles[3] []{%
27   \AsciiList[#1]{#2}%
```

We do the same here as for the `\AsciiListFromFile` macro, just in a loop over the *file-list*. Note that `\dolistloop` is to be avoided here, because there is quite some chance that the `\do` macro gets redefined in the included code.

```
28   \everyeof{\relax}%
29   \def\asclst@load##1{\expandafter\asclst@newline\@@input ##1\relax}%
30   \forcsvlist{\asclst@load}{#3}%
31   \endAsciiList}
```

6.1.1 Handling of Line Breaks

For our code, line breaks are important to be tracked, because an item-indicating char at the beginning of a line (i.e., after a line break) is crucial.

```
32 {\catcode'\^^M=\active%
```

`\asclst@checknext` The `\asclst@checknext{first-char}{item-char}` checks whether *first-char* (used with the first char of a line after a line break) is equal to a given *item-char* (a character that indicates the beginning of a new list item) and, if so, sets the counter `\asclst@newlevel` to the value of `\@tempcntb`. This is used in `\asclst@newline` to store the index of a found *item-char* in the list of *item-chars* of the `AsciiList` environment.

```
33   \gdef\asclst@checknext#1#2{%
34     \ifstrequal{#1}{#2}{\asclst@newlevel=\@tempcntb}{}}%
```

`\asclst@newline` The `\asclst@newline{first-char}` macro is executed whenever a newline character occurs in the `AsciiList` environment. The *first-char* then is the first character (or, rather, token) after the newline. Important in this macro: all lines

must end with a percent char, to not introduce new newline chars in the macro itself (this would yield an endless recursion).

```
35 \gdef\asclst@newline#1{%
```

First, we find out whether the next char is in list $\langle item-chars \rangle$ (i.e., in $\backslash asclst@itemchars$) and return the position in the list in $\backslash asclst@newlevel$ (or 0 if not found).

```
36 \asclst@newlevel=0\@tempcntb=0\relax%
37 \forlistloop{\advance\@tempcntb by 1\asclst@checknext{#1}}%
38 {\asclst@itemchars}%
```

If the next char is not in the $\langle item-chars \rangle$, but the `AsciiList` was given the `auto` parameter for $\langle item-chars \rangle$, then we check whether we can automatically determine the character for a new nesting level. The char for this new nesting level is then (globally) added to the list of known $\langle item-chars \rangle$ in $\backslash asclst@itemchars$. Note that in the following code, the value of $\backslash @tempcntb$ still is the length of the $\backslash asclst@itemchars$ list.

```
39 \ifnum\asclst@newlevel=0\ifasclst@autochars%
40 \ifinlist{#1}{\asclst@autocharlist}{%
41 \listgadd\asclst@itemchars{#1}%
42 \asclst@newlevel=\@tempcntb%
43 \advance\asclst@newlevel by 1\relax%
44 }{ }%
45 \fi\fi%
```

If we found a character from $\langle item-chars \rangle$, then we ensure to change to the nesting level of this character (which is in $\backslash asclst@newlevel$) and then trigger a new $\backslash item$.

```
46 \ifnum\asclst@newlevel>0\relax%
47 \def\asclst@@do{%
48 \asclst@changelistlevel{\asclst@curlevel}{\asclst@newlevel}%
49 \ifhmode\unskip\space\fi\asclst@@item}%
50 \else%
```

If no character from $\langle item-chars \rangle$ is found, we check whether the newline causing $\backslash asclst@newline$ to be invoked is followed by another newline character. In this case, we insert a $\backslash par$.

```
51 \def\asclst@@tmpone{#1}\def\asclst@@test{^^M}%
52 \ifx\asclst@@test\asclst@@tmpone%
53 \def\asclst@@do{\par #1}%
```

Otherwise, we just use a $\backslash space$ for the newline character and flush out the token $\#1$ that we captured after the newline character.

```
54 \else\def\asclst@@do{\space #1}\fi%
55 \fi\asclst@@do}%
```

$\backslash asclst@ifnextnewline$ The $\backslash asclst@ifnextnewline\{\langle iftrue \rangle\}\{\langle iffalse \rangle\}$ macro checks whether the next character is a newline. If the check succeeds, then the macro expands to $\langle iftrue \rangle$. Otherwise, the macro expands to $\langle iffalse \rangle$.

```
56 \gdef\asclst@ifnextnewline{\@ifnextchar^^M}
```

The following ends the group with active line break catcode.

```
57 }
```

6.1.2 Level-Changing Macros

`\asclst@curlevel` We use the `\asclst@curlevel` counter to capture the current nesting depth of list environments within an `AsciiList`. We also use a counter for changing the level to a new one.

```
58 \newcount\asclst@curlevel
59 \newcount\asclst@newlevel
```

`\asclst@changelistlevel` The `\asclst@changelistlevel{<from>}{<to>}` changes the list nesting level from level `<from>` (a number) to level `<to>` (a number), by issuing the right number of `\begin` or `\end` environments.

```
60 \newcommand\asclst@changelistlevel[2]{%
61   \def\asclst@@envchanger{%
62     \ifnum#2<#1\relax
```

If `<to>` < `<from>`, then we must change to a lower list nesting level. We do this by inserting `<from>` – `<to>` `\end`-environments, which we store in `\asclst@@envchanger`.

```
63   \def\asclst@@last{%
```

First, we collect all affected environment names from the given `<environments>` parameter to `AsciiList` (which at this point is in `\asclst@listenv`): We take all those from list index `<to>` + 1 until list index `<from>`, in reverse order (hence `\preto`), which are actually in the list.

```
64   \@tempcnta=0\relax
65   \@tempcntb=#2 \advance\@tempcntb by 1\relax
66   \def\do##1{\advance\@tempcnta by 1\relax
67     \ifnum\@tempcnta<\@tempcntb\else
68       \ifnum\@tempcnta>#1\else
69         \preto\asclst@@envchanger{\asclst@end ##1<>\@undefined}\fi\fi
70     \def\asclst@@last{##1}}%
71   \expandafter\docsvlist\expandafter{\asclst@listenv}%
```

Second, for all indices from `<to>` + 1 until `<from>` that are *not* in the `<environments>` list, we just take the last list entry (stored in `\asclst@@last` by the above code) and repeat it sufficiently often, i.e., from `max(len<environments>, <from>) + 1` until `<to>` times.

```
72   \advance\@tempcnta by 1\ifnum\@tempcnta<\@tempcntb
73     \@tempcnta=\@tempcntb\fi
74   \loop \ifnum\@tempcnta>#1\else
75     \advance\@tempcnta by 1%
76     \preto\asclst@@envchanger{\noexpand
77       \asclst@end\expandonce\asclst@@last<>\noexpand\@undefined}%
78   \repeat%
79   \else\ifnum#2>#1\relax%
```

The following does the same as the above, except that: `\begin` instead of `\end` of environments are collected; they are collected in the ordering as in `\environments` (hence `\appto`); and entries are collected from `\from` + 1 to `\to`.

```

80 \def\asclst@@last{%
81 \@tempcnta=0\relax
82 \@tempcntb=#1\relax\advance\@tempcntb by 1\relax
83 \def\do##1{\advance\@tempcnta by 1\relax
84 \ifnum\@tempcnta<\@tempcntb\else
85 \ifnum\@tempcnta>#2\else
86 \appto\asclst@@envchanger{\asclst@begin ##1<\undefined}\fi\fi
87 \def\asclst@@last{##1}}%
88 \expandafter\docsvlist\expandafter{\asclst@listenv}%
89 \advance\@tempcnta by 1\ifnum\@tempcnta<\@tempcntb
90 \@tempcnta=\@tempcntb\fi
91 \loop \ifnum\@tempcnta>#2\else
92 \advance\@tempcnta by 1%
93 \eappto\asclst@@envchanger{%
94 \noexpand\asclst@begin
95 \expandonce\asclst@@last<\noexpand\undefined}%
96 \repeat%
97 \fi\fi%

```

Update the current level to the new value, `\to`. Then write out the begin/end environments collected in `\asclst@@envchanger`.

```

98 \global\asclst@curlevel=#2%
99 \asclst@@envchanger}

```

`\asclst@levelsup` The `\asclst@levelsup[<number>]` macro allows switching to a nesting level that is *<number>* levels upwards (default: *<number>* = 1). In the `AsciiList` environment, this macro is accessible via the `\UP` command.

```

100 \newcommand*\asclst@levelsup[1][1]{%
101 \asclst@newlevel=\asclst@curlevel
102 \advance\asclst@newlevel by-#1\relax
103 \asclst@changelistlevel{\asclst@curlevel}{\asclst@newlevel}}

```

`\asclst@levelsupto` The `\asclst@levelsupto{<number>}` macro allows switching to list nesting level *<number>*. In the `AsciiList` environment, this macro is accessible via the `\UPTO` command.

```

104 \newcommand*\asclst@levelsupto[1]{%
105 \ifnum\asclst@curlevel<#1\relax
106 \PackageError{asciilist}{Cannot change level downwards!}{}%
107 \else
108 \asclst@changelistlevel{\asclst@curlevel}{#1}%
109 \fi}

```

6.2 The AsciiDocList Environment

`AsciiDocList` The `AsciiDocList[<environments>]` creates an environment in which nested lists can be typeset without much L^AT_EX interference. The *<environments>* must be a

comma-separated list of “ $\langle char \rangle = \langle environment \rangle$ ” entries.

```
110 \newenvironment{AsciiDocList}[1] [] {%
111   \bgroup
```

Modify the newline character to scan for characters that trigger items.

```
112   \def\asclst@curnestlvl{%
113   \asclst@nlsetup\asclst@docnewline%
```

Now setup the characters that trigger items.

```
114   \def\asclst@levelchrs{%
115   \def\do##1{\asclst@parsechmapentry##1\@undefined}%
116   \ifstrempy{#1}%
117     {\expandafter\docsvlist\expandafter{\asclst@docdefaultenvs}}%
118     {\docsvlist{#1}}%
```

And make the macro \UPTO available for switching to an upper level.

```
119   \let\UPTO=\asclst@changedoclistlevel%
```

Ensure that the scanning for an item char starts with the very first line of the environment.

```
120   \asclst@ifnextnewline{}\asclst@docnewline}%
121 }{%
```

Ensure that all remaining open list environments are closed before the end of the AsciiDocList environment.

```
122   \asclst@changedoclistlevel{%
123   \asclst@restorenewline%
124   \egroup}
```

\AsciiDocListFromFile The `\AsciiDocListFromFile[$\langle environments \rangle$]{ $\langle file-name \rangle$ }` macro produces a result like the AsciiDocList environment does, but takes the content of the list from $\langle file-name \rangle$.

```
125 \newcommand\AsciiDocListFromFile[2] [] {%
126   \AsciiDocList[#1]%
```

The following is a bit of a hack to ensure that there is some parameter to `\asclst@docnewline` even if the input file ends with a newline.

```
127   \everyeof{\relax}%
```

Now we input the file using the `\@@input` primitive, because this primitive can be expanded via `\expandafter` such that the `\asclst@docnewline` can parse the first line of the file already.

```
128   \expandafter\asclst@docnewline\@@input #2\relax
129   \endAsciiDocList}
```

\AsciiDocListFromFiles The `\AsciiDocListFromFiles[$\langle environments \rangle$]{ $\langle file-list \rangle$ }` macro produces a result like the AsciiDocList environment does, but takes the content of the list from the comma-separated $\langle file-list \rangle$.

```
130 \newcommand\AsciiDocListFromFiles[2] [] {%
131   \AsciiDocList[#1]%
```

We do the same here as for the `\AsciiDocListFromFile` macro, just in a loop over the `\file-list`. Note that `\dolistloop` is to be avoided here, because there is quite some chance that the `\do` macro gets redefined in the included code.

```

132 \everyeof{\relax}%
133 \def\asclst@load##1{\expandafter\asclst@docnewline\@input##1\relax}%
134 \forcsvlist{\asclst@load}{#2}%
135 \endAsciiDocList}

```

6.2.1 Handling of Line Breaks

For our code, line breaks are important to be tracked, because an item-indicating char at the beginning of a line (i.e., after a line break) is crucial.

```

136 {\catcode'\^^M=\active%

```

`\asclst@docnewline` The `\asclst@docnewline{<first-char>}` macro is executed whenever a newline character occurs in the `AsciiDocList` environment. The `<first-char>` then is the first character (or, rather, token) after the newline.

```

137 \gdef\asclst@docnewline{\asclst@docnewline@i{}}%

```

`\asclst@docnewline@i` The `\asclst@docnewline@i{<char-seq>}{<test-char>}` macro recursively collects characters (tokens) until a token is found that is not in `\asclst@levelchrs`.

```

138 \gdef\asclst@docnewline@i#1#2{%
139 \ifinlist{#2}{\asclst@levelchrs}%
140 {\asclst@docnewline@i{#1#2}}%
141 {\asclst@docnewline@ii{#1}{#2}}}%

```

`\asclst@docnewline@ii` The `\asclst@docnewline@ii{<char-seq>}{<next>}` macro creates a new list item (in the right environment), if `<char-seq>` is non-empty. The `<next>` character must not be in the `\asclst@levelchrs` and is inserted after the new item.

```

142 \gdef\asclst@docnewline@ii#1#2{%
143 \ifstrempy{#1}{%

```

If `<char-seq>` is empty, this means there was no item char at the beginning of the line. Hence, no item shall be put here, but the `<next>` char must be inserted again. If `<next>` is another newline character, then we insert a `\par` instead.

```

144 \def\asclst@tmpone{#2}\def\asclst@test{^^M}%
145 \ifx\asclst@test\asclst@tmpone%
146 \def\asclst@do{\par #2}%
147 \else%
148 \def\asclst@do{\space #2}\fi%
149 \asclst@do%
150 }{%

```

If `<char-seq>` is non-empty, then change the nesting level to `<char-seq>` and then trigger a new `\item`.

```

151 \asclst@changedoclistlevel{#1}%
152 \ifhmode\unskip\space\fi\asclst@@item #2%
153 }}%

```

The following ends the group with active line break catcode.

```
154 }
```

`\asclst@parsechmapentry` The `\asclst@parsechmap` $\langle char \rangle = \langle env \rangle \backslash @undefined$ macro adds a $\langle char \rangle$ to the list of level characters and defines a mapper macro from the $\langle char \rangle$ to the $\langle env \rangle$.

```
155 \def\asclst@parsechmapentry#1=#2\@undefined{%
156   \listadd\asclst@levelchrs{#1}%
157   \csdef{asclst@levelchr#1}{#2}}
```

6.2.2 Level-Changing Macros

`\asclst@changedoclistlevel` The `\asclst@changedoclistlevel` $\{ \langle to \rangle \}$ changes the list nesting level from level `\asclst@curnestlvl` (a character sequence) to level $\langle to \rangle$ (a character sequence), by issuing the right number of `\begin` or `\end` environments.

```
158 \newcommand\asclst@changedoclistlevel[1]{%
159   \def\asclst@@envchanger{}%
160   \expandafter\asclst@changedoclistlevel@i\asclst@curnestlvl{}{} \@undefined
161   #1{}{} \@undefined
162   \asclst@@envchanger
163   \gdef\asclst@curnestlvl{#1}}
```

`\asclst@changedoclistlevel@i` The `\asclst@changedoclistlevel@i` $\langle ohd \rangle \langle otl \rangle \backslash @undefined \langle nhd \rangle \langle ntl \rangle \backslash @undefined$ macro recursively strips off the common prefixes from the old environment list (with head $\langle ohd \rangle$ and tail $\langle otl \rangle$) and the new environment list (with head $\langle nhd \rangle$ and tail $\langle ntl \rangle$). Afterwards, the macro uses `\asclst@changedoclistlevel@ii` to construct the `\asclst@@envchanger`.

```
164 \def\asclst@changedoclistlevel@i#1#2\@undefined#3#4\@undefined{%
```

If both environment lists have been processed, then $\langle ohd \rangle$ and $\langle nhd \rangle$ are empty and nothing needs to be done.

```
165   \ifstrempy{#1#3}{}{%
```

Otherwise, if the heads are equal, go to the recursive case for further stripping off the common prefix. If the heads differ, proceed with `\asclst@changedoclistlevel@ii`.

```
166   \ifstrequal{#1}{#3}%
167     {\asclst@changedoclistlevel@i#2{} \@undefined#4{} \@undefined}%
168     {\asclst@changedoclistlevel@ii#1#2{} \@undefined#3#4{} \@undefined}%
169   }}
```

`\asclst@changedoclistlevel@ii` The `\asclst@changedoclistlevel@ii` $\langle ohd \rangle \langle otl \rangle \backslash @undefined \langle nhd \rangle \langle ntl \rangle \backslash @undefined$ macro recursively decomposes the old environment list (with head $\langle ohd \rangle$ and tail $\langle otl \rangle$) and the new environment list (with head $\langle nhd \rangle$ and tail $\langle ntl \rangle$). In the process, the macro constructs the `\asclst@@envchanger` macro by prepending a closing environment for $\langle ohd \rangle$ and appending an opening environment for $\langle nhd \rangle$. After the construction, the change of environment is performed by expanding `\asclst@@envchanger`.

```
170 \def\asclst@changedoclistlevel@ii#1#2\@undefined#3#4\@undefined{%
```


If both environment lists have been processed, then $\langle ohd \rangle$ and $\langle nhd \rangle$ are empty and nothing needs to be done further.

```
171 \ifstrempy{#1#3}{}{%
```

Otherwise, first process $\langle ohd \rangle$ by prepending a closing environment.

```
172 \ifstrempy{#1}{}{%
```

```
173 \epreto\asclst@@envchanger{%
```

```
174 \noexpand\asclst@end
```

```
175 \csuse{asclst@levelchr@#1}<>\noexpand\@undefined}}%
```

Second, process $\langle nhd \rangle$ by appending an opening environment.

```
176 \ifstrempy{#3}{}{%
```

```
177 \eappto\asclst@@envchanger{%
```

```
178 \noexpand\asclst@begin
```

```
179 \csuse{asclst@levelchr@#3}<>\noexpand\@undefined}}%
```

Third, recurse with the remainders of the lists.

```
180 \asclst@changedoclistlevel@ii#2{}\@undefined#4{}\@undefined}}
```

6.3 Shared Code between AsciiList and AsciiDocList

$\backslash\text{asclst@begin}$ The macro $\backslash\text{asclst@begin}\langle env \rangle\langle opt \rangle\langle ignored \rangle\backslash\text{@undefined}$ corresponds to L^AT_EX's $\backslash\text{begin}\langle opt \rangle$ and $\backslash\text{asclst@end}\langle env \rangle\langle ignored \rangle\langle ignored \rangle\backslash\text{@undefined}$ corresponds to L^AT_EX's $\backslash\text{end}$. If $\langle env \rangle$ was not registered by $\backslash\text{AsciiListRegisterEnv}$, then the macros are even identical to $\backslash\text{begin}$ and $\backslash\text{end}$. Otherwise, the macros expand to the $\langle begin \rangle$ and $\langle end \rangle$ code registered for $\langle env \rangle$. In addition, $\backslash\text{asclst@begin}$ sets up the $\backslash\text{asclst@@item}$ macro that is used to create a list item.

```
181 \long\def\asclst@begin#1<#2>#3\@undefined{%
```

```
182 \ifinlist{#1}{\asclst@registeredenvs}%
```

```
183 {\bgroup
```

```
184 \ifstrempy{#2}%
```

```
185 {\@nameuse{asclst@env@#1@begin}}%
```

```
186 {\@nameuse{asclst@env@#1@begin}[#2]}%
```

```
187 \edef\asclst@@item{\csexpandonce{asclst@env@#1@item}}%
```

```
188 {%
```

```
189 \ifstrempy{#2}%
```

```
190 {\begin{#1}}%
```

```
191 {\begin{#1}[#2]}%
```

```
192 \def\asclst@@item{\item}}}
```

```
193 \long\def\asclst@end#1<#2>#3\@undefined{%
```

```
194 \ifinlist{#1}{\asclst@registeredenvs}%
```

```
195 {\@nameuse{asclst@env@#1@end}\egroup}%
```

```
196 {\end{#1}}}
```

For our code, line breaks are important to be tracked, because an item-indicating char at the beginning of a line (i.e., after a line break) is crucial.

```
197 {\catcode'\^M=\active%
```

`\asclst@nlsetup` The `\asclst@nlsetup{<nl-macro>}` macro sets up the newline character such that it runs the `<nl-macro>` upon every newline.

```
198 \gdef\asclst@nlsetup#1{%
```

Save the current newline, make it active, and set the newline to expand to our `<nl-macro>` macro, which does the main job of this whole package at every newline.

```
199 \let\asclst@orignewline=^M%
```

```
200 \catcode'\^M\active%
```

```
201 \let^M=#1}%
```

`\asclst@restorenewline` The `\asclst@restorenewline` restores the meaning of the newline character that is changed by `\asclst@nlsetup`.

```
202 \gdef\asclst@restorenewline{\let^M=\asclst@orignewline}%
```

The following ends the group with active line break catcode.

```
203 }
```

6.4 Configuration

`\AsciiListSetAutochars` The `\asclst@autochars` conditional is used to store whether the `<item-chars>` are to be auto-detected. In this case, the `<item-chars>` are attempted to be derived from the candidates in `\asclst@autocharlist`. The `\AsciiListSetAutochars{<chars>}` allows one to specify a user-defined list of characters for the auto-detection.

```
204 \newif\ifasclst@autochars
```

```
205 \newcommand*\AsciiListSetAutochars[1]{%
```

```
206 \def\asclst@autocharlist{}
```

```
207 \forcsvlist{\listadd\asclst@autocharlist}{#1}}
```

```
208 \AsciiListSetAutochars{-,*,+}
```

`\AsciiListSetEnvironments` The `\AsciiListSetEnvironments{<environments>}` macro sets the default list environment. This default is used when the optional `<environments>` argument is not given to `AsciiList`.

`\asclst@defaultenvs`

```
209 \newcommand*\AsciiListSetEnvironments[1]{%
```

```
210 \def\asclst@defaultenvs{#1}}
```

```
211 \AsciiListSetEnvironments{itemize}
```

`\AsciiDocListSetEnvironments` The `\AsciiDocListSetEnvironments{<environments>}` macro sets the default `<environments>` argument for the `AsciiDocList` environment.

`\asclst@docdefaultenvs`

```
212 \newcommand*\AsciiDocListSetEnvironments[1]{%
```

```
213 \def\asclst@docdefaultenvs{#1}}
```

```
214 \AsciiDocListSetEnvironments{*=itemize,+=enumerate,;=Description}
```

`\NewAsciiListEnv` The `\NewAsciiListEnv[<environments>]{<item-chars>}{<envname>}` macro creates a new environment named `<envname>`. The use of this environment then is equivalent to using `AsciiList[<environments>]{<item-chars>}`. Moreover, the macro creates a new macro `\<envname>FromFile{<file-name>}`, which is equivalent to `\AsciiListFromFile[<environments>]{<item-chars>}{<file-name>}`. Analogously, the macro `\<envname>FromFiles` is defined.

```

215 \newcommand*\NewAsciiListEnv[3][\%
216   \newenvironment{#3}%
217     {\begin{AsciiList}[#1]{#2}}%
218     {\end{AsciiList}}%
219   \csdef{#3FromFile}##1{\AsciiListFromFile[#1]{#2}{##1}}%
220   \csdef{#3FromFiles}##1{\AsciiListFromFiles[#1]{#2}{##1}}%
221 }

```

`\NewAsciiDocListEnv` The `\NewAsciiDocListEnv[environments]{envname}` macro creates a new environment named *envname*. The use of this environment then is equivalent to using `AsciiDocList[environments]`. Moreover, the macro creates a new macro `\<envname>FromFile{<file-name>}`, which is defined to be equivalent to `\AsciiDocListFromFile[environments]{<file-name>}`. Analogously, the macro `\<envname>FromFiles` is defined.

```

222 \newcommand*\NewAsciiDocListEnv[2][\asclst@docdefaultenvs]{%
223   \newenvironment{#2}%
224     {\begin{AsciiDocList}[#1]}%
225     {\end{AsciiDocList}}%
226   \csdef{#2FromFile}##1{\AsciiDocListFromFile[#1]{##1}}%
227   \csdef{#2FromFiles}##1{\AsciiDocListFromFiles[#1]{##1}}%
228 }

```

`\AsciiListRegisterEnv` The `\AsciiListRegisterEnv{<envname>}{<begin>}{<end>}{<item>}` macro registers a new environment name for use with `AsciiList` and `AsciiDocList`. After such a registration, one may use the *envname* as an element in the *environments* parameter to `AsciiList` or `AsciiDocList`. The *begin*, *end*, and *item* code is used whenever such a newly defined environment is begun or ended, or a list entry is started, respectively. Note that by using the `\AsciiListRegisterEnv`, no real L^AT_EX environment is created. All registered environments are collected in the `\asclst@registeredenvs` list.

```

229 \def\asclst@registeredenvs{}
230 \newcommand*\AsciiListRegisterEnv[4]{%
231   \listadd\asclst@registeredenvs{#1}%
232   \csdef{asclst@env@#1@begin}{#2}%
233   \csdef{asclst@env@#1@end}{#3}%
234   \csdef{asclst@env@#1@item}{#4}}

```

`\AsciiListRegisterDescEnv` The `\AsciiListRegisterDescEnv{<envname>}` is a macro that can be used as a shorthand for `\AsciiListRegisterEnv` to declare existing description environments. The environment *envname* must exist and must be a description environment, i.e., one whose entries are specified via `\item[text]`.

```

235 \newcommand*\AsciiListRegisterDescEnv[1]{%
236   \AsciiListRegisterEnv{#1}{\csuse{#1}}{\csuse{end#1}}%
237   {\AsciiListEnd0Arg{\item}}}

```

`\AsciiListEndArg` The `\AsciiListEndArg{<command>}` macro is equal to the given *command*, except that the first argument passed to *command* is the remainder of the line in which the macro is used.

```

238 {\catcode'\^^M=\active%
239 \gdef\AsciiListEndArg#1#2^^M{%

```

Note the line break at the end of the following line. This line break is important, because the definition of `\AsciiListEndArg` swallows one line break. By having the line break below, we essentially re-insert the line break, such that `AsciiList` or `AsciiDocList` can use it again to check for list items in the subsequent line.

```

240 \begingroup%
241 \def\asclst@@result{#2}\trim@pre@space@in{\asclst@@result}%
242 \edef\asclst@@result{%
243 \endgroup\unexpanded{#1}{\expandonce{\asclst@@result}}}%
244 \asclst@@result
245 }%

```

`\AsciiListEnd0Arg` The `\AsciiListEnd0Arg{<command>}` macro constitutes the counterpart to the `\AsciiListEndArg` macro for the case of a `<command>` that takes an *optional* argument (like the `\item` of a `description` environment).

```

246 \gdef\AsciiListEnd0Arg#1#2^^M{%
247 \begingroup%
248 \def\asclst@@result{#2}\trim@pre@space@in{\asclst@@result}%
249 \edef\asclst@@result{%
250 \endgroup\unexpanded{#1}{\expandonce{\asclst@@result}}}%
251 \asclst@@result
252 }%
253 }

```

6.5 Pre-Defined List Environments

In the following, we define some environment names that allow one to use sectioning commands for list items. When using these environments, one should be aware that list entries in these environments must fit into a single line (i.e., everything after a line break is not put into the argument of the sectioning command).

```

254 \AsciiListRegisterEnv{chapter}{-}{-}{\AsciiListEndArg{\chapter}}
255 \AsciiListRegisterEnv{section}{-}{-}{\AsciiListEndArg{\section}}
256 \AsciiListRegisterEnv{subsection}{-}{-}{\AsciiListEndArg{\subsection}}
257 \AsciiListRegisterEnv{subsubsection}{-}{-}{\AsciiListEndArg{\subsubsection}}
258 \AsciiListRegisterEnv{section*}{-}{-}{\AsciiListEndArg{\section*}}
259 \AsciiListRegisterEnv{subsection*}{-}{-}{\AsciiListEndArg{\subsection*}}
260 \AsciiListRegisterEnv{subsubsection*}{-}{-}{\AsciiListEndArg{\subsubsection*}}
261 \AsciiListRegisterEnv{paragraph}{-}{-}{\AsciiListEndArg{\paragraph}}
262 \AsciiListRegisterEnv{description}{-}{-}{\AsciiListEndArg{\description}}
263 \AsciiListRegisterEnv{compactdesc}{-}{-}{\AsciiListEndArg{\compactdesc}}

```

To simplify the use of the `description` environment as well as other common description environments (the packages defining these environments need not be loaded until the environments are actually used).

```

264 \AsciiListRegisterDescEnv{description}
265 \AsciiListRegisterDescEnv{compactdesc}

```


v1.8		AsciiList: Improved first-line handling.	9
\AsciiListFromFile: Added this macro	10	v2.1	
v1.8b		General: Documentation of \UPTO .	5
\AsciiListFromFiles: Added this macro	10	Documentation of \UP and \UPTO	4
\asclst@newline: Robustified by not using \do	11	AsciiDocList: Add \UPTO to explicitly change levels upwards	14
\NewAsciiListEnv: Registered list environments can now be loaded from files	18	v2.1b	
v2.0		\AsciiListEndArg: Robustified ignoring initial spaces	20
AsciiDocList: Added AsciiDocList environment	14	\AsciiListEndOArg: Robustified against nested square brackets	20
v2.0a		\asclst@BreakingDescItem: Robustified against nested square brackets	21
AsciiDocList: Improved first-line handling.	14	AsciiList: Eliminated expansion of optional argument	9

Index

Symbols

\@@input	24, 29, 128, 133
\@ifnextchar	56
\@nameuse	185, 186, 195
\@tempcnta	64, 66, 67, 68, 72, 73, 74, 75, 81, 83, 84, 85, 89, 90, 91, 92
\@tempcntb	34, 36, 37, 42, 65, 67, 72, 73, 82, 84, 89, 90
\@undefined	69, 77, 86, 95, 115, 155, 160, 161, 164, 167, 168, 170, 175, 179, 180, 181, 193
\^	32, 136, 197, 200, 238

A

\active	32, 136, 197, 200, 238
\advance	37, 43, 65, 66, 72, 75, 82, 83, 89, 92, 102
\appto	86
\AsciiDocList	126, 131
AsciiDocList (environment)	4, 110
\AsciiDocListFromFile	4, 125, 226
\AsciiDocListFromFiles	5, 130, 227

\AsciiDocListSetEnvironments	7, 212
\AsciiList	22, 27
AsciiList (environment)	1, 3
\AsciiListEndArg	6, 238, 254, 255, 256, 258, 259, 260, 262, 263
\AsciiListEndOArg	6, 237, 246, 267, 269
\AsciiListFromFile	2, 21, 219
\AsciiListFromFiles	2, 26, 220
\AsciiListRegisterDescEnv	235, 264, 265
\AsciiListRegisterEnv	6, 229, 236, 254, 255, 256, 257, 259, 260, 261, 263, 266, 268
\AsciiListSetAutochars	3, 204
\AsciiListSetEnvironments	6, 209
\asclst@@do	47, 53, 54, 55, 146, 148, 149
\asclst@@envchanger	61, 69, 76, 86, 93, 99, 159, 162, 173, 177
\asclst@@item	49, 152, 187, 192
\asclst@@last	63, 70, 77, 80, 87,

95
 \asclst@load .. 29, 30, 133, 134
 \asclst@result .. 241, 242, 243,
 244, 248, 249, 250, 251
 \asclst@test .. 51, 52, 144, 145
 \asclst@tmpone 51, 52, 144, 145
 \asclst@autocharlist . 40, 204
 \asclst@autocharsfalse 11
 \asclst@autocharstrue 10
 \asclst@begin .. 86, 94, 178, 181
 \asclst@BreakingDescItem . 267,
 269, 270
 \asclst@changedoclistlevel 119,
 122, 151, 158
 \asclst@changedoclistlevel@i
 160, 164
 \asclst@changedoclistlevel@ii
 168, 170
 \asclst@changelistlevel 18, 48,
 60, 103, 108
 \asclst@checknext 33, 37
 \asclst@curlevel 13, 18, 48, 58,
 98, 101, 103, 105, 108
 \asclst@curnestlvl 112, 160, 163
 \asclst@defaultenvs ... 6, 209
 \asclst@docdefaultenvs ... 117,
 212, 222
 \asclst@docnewline .. 113, 120,
 128, 133, 137
 \asclst@docnewline@i . 137, 138
 \asclst@docnewline@ii . 141, 142
 \asclst@end ... 69, 77, 174, 181
 \asclst@ifnextnewline .. 16, 56,
 120
 \asclst@itemchars . 8, 11, 38, 41
 \asclst@levelchrs 114, 139, 156
 \asclst@levelsup 14, 100
 \asclst@levelsupto ... 15, 104
 \asclst@listenv 6, 7, 71, 88
 \asclst@newlevel 34, 36, 39, 42,
 43, 46, 48, 59, 101, 102, 103
 \asclst@newline 12, 16, 24, 29, 35
 \asclst@nlsetup ... 12, 113, 198
 \asclst@orignewline .. 199, 202
 \asclst@parsechmapentry .. 115,
 155

\asclst@registeredenv 229
 \asclst@registeredenvs ... 182,
 194, 229, 231
 \asclst@restorenewline 19, 123,
 202

B

\begin 190, 191, 217, 224
 \begingroup 240, 247
 \bgroup 4, 111, 183

C

\catcode .. 32, 136, 197, 200, 238
 \chapter 254
 \compactdesc 268
 \csdef ... 157, 219, 220, 226, 227,
 232, 233, 234
 \csexpandonce 187
 \csuse 175, 179, 236

D

\description 266
 \do 66, 83, 115
 \docsvlist 71, 88, 117, 118

E

\eappto 93, 177
 \egroup 20, 124, 195
 \else 50, 54, 67, 68, 74, 79, 84, 85,
 91, 107, 147
 \end 196, 218, 225
 \endAsciiDocList 129, 135
 \endAsciiList 25, 31
 \endcompactdesc 268
 \enddescription 266
 \endgroup 243, 250
 environments:
 AsciiDocList 4, 110
 AsciiList 1, 3
 \epreto 76, 173
 \everyeof 23, 28, 127, 132
 \expandafter 24, 29, 71, 88, 117,
 128, 133, 160
 \expandonce ... 77, 95, 243, 250

F

\fi . 45, 49, 54, 55, 69, 73, 86, 90,
 97, 109, 148, 152

\forcsvlist ... 11, 30, 134, 207
\forlistloop 37

G

\global 13, 98

I

\ifasclst@autochars .. 39, 204
\ifhmode 49, 152
\ifinlist 40, 139, 182, 194
\ifnum . 39, 46, 62, 67, 68, 72, 74,
79, 84, 85, 89, 91, 105
\ifstrempy 5, 116, 143, 165, 171,
172, 176, 184, 189
\ifstrequal 9, 34, 166
\ifx 52, 145
\item 192, 237, 270

L

\leavevmode 270
\let 6, 14, 15, 119, 199, 201, 202
\listadd 11, 156, 207, 231
\listgadd 41
\long 181, 193
\loop 74, 91

N

\NewAsciiDocListEnv ... 7, 222
\NewAsciiListEnv 6, 215
\newcount 58, 59
\newif 204

\noexpand 76, 77, 94, 95, 174, 175,
178, 179

P

\PackageError 106
\par 53, 146
\paragraph 263
\preto 69

R

\relax 13, 23, 24, 28,
29, 36, 43, 46, 62, 64, 65,
66, 79, 81, 82, 83, 102, 105,
127, 128, 132, 133
\repeat 78, 96
\RequirePackage 1, 2

S

\section 255, 259
\space 49, 54, 148, 152
\subsection 256, 260
\subsubsection 258, 262

T

\trim@pre@space@in ... 241, 248

U

\unexpanded 243, 250
\unskip 49, 152
\UP 4, 14
\UPTO 4, 5, 15, 119