The asciilist package*

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

March 6, 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 v2.1, dated 2016/03/05. 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 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 `\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.

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

`\UP`
`\UPTO`

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

`AsciiDocList` 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).

`\AsciiDocListFromFile` 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
\UPTO 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.

```
\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*⟩}
\AsciiListEndArg 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

\AsciiListEndOArg 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{`⟨*environments*⟩`}` 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 LaTeX 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 LaTeX-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 LaTeX-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.

```
\RequirePackage{etoolbox}
```

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

```
\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 LaTeX 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.

```
\newenvironment{AsciiList}[2][\asclst@defaultenvs]{%
  \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.

```
  \edef\asclst@listenv{#1}%
```

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

```
  \gdef\asclst@itemchars{}%
  \ifstrequal{#2}{auto}%
    {\asclst@autocharstrue}%
    {\asclst@autocharsfalse\forcsvlist{\listadd\asclst@itemchars}{#2}}%
  \asclst@nlsetup\asclst@newline
```

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

```
  \global\asclst@curlevel=0\relax%
  \let\UP=\asclst@levelsup%
  \let\UPTO=\asclst@levelsupto%
```

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

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

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

```
  \asclst@changelistlevel{\asclst@curlevel}{0}%
  \asclst@restorenewline%
  \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*⟩.

```
\newcommand\AsciiListFromFile[3][\asclst@defaultenvs]{%
  \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.

```
  \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.

```
  \expandafter\asclst@newline\@@input #3\relax
  \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*⟩.

```
\newcommand\AsciiListFromFiles[3][\asclst@defaultenvs]{%
  \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.

```
  \everyeof{\relax}%
  \def\asclst@@load##1{\expandafter\asclst@newline\@@input ##1\relax}%
  \forcsvlist{\asclst@@load}{#3}%
  \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.

```
{\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.

```
  \gdef\asclst@checknext#1#2{%
    \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).

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

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

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

If the next char is not in the ⟨*item-chars*⟩, but the `AsciiList` was given the `auto` parameter for ⟨*item-chars*⟩, 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 ⟨*item-chars*⟩ in `\asclst@itemchars`. Note that in the following code, the value of `\@tempcntb` still is the length of the `\asclst@itemchars` list.

```
  \ifnum\asclst@newlevel=0\ifasclst@autochars%
    \ifinlist{#1}{\asclst@autocharlist}{%
      \listgadd\asclst@itemchars{#1}%
      \asclst@newlevel=\@tempcntb%
      \advance\asclst@newlevel by 1\relax%
    }{}%
  \fi\fi%
```

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

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

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

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

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

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

`\asclst@ifnextnewline` The `\asclst@ifnextnewline{`⟨*iftrue*⟩`}{`⟨*iffalse*⟩`}` macro checks whether the next character is a newline. If the check succeeds, then the macro expands to ⟨*iftrue*⟩. Otherwise, the macro expands to ⟨*iffalse*⟩.

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

The following ends the group with active line break catcode.

```
}
```

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.

```
\newcount\asclst@curlevel
\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.

```
\newcommand\asclst@changelistlevel[2]{%
  \def\asclst@@envchanger{}%
  \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`.

```
    \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.

```
    \@tempcnta=0\relax
    \@tempcntb=#2 \advance\@tempcntb by 1\relax
    \def\do##1{\advance\@tempcnta by1\relax
      \ifnum\@tempcnta<\@tempcntb\else
        \ifnum\@tempcnta>#1\else
        \preto\asclst@@envchanger{\asclst@end ##1<>\@undefined}\fi\fi
      \def\asclst@@last{##1}}%
    \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.

```
    \advance\@tempcnta by1\ifnum\@tempcnta<\@tempcntb
      \@tempcnta=\@tempcntb\fi
    \loop \ifnum\@tempcnta>#1\else
      \advance\@tempcnta by 1%
      \epreto\asclst@@envchanger{\noexpand
        \asclst@end\expandonce\asclst@@last<>\noexpand\@undefined}%
    \repeat%
  \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*⟩.

```
\def\asclst@@last{}%
\@tempcnta=0\relax
\@tempcntb=#1\relax\advance\@tempcntb by 1\relax
\def\do##1{\advance\@tempcnta by1\relax
  \ifnum\@tempcnta<\@tempcntb\else
    \ifnum\@tempcnta>#2\else
    \appto\asclst@@envchanger{\asclst@begin ##1<>\@undefined}\fi\fi
  \def\asclst@@last{##1}}%
\expandafter\docsvlist\expandafter{\asclst@listenv}%
\advance\@tempcnta by1\ifnum\@tempcnta<\@tempcntb
  \@tempcnta=\@tempcntb\fi
\loop \ifnum\@tempcnta>#2\else
  \advance\@tempcnta by 1%
  \eappto\asclst@@envchanger{%
    \noexpand\asclst@begin
      \expandonce\asclst@@last<>\noexpand\@undefined}%
\repeat%
\fi\fi%
```

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

```
\global\asclst@curlevel=#2%
\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.

```
\newcommand*\asclst@levelsup[1][1]{%
  \asclst@newlevel=\asclst@curlevel
  \advance\asclst@newlevel by-#1\relax
  \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.

```
\newcommand*\asclst@levelsupto[1]{%
  \ifnum\asclst@curlevel<#1\relax
    \PackageError{asciilist}{Cannot change level downwards!}{}%
  \else
    \asclst@changelistlevel{\asclst@curlevel}{#1}%
  \fi}
```

6.2 The `AsciiDocList` Environment

`AsciiDocList` The `AsciiDocList[`⟨*environments*⟩`]` creates an environment in which nested lists can be typeset without much LaTeX interference. The ⟨*environments*⟩ must be a

comma-separated list of "⟨*char*⟩=⟨*environment*⟩" entries.

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

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

```
  \def\asclst@curnestlvl{}%
  \asclst@nlsetup\asclst@docnewline%
```

Now setup the characters that trigger items.

```
  \def\asclst@levelchrs{}%
  \def\do##1{\asclst@parsechmapentry##1\@undefined}%
  \ifstrempty{#1}%
    {\expandafter\docsvlist\expandafter{\asclst@docdefaultenvs}}%
    {\docsvlist{#1}}%
```

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

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

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

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

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

```
  \asclst@changedoclistlevel{}%
  \asclst@restorenewline%
  \egroup}
```

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

```
\newcommand\AsciiDocListFromFile[2][]{%
  \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.

```
  \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.

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

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

```
\newcommand\AsciiDocListFromFiles[2][]{%
  \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.

```
\everyeof{\relax}%
\def\asclst@@load##1{\expandafter\asclst@docnewline\@@input##1\relax}%
\forcsvlist{\asclst@@load}{#2}%
\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.

```
{\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.

```
\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`.

```
\gdef\asclst@docnewline@i#1#2{%
  \ifinlist{#2}{\asclst@levelchrs}%
    {\asclst@docnewline@i{#1#2}}%
    {\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.

```
\gdef\asclst@docnewline@ii#1#2{%
  \ifstrempty{#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.

```
    \def\asclst@@tmpone{#2}\def\asclst@@test{^^M}%
    \ifx\asclst@@test\asclst@@tmpone%
      \def\asclst@@do{\par #2}%
    \else%
      \def\asclst@@do{\space #2}\fi%
    \asclst@@do%
  }{%
```

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

```
    \asclst@changedoclistlevel{#1}%
    \ifhmode\unskip\space\fi\asclst@@item #2%
  }}%
```

The following ends the group with active line break catcode.

```
}
```

`\asclst@parsechmapentry` The `\asclst@parsechmap`⟨*char*⟩`=`⟨*env*⟩`\@undefined` macro adds a ⟨*char*⟩ to the list of level characters and defines a mapper macro from the ⟨*char*⟩ to the ⟨*env*⟩.

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

6.2.2 Level-Changing Macros

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

```
\newcommand\asclst@changedoclistlevel[1]{%
  \def\asclst@@envchanger{}%
  \expandafter\asclst@changedoclistlevel@i\asclst@curnestlvl{}{}\@undefined
    #1{}{}\@undefined
  \asclst@@envchanger
  \gdef\asclst@curnestlvl{#1}}
```

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

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

If both environment lists have been processed, then ⟨*ohd*⟩ and ⟨*nhd*⟩ are empty and nothing needs to be done.

```
  \ifstrempty{#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`.

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

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

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

If both environment lists have been processed, then ⟨*ohd*⟩ and ⟨*nhd*⟩ are empty and nothing needs to be done further.

```
\ifstrempty{#1#3}{}{%
```

Otherwise, first process ⟨*ohd*⟩ by prepending a closing environment.

```
  \ifstrempty{#1}{}{%
    \epreto\asclst@@envchanger{%
      \noexpand\asclst@end
        \csuse{asclst@levelchr@#1}<>\noexpand\@undefined}}%
```

Second, process ⟨*nhd*⟩ by appending an opening environment.

```
  \ifstrempty{#3}{}{%
    \eappto\asclst@@envchanger{%
      \noexpand\asclst@begin
        \csuse{asclst@levelchr@#3}<>\noexpand\@undefined}}%
```

Third, recurse with the remainders of the lists.

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

6.3 Shared Code between `AsciiList` and `AsciiDocList`

`\asclst@begin` `\asclst@end`

The macro `\asclst@begin{`⟨*env*⟩`}<`⟨*opt*⟩`>`⟨*ignored*⟩`\@undefined` corresponds to LaTeX's `\begin[`⟨*opt*⟩`]` and `\asclst@end{`⟨*env*⟩`}<`⟨*ignored*⟩`>`⟨*ignored*⟩`\@undefined` corresponds to LaTeX's `\end`. If ⟨*env*⟩ was not registered by `\AsciiListRegisterEnv`, then the macros are even identical to `\begin` and `\end`. Otherwise, the macros expand to the ⟨*begin*⟩ and ⟨*end*⟩ code registered for ⟨*env*⟩. In addition, `\asclst@begin` sets up the `\asclst@@item` macro that is used to create a list item.

```
\long\def\asclst@begin#1<#2>#3\@undefined{%
  \ifinlist{#1}{\asclst@registeredenvs}%
    {\bgroup
     \ifstrempty{#2}%
       {\@nameuse{asclst@env@#1@begin}}%
       {\@nameuse{asclst@env@#1@begin}[#2]}%
     \edef\asclst@@item{\csexpandonce{asclst@env@#1@item}}}%
    {%
     \ifstrempty{#2}%
       {\begin{#1}}%
       {\begin{#1}[#2]}%
     \def\asclst@@item{\item}}}
\long\def\asclst@end#1<#2>#3\@undefined{%
  \ifinlist{#1}{\asclst@registeredenvs}%
    {\@nameuse{asclst@env@#1@end}\egroup}%
    {\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.

```
{\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.

```
\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.

```
  \let\asclst@orignewline=^^M%
  \catcode`\^^M\active%
  \let^^M=#1}%
```

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

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

The following ends the group with active line break catcode.

```
}
```

6.4 Configuration

\AsciiListSetAutochars \ifasclst@autochars \asclst@autocharlist 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.

```
\newif\ifasclst@autochars
\newcommand*\AsciiListSetAutochars[1]{%
  \def\asclst@autocharlist{}%
  \forcsvlist{\listadd\asclst@autocharlist}{#1}}
\AsciiListSetAutochars{-,*,+}
```

\AsciiListSetEnvironments \asclst@defaultenvs The \AsciiListSetEnvironments{⟨*environments*⟩} macro sets the default list environment. This default is used when the optional ⟨*environments*⟩ argument is not given to AsciiList.

```
\newcommand*\AsciiListSetEnvironments[1]{%
  \def\asclst@defaultenvs{#1}}
\AsciiListSetEnvironments{itemize}
```

\AsciiDocListSetEnvironments \asclst@docdefaultenvs The AsciiDocListSetEnvironments{⟨*environments*⟩} macro sets the default ⟨*environments*⟩ argument for the AsciiDocList environment.

```
\newcommand*\AsciiDocListSetEnvironments[1]{%
  \def\asclst@docdefaultenvs{#1}}
\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.

```
\newcommand*\NewAsciiListEnv[3][\asclst@defaultenvs]{%
  \newenvironment{#3}%
    {\begin{AsciiList}[#1]{#2}}%
    {\end{AsciiList}}%
  \csdef{#3FromFile}##1{\AsciiListFromFile[#1]{#2}{##1}}%
  \csdef{#3FromFiles}##1{\AsciiListFromFiles[#1]{#2}{##1}}%
}
```

`\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.

```
\newcommand*\NewAsciiDocListEnv[2][\asclst@docdefaultenvs]{%
  \newenvironment{#2}%
    {\begin{AsciiDocList}[#1]}%
    {\end{AsciiDocList}}%
  \csdef{#2FromFile}##1{\AsciiDocListFromFile[#1]{##1}}%
  \csdef{#2FromFiles}##1{\AsciiDocListFromFiles[#1]{##1}}%
}
```

`\AsciiListRegisterEnv` `\asclst@registeredenv` 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 LaTeX environment is created. All registered environments are collected in the `\asclst@registeredenvs` list.

```
\def\asclst@registeredenvs{}
\newcommand*\AsciiListRegisterEnv[4]{%
  \listadd\asclst@registeredenvs{#1}%
  \csdef{asclst@env@#1@begin}{#2}%
  \csdef{asclst@env@#1@end}{#3}%
  \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*⟩`]`.

```
\newcommand*\AsciiListRegisterDescEnv[1]{%
  \AsciiListRegisterEnv{#1}{\csuse{#1}}{\csuse{end#1}}%
                           {\AsciiListEndOArg{\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.

```
{\catcode`\^^M=\active%
  \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.

```
    \begingroup%
    \protected@edef\asclst@@result{%
      \endgroup\unexpanded{#1}{\trim@pre@space{#2}}}%
    \asclst@@result
  }%
```

`\AsciiListEndOArg` The `\AsciiListEndOArg{`⟨*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).

```
  \gdef\AsciiListEndOArg#1#2^^M{%
    \begingroup%
    \protected@edef\asclst@@result{%
      \endgroup\unexpanded{#1}[\trim@pre@space{#2}]}%
    \asclst@@result
  }%
}
```

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).

```
\AsciiListRegisterEnv{chapter}{}{}{\AsciiListEndArg{\chapter}}
\AsciiListRegisterEnv{section}{}{}{\AsciiListEndArg{\section}}
\AsciiListRegisterEnv{subsection}{}{}{\AsciiListEndArg{\subsection}}
\AsciiListRegisterEnv{subsubsection}{}{}%
                      {\AsciiListEndArg{\subsubsection}}
\AsciiListRegisterEnv{section*}{}{}{\AsciiListEndArg{\section*}}
\AsciiListRegisterEnv{subsection*}{}{}{\AsciiListEndArg{\subsection*}}
\AsciiListRegisterEnv{subsubsection*}{}{}%
                      {\AsciiListEndArg{\subsubsection*}}
\AsciiListRegisterEnv{paragraph}{}{}{\AsciiListEndArg{\paragraph}}
```

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).

```
\AsciiListRegisterDescEnv{description}
\AsciiListRegisterDescEnv{compactdesc}
```

It often looks less appealing to have the first item of an `itemize` or `enumerate` environment in the same line as the parent item in a description environment. As an alternative to this default behavior of LaTeX, we offer the following `Description`

and `CompactDesc` environments, with which the child items start in a new line (but if the environment starts with text, then this will still be placed in the same line as the description item).

```
\newcommand\asclst@BreakingDescItem[1][]{\item[#1]\leavevmode}
\AsciiListRegisterEnv{Description}{\description}{\enddescription}%
                      {\AsciiListEndOArg{\asclst@BreakingDescItem}}
\AsciiListRegisterEnv{CompactDesc}{\compactdesc}{\endcompactdesc}%
                      {\AsciiListEndOArg{\asclst@BreakingDescItem}}
```

Change History

v2.0a

AsciiDocList: Improved first-line handling. 14

AsciiList: Improved first-line handling. 9

v2.1

General: Documentation of \UPTO . 5

Documentation of \UP and \UPTO 3

AsciiDocList: Add \UPTO to explicitly changle levels upwards 14

Index

N

P

R

S

T

U