The readarray Package

Routines for inputting formatted array data and recalling it on an element-by-element basis.

Steven B. Segletes
steven.b.segletes.civ@mail.mil

2016/11/07
V2.0

Comments About Version 2.0

Version 2.0 of the readarray package has brought major changes, including a *new and improved* syntax. Functionally, the data-reading/parsing code of the package has been revised to use the powerful listofitems package. This has two primary advantages: 1) the data that is read is no longer expanded prior to the read, so that macros can be read and stored in the data arrays using their unexpanded tokens; and 2) list separators other than a space may now be employed to parse the data into array cells.

While a newer preferred syntax has been introduced for reading and recalling arrays, the deprecated syntax is still supported. The user will also note other small changes, such as the fact that errors arising from array-boundary violations now appear in the log file rather than the document itself.

1 Description and Commands

The readarray package allows for the creation of data arrays (numeric, string, or even formatted) using either file contents or `\def` format for input, such that the elements of multiple arrays can be set and later recalled in an orderly fashion, on a cell-by-cell basis. Routines have been developed to support the storage and recall of both 2-D and 3-D arrays, as well as 1-D file-record arrays.[1]

[1] Note: for 1-D arrays that are to be simply parsed on the basis of a specified separator, the listofitems package is already prepared to do this, without the help of this package.

The commands included in this package help the user to input data, define it in terms of array elements, and recall those elements at will. Those commands are:

To place file data into a data macro:

`\readdef{`*filename*`}\`*data-macro*

To place file data into a 1-D file-record array:

`\readrecordarray{`*filename*`}\`*array-identifier*

To parse a data macro and place the results into a 2-D or 3-D array:

`\readarray\`*data-macro*`\`*array-identifier*`[-,`*columns*`]` (2-D)

`\readarray\`*data-macro*`\`*array-identifier*`[-,`*rows*`,`*columns*`]` (3-D)

Same as above, with leading/trailing spaces removed from array cells:

`\readarray*\`*data-macro*`\`*array-identifier*`[-,`*columns*`]` (2-D)

`\readarray*\`*data-macro*`\`*array-identifier*`[-,`*rows*`,`*columns*`]` (3-D)

Recall data from indexed array cell:

`\`*array-identifier*`[`*row*`,`*column*`]` (2-D)

`\`*array-identifier*`[`*plane*`,`*row*`,`*column*`]` (3-D)

To place the actual tokens of an array cell into a macro:

`\arraytomacro\`*array-identifier*`[-,`*columns*`]\`*macro* (2-D)

`\arraytomacro\`*array-identifier*`[-,`*rows*`,`*columns*`]\`*macro* (3-D)

To change the array-parsing separator character:

`\readarraysepchar{`*parsing-separator-char*`}`

To select the level of bounds checking on array cell recall:

`\nocheckbounds` *OR* `\checkbounds` *OR* `\hypercheckbounds`

In these commands, `\`*data-macro* is a command sequence into which the contents of `filename` are set into a `\def`. The *array-identifier* is a sequence of (catcode 11) letters that identify the array. The starred version of the commands are used if, during the array creation, it is desired to automatically excise the array data of leading and trailing spaces.

Unlike earlier versions of this package, where error messages were output into the typeset document, error messages are now set in the log file. The level of error messaging is defined by the level of bounds checking, with `\hypercheckbounds` providing the most intense level of error checking. When a bounds-checking error is found in an array invocation, in addition to the error message in the log file, a "?" is typeset in the document, unless bound checking is disabled with `\nocheckbounds`.

Several strings of fixed name are defined through the use the `\readdef` command, which are accessible to the user:

`\nrows`

`\ncols`

`\nrecords`

`\ArrayRecord[`*record*`]` (to retrieve record from most recent `\readdef`)

The macros `\nrows` and `\ncols`, which were gleaned from the file structure, may be used in the subsequent `\readarray` invocation to specify the array dimensions. Alternately, those values may be manually overridden by specifying the desired values in the `\readarray` invocation. Individual records of the original file, from the most recent `\readdef`, may be recalled with the `\ArrayRecord` macro.

In addition to the strings of fixed name created during the `\readdef`, there are various strings created during the `\readarray` whose name is a function of the *array-identifier*, such as

`\`*array-identifier*`CELLS`
`\`*array-identifier*`PLANES`
`\`*array-identifier*`ROWS`
`\`*array-identifier*`COLS`

where *array-identifier* is the alphabetic-character string by which you have designated a particular array. Their meaning will be discussed later in this document.

Support routines which are generally not required directly by the user for the specification and recall of data arrays, but which are useful for debugging include the following:

`\arraydump\`*array-identifier*
`\scalardump\`*array-identifier*

These macros print out the complete array, in either a structured or unstructured form, respectively.

2 Data Structure

The first requirement is to lay out a format for the data interface to this package. The readarray package is set up to digest data separated by a user-defined separator character. The default separator is a space character but, as of V2.0, the separator may be specified by way of `\readarraysepchar{`*separator*`}`. The format for the data organization to be digested is as follows, for 2-D arrays:

$$\begin{array}{lllll}
A_{11}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{12}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{13}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & \ldots & A_{1(\mathrm{columns})} \\
A_{21}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{22}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & \ldots & & \\
\vdots & & & & \\
A_{(\mathrm{rows})1}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{(\mathrm{rows})2}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{(\mathrm{rows})3}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & \ldots & A_{(\mathrm{rows})(\mathrm{columns})}
\end{array}$$

For 3-D arrays, the following structure is employed:

$$\begin{array}{lllll}
A_{111}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{112}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{113}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & \dots & A_{11(\mathrm{columns})}\\
A_{121}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{122}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & \dots & & \\
\vdots & & & & \\
A_{1(\mathrm{rows})1}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{1(\mathrm{rows})2}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{1(\mathrm{rows})3}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & \dots & A_{1(\mathrm{rows})(\mathrm{columns})}\\
<\text{blank line}> & & & & \\
A_{211}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{212}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{213} & \dots & A_{21(\mathrm{columns})}\\
A_{221}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{222}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & \dots & & \\
\vdots & & & & \\
A_{2(\mathrm{rows})1}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{2(\mathrm{rows})2}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{2(\mathrm{rows})3}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & \dots & A_{2(\mathrm{rows})(\mathrm{columns})}\\
 & & & & \\
\vdots & & & & \\
 & & & & \\
A_{(\mathrm{planes})11}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{(\mathrm{planes})12}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{(\mathrm{planes})13}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & \dots & A_{(\mathrm{planes})1(\mathrm{columns})}\\
A_{(\mathrm{planes})21}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{(\mathrm{planes})22}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & \dots & & \\
\vdots & & & & \\
A_{(\mathrm{planes})(\mathrm{rows})1}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{(\mathrm{planes})(\mathrm{rows})2}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & A_{(\mathrm{planes})(\mathrm{rows})3}\ \langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle & \dots & A_{(\mathrm{planes})(\mathrm{rows})(\mathrm{columns})}
\end{array}$$

Here, $\langle\substack{\mathrm{s}\\\mathrm{e}\\\mathrm{p}}\rangle$ is the data separator that is used to parse the input. Terms like $A_{(\mathrm{plane})(\mathrm{row})(\mathrm{column})}$ refers to the LaTeX-formatted data to be associated with the particular plane, row, and column of data. Note, that for 3-D arrays, a blank line can be used to signify to the parsing algorithm the size of a data plane (alternately, the number of rows per data plane can be explicitly provided to the `\readarray` command).

3 Getting Data into Array Structures

`\readdef`

One can provide data to be digested by this package in one of two ways: either through an external file, or by way of "cut and paste" into a `\def`. If one chooses the external file approach, the command `\readdef` is the command which can achieve this result. The command takes two arguments. The first is the file in which the data is stored, while the second is the data macro into which the file's data will be placed, for example

```
\readdef{data.txt}{\dataA}
```

In this case, the contents of the file `data.txt` will be placed into the data macro `\dataA`. Two alterations to the format occur during this conversion from file to `\def`: 1) blank lines in the file are ignored; and 2) a data separator replaces the end-of-line. At this point, the data is still not digested into a 2-D or 3-D data "array." However, two things have been accomplished: 1) the file contents are `\def`'ed into the data macro `\dataA`; and 2) they are also placed into a 1-D file record array, `\ArrayRecord`.

There is no *requirement* that the input file be organized with structured rows of data corresponding to individual file records, nor that blank lines exist between planes of data (if the data is 3-D). *However*, there is a reason to do so, nonetheless. In particular, for datafiles that are organized in the preferred fashion, for example:

```
A111 A112 A113 A114
A121 A122 A123 A124
A131 A132 A133 A134

A211 A212 A213 A214
A221 A222 A223 A224
A231 A232 A233 A234
```

`\ncols`
`\nrows`
`\nrecords`
`\ArrayRecord`

a `\readdef` attempts to estimate the number columns, and rows-per-plane of the dataset by analyzing the data structure. These estimates are given by `\ncols` and `\nrows`, in this case to values of `4` and `3`, respectively. Such data could prove useful if the array size is not known in advance. When `\readdef` is invoked, a string `\nrecords` will also be set to the number of file records processed by the `\readdef` command, in this case, to `8`. Finally, the 1-D file-record array, `\ArrayRecord`, is created to allow access to the most recently read file records. For example, `\ArrayRecord[3]` produces: "A131 A132 A133 A134". Note, however, that the array, `\ArrayRecord`, will be overwritten on the subsequent invocation of `\readdef`.

Because `\ArrayRecord` is only a 1-D file-record array, the *actual* array metrics, given by `\ArrayRecordCOLS`, `\ArrayRecordROWS`, `\ArrayRecordPLANES`, and `\ArrayRecordCELLS` are 0, 8, 0, and 8, respectively, which do not align with the estimations provided by `\ncols` and `\nrows`.

In lieu of `\readdef`, a generally less preferred, but viable way to make the data available is to cut and paste into a `\def`. However, because a blank line is not permitted as part of the `\def`, a filler symbol (`%` or `\relax`) must be used in its place, if it is desired to visually separate planes of data, as shown in the `\def` example at the top of the following page. Note that the `%` is also required at the end of the line containing `\def`, in order to guarantee that, in this case, `A111` is the first element of data (and not a space separator). However, unlike `\readdef`, this definition will neither set the value of `\ncols` nor `\nrows`.

```
\def{\dataA}{%
A111 A112 A113 A114
A121 A122 A123 A124
A131 A132 A133 A134
%
A211 A212 A213 A214
A221 A222 A223 A224
A231 A232 A233 A234
}
```

Once the data to be placed into an array is available in a macro, by way of either `\readdef` or `\def`, the command to digest the data into an array is `\readarray` for the case of 2-D or 3-D data. For 1-D file-record arrays, in contrast, the `\readrecordarray` command is used to go directly from a file into the 1-D array, bypassing the intermediate step of a data macro.

3.1 1-D File-Record Arrays

If the desire is merely to parse a string of data based on a common data separator, such as a comma or any other character, there is no need to use the readarray package. The listofitems package, which is employed by readarray, already has those provisions and should be used directly.[2]

On the other hand, if one wishes a 1-D file-record array, in which each array element corresponds to the record from a file, then readarray can be used. The command `\readrecordarray` can be used to stick the individual "file records" from a designated file into a 1-D array.

The `\readrecordarray` command takes two arguments: a file name containing data records, and the name of a 1-D record-array into which to place the file records.

So, for example, with the invocation of `\readrecordarray{data.txt}\oneD`, the data from the file `data.txt` is now saved in the `\oneD` array, and can be retrieved, for example, the 3rd record, with `\oneD[3]`, which returns "A131 A132 A133 A134".

If an array name is reutilized, its prior definitions are cleared, so that "old" data is not inadvertantly retrieved following the reutilization.

[2] For a simple 1-D list punctuated by data separators, one may use the listofitems package directly:

```
\setsepchar{ }
\readlist\oneDlist{\dataA}
\oneDlistlen{} list items, 12th item is ``\oneDlist[12]''.
```

which produces the following output: 25 list items, 12th item is "A134".

3.2 Creating 2-D and 3-D Arrays

`\readarray` The `\readarray` command, used to convert raw parsable data into data arrays, takes three arguments. The first is the data macro into which the unarrayed raw data had previously been stuffed (e.g., by way of `\readdef` or `\def`). The second is array-identifier macro into which the parsed data is to be placed. Finally, the last compound argument, enclosed in square brackets, denotes the rank and range of the array to be created.

There is a starred version of the command, `\readarray*`, which is used to remove leading/trailing spaces from the array elements, when parsed. This option, is only relevant when the data separator is not already a space.

If an array name is reutilized, its prior definitions are cleared, so that “old” data is not inadvertantly retrieved following the reutilization.

3.2.1 2-D Arrays

For a 2-D array, this last argument of `\readarray` will be of the form `[-,<columns>]`. If the data had recently been read by way of `\readdef`, the string `\ncols` may be used to signify the `<columns>` value. The `-` (or any other character before the initial comma) reminds us that the range of row numbers is not specified in advance, but is dictated by the length of the data macro containing the raw file data. For such a 2-D array, only the column range is specified.

Consider, for example, the previously discussed file, `dataA.txt`, which had been digested into the data macro `\dataA`. One can process that as a 2-D array with an invocation of `\readarray\dataA\twoD[-,\ncols]`, since `\ncols` had been set to a value of 4, based on the prior `\readdef`. Thereafter, data may be retieved, for example the 3rd row, 2nd column, with `\twoD[3,2]`, to give “A132”.

The actual array size is given by `\twoDROWS`, `\twoDCOLS`, `\twoDCELLS` as 6, 4, and 24, respectively. The number of rows in the array is fewer than the number of file records, 8, because blank rows in the input file are ignored. One should also note that if the end of the data stream results in a partial final row of data, the partial row will be discarded.

3.2.2 3-D Arrays

For the 3-D case, the only difference in the invocation of `\readarray` is in the 3rd argument, in which the rank and range of the array is specified. This last argument will be of the form `[-,<rows>,<columns>]`. As before, the `-` denotes the fact that the range of the planes of data is unknown before the fact, and governed by the length of data in the dataset. Only the range of rows and columns are specifiable here. If `\readdef` had been used on a properly formed input file, both `\nrows` and `\ncols` may be used to supply the range arguments of the 3-D array.

For example, using the same `\dataA` dataset, but reading it as a 3-D array can be accomplished with `\readarray\dataA\threeD[-,\nrows,\ncols]`. This results in an array with 2 planes, 3 rows, and 4 columns (24 data cells total). Data from the 2nd plane, 1st row, 2nd column can be obtained via `\threeD[2,1,2]` as “A212”.

If, perchance, a row or plane is only partially defined by `\readarray`, the partial data is discarded from the array.

3.2.3 Array Parsing Separator

While it may be easily envisioned that the array data is numerical, this need not be the case. The array data may be text, and even formatted text.

`\readarraysepchar`

Furthermore, one may introduce space characters into the data of individual cells simply by resetting the readarray parsing separator to something other than the default space, “ ”. This can be done, for example employing a comma as the separator, by way of `\readarraysepchar{,}`.

Note also, using the facilities of the underlying listofitems package, that compound separators are possible. For example, *either* a comma *or* a period may be used for the data parsing, by specifying a **logical-OR** (`||`) separated list: `\readarraysepchar{,||.}`. Similarly, a multicharacter separator is possible, so that setting `\readarraysepchar{!!}` will cause `\readarray` to look for instances of “!!” to divide the data into separate array elements.

Consider the following comma-separated input in, let us say, the file conjugation.txt.

```
\textit{am} ,  \textit{are}, have \textit{been}, have \textit{been}
\textit{are}, \textit{are} , have \textit{been}, have \textit{been}
\textit{is} , \textit{are} , has \textit{been} , have \textit{been}

\textit{was} , \textit{were}, had \textit{been}, had \textit{been}
\textit{were}, \textit{were}, had \textit{been}, had \textit{been}
\textit{was} , \textit{were}, had \textit{been}, had \textit{been}

will \textit{be}, will \textit{be}, will have \textit{been}, will have \textit{been}
will \textit{be}, will \textit{be}, will have \textit{been}, will have \textit{been}
will \textit{be}, will \textit{be}, will have \textit{been}, will have \textit{been}
```

The sequence of commands

```
\readarraysepchar{,}
\readdef{conjugation.txt}\dataC
\readarray*\dataC\tobeConjugation[-,\nrows,\ncols]
```

will employ a comma separator to parse the file. It will then create a 3-D array using data from the file, placed into the array `\tobeConjugation`. Leading/trailing spaces will be removed from the data, with the use of the star form of the `\readarray` command. Data can then be directly accessed, so that, for example `\tobeConjugation[1,3,3]` will yield the entry from the 1st plane, 3rd row, 3rd column as “has *been*”.

The 3-D array metrics are `\tobeConjugationPLANES`, `\tobeConjugationROWS`, `\tobeConjugationCOLS`, and `\tobeConjugationCELLS`, which are here given as 3, 3, 4, and 36. respectively.

4 Recalling Data from Array Structures

\array-identifier`CELLS`
\array-identifier`PLANES`
\array-identifier`ROWS`
\array-identifier`COLS`

While one must specify the number of columns and/or rows associated with the `\readarray` invocation, those numbers may not yet be known to the user, if the values employed came from the `\readdef` estimations of `\ncols` and `\nrows`. Therefore, the `\readrray` command variants also define the following strings: \array-identifier`CELLS`, \array-identifier`PLANES`, \array-identifier`ROWS`, and \array-identifier`COLS`, where `\array-identifier` is the array name supplied to the `\readarray` command. Note, for 3-D arrays, that

\array-identifier`CELLS` =
  \array-identifier`PLANES` × \array-identifier`ROWS` × \array-identifier`COLS`

For the `\tobeConjugation` example of the prior section, 36=3×3×4. Likewise, for 2-D arrays

\array-identifier`CELLS` = \array-identifier`ROWS` × \array-identifier`COLS`

\array-identifier`[...]`

To retrieve the data from the array, one merely supplies the array name in the form of *array-identifier*, along with the array-cell nomenclature in the form of `[`*plane*`,`*row*`,`*column*`]` for 3-D arrays, `[`*row*`,`*column*`]` for 2-D arrays, and `[`*row*`]` for 1-D arrays.

Thus, in the case of the earlier example involving conjugation of the verb *to be*, the second-person future-perfect tense of the verb is given by

```
\tobeConjugation[3,2,4]
```

which yields “will have *been*”.

5 Bounds Checking

While the user is developing his or her application involving the readarray package, there may accidentally arise the unintended circumstance where an array element is requested which falls outside the array bounds. In general, when a non-existent array element is requested in the absence of bounds checking, the call will expand to `\relax`.

`\nocheckbounds`
`\checkbounds`
`\hypercheckbounds`

The package provides three declarations to set the manner in which array bounds are be monitored. The setting `\nocheckbounds` is used when bounds are not to be checked. This is the default behavior for readarray. For some bounds checking, `\checkbounds` may be set. With this setting, bounds violations are noted, but no guidance is provided as to the allowable index range for the array. However, with `\hypercheckbounds` set, full bounds checking is possible. With this setting, not only are violations noted, but a description of the actual array range is provided.

As of V2.0, bounds violations are noted in the log file, rather than the document itself. However, if an array bound is violated when bounds checking is turned on, a “?” shows up in the document itself.

6 Accessing Array Cells if Full Expansion is Required (e.g., placed in an `\edef`)

If full expansion is required of array cell contents (and assuming the cell content is expandable), it is advisable to set `\nocheckbounds`, so that the error checking code is not included in the expansion. Results may be also expanded even with `\checkbounds` set, though the error-checking code is part of the expansion. However, with `\hypercheckbounds` set, full expansion of array cells is no longer possible.

7 Accessing Array Cells if No Expansion is Required

With the normal use of \array-identifier syntax for accessing array cells, several levels of expansion are required to directly recover the original tokens of the cell,

`\arraytomacro`

and then only when bounds checking is disabled. When the actual unexpanded tokens of cell are required, the use of the `\arraytomacro` command provides the means to accomplish this. The command takes the array name and index as the initial arguments followed by a generic macro name into which to place the unexpanded tokens of the indexed array cell.

So, for example `\arraytomacro\tobeConjugation[2,2,3]\thiscell` will place the cell's original tokens in the macro `\thiscell`. Upon detokenization, `\thiscell` contains "`had \textit {been}`".

8 Support Routines

`\arraydump`

`\scalardump`

The package provides two commands that can help one understand how a data set has been parsed into an array. Both of these commands dump the specified array to the document. In the case of `\arraydump`, the array is formatted in the structure of the array, broken up by columns, rows, and planes. In the case of `\scalardump`, however, the elements of the array are dumped sequentially, without reference to the array's heirarchy.

For the case of 1-D record array `\oneD` employed in prior sections, for example, the invocations of

```
\arraydump\oneD
\scalardump\oneD
```

results in

```
1-D: ____________________________________________________
▮A111 A112 A113 A114▮                          <Record 1>
▮A121 A122 A123 A124▮                          <Record 2>
▮A131 A132 A133 A134▮                          <Record 3>
▮▮                                             <Record 4>
▮A211 A212 A213 A214▮                          <Record 5>
▮A221 A222 A223 A224▮                          <Record 6>
▮A231 A232 A233 A234▮                          <Record 7>
▮▮                                             <Record 8>
_________________________________________________________
```

```
8 ELEMENTS: _____________________________________________
▮A111 A112 A113 A114▮A121 A122 A123 A124▮A131 A132 A133 A134▮▮A211
A212 A213 A214▮A221 A222 A223 A224▮A231 A232 A233 A234▮▮
_________________________________________________________
```

The \twoD equivalent, resulting from parsing the same data file as a 2-D, rather than a 1-D record array, is

```
\arraydump\twoD
\scalardump\twoD
```

```
2-D:
▮A111▮A112▮A113▮A114▮                                  <Row 1>
▮A121▮A122▮A123▮A124▮                                  <Row 2>
▮A131▮A132▮A133▮A134▮                                  <Row 3>
▮A211▮A212▮A213▮A214▮                                  <Row 4>
▮A221▮A222▮A223▮A224▮                                  <Row 5>
▮A231▮A232▮A233▮A234▮                                  <Row 6>

24 ELEMENTS:
▮A111▮A112▮A113▮A114▮A121▮A122▮A123▮A124▮A131▮A132▮A133▮A134▮A211▮
A212▮A213▮A214▮A221▮A222▮A223▮A224▮A231▮A232▮A233▮A234▮
```

For the case of the 3-D array (earlier read as \threeD), the \arraydump would appear as

```
\arraydump\threeD
\scalardump\threeD
```

```
3-D: Plane 1
▮A111▮A112▮A113▮A114▮                                  <Row 1>
▮A121▮A122▮A123▮A124▮                                  <Row 2>
▮A131▮A132▮A133▮A134▮                                  <Row 3>
Plane 2
▮A211▮A212▮A213▮A214▮                                  <Row 1>
▮A221▮A222▮A223▮A224▮                                  <Row 2>
▮A231▮A232▮A233▮A234▮                                  <Row 3>

24 ELEMENTS:
▮A111▮A112▮A113▮A114▮A121▮A122▮A123▮A124▮A131▮A132▮A133▮A134▮A211▮
A212▮A213▮A214▮A221▮A222▮A223▮A224▮A231▮A232▮A233▮A234▮
```

Note that the \scalardump of \threeD is indistinguishable from that of \twoD, since both arrays are comprised of the same data cells, though arrayed into different plane/row/column structures.

For comparison, the `\arraydump` of `\tobeConjugation` is

```
3-D: PLANE 1
▮am▮are▮have been▮have been▮                                  <Row 1>
▮are▮are▮have been▮have been▮                                 <Row 2>
▮is▮are▮has been▮have been▮                                   <Row 3>
PLANE 2
▮was▮were▮had been▮had been▮                                  <Row 1>
▮were▮were▮had been▮had been▮                                 <Row 2>
▮was▮were▮had been▮had been▮                                  <Row 3>
PLANE 3
▮will be▮will be▮will have been▮will have been▮               <Row 1>
▮will be▮will be▮will have been▮will have been▮               <Row 2>
▮will be▮will be▮will have been▮will have been▮               <Row 3>
```

9 Deprecated, Vestigial, and Defunct Features

Deprecated

The following commands are supplied, but are no longer the preferred embodiment of package syntax.

```
\copyrecords{array-identifier}
\readArrayij{\data-macro}{array-identifier}{columns}
\readArrayij*{\data-macro}{array-identifier}{columns}
\readArrayijk{\data-macro}{array-identifier}{rows}{columns}
\readArrayijk*{\data-macro}{array-identifier}{rows}{columns}
\showrecord[error]{record number}
\Arrayij[error]{array-identifier}{row}{column}
\Arrayijk[error]{array-identifier}{plane}{row}{column}
\arrayij{array-identifier}{row}{column}
\arrayijk{array-identifier}{plane}{row}{column}
```

Vestigial

The following support macros are provided but no longer recommended. Their capability is more fully served by way of the listofitems package.

```
\getargsC{\macro or string}
\argindex
\narg
\showargs
```

Note that whereas `\getargs` could previously (pre-V2.0 readarray) employ only a space as the parsing separator, `\getargs` now respects the currently set value of separator, as (re)defined by `\readarraysepchar`.

Defunct

The following macros are no longer supported.

```
\converttilde
\recordindex
```

Since the package now supports arbitrary parsing separators, there is no need for the function of `\converttilde`. However, were one desiring to parse, while treating hard spaces as spaces, this can be simply achieved under V2.0 readarray by setting the parsing character as either a space or a hard space, using `readarraysepchar{ ||~}`. Likewise, the indirect addressing (using a roman-numeral *index*) provided by the internal command `\record`*index* is fully superceded by the ability to directly address any record of readarray's 1-D record arrays.

10 Acknowledgements

I am profoundly thankful to Christian Tellechea for using my simplistic (read "deficient") getargs package to inspire his effort in creating the powerful listofitems package. It is precisely the tool I have sought for a long time, and I have adapted its use into the workings of this package.

I would like to thank Dr. David Carlisle for his assistance in helping the author rewrite the `\getargs` command, originally found in the stringstrings package. To distinguish the two versions, and in deference to him, it is herein named `\getargsC`. However, as of V2.0, its presence is vestigial, having instead been superceded with the listofitems package macros.

I am likewise grateful to Ken Kubota, who suggested moving the `\newread` outside of `\readdef`, so as not to prematurely exhaust the 16 available file streams.

11 Code Listing

```
\def\readarrayPackageVersion{2.0}
\def\readarrayPackageDate{2016/11/07}
\ProvidesPackage{readarray}
[\readarrayPackageDate\ \readarrayPackageVersion\ %
Routines for inputting 2D and 3D array data and recalling it on an
element-by-element basis.]
%
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3c or later is part of all distributions of LaTeX
% version 2005/12/01 or later.
%
% This work has the LPPL maintenance status ‘maintained’.
%
% The Current Maintainer of this work is Steven B. Segletes.
%
% Revisions:
% v1.01 -Documentation revision
% v1.1  -Added \csname record\roman{@row}\endcsname to \readdef
% v1.2  -Corrected the [truncated] LPPL license info
%       -Added \arrayij and \arrayijk, which can be put into \edef
%       -Used \romannumeral in preference to \roman{}, when possible,
%        to avoid unnecessary use of counters.
% v1.3  -Moved \newread outside of \readdef, so as not to exhaust the
%        16 allotted file streams (Thanks to Ken Kubota for the tip).
% v2.0  -Converted parsing to listofitems package.  This allows for
%        ANY parsing character or combination of characters (via logical OR).
%       -Replaced all \protected@edef’s with appropriately expanded \def’s.
%       -Use listofitems package in preference to \getargsC.
%       -Deprecated \Arrayijk, \arrayijk, \Arrayij, & \arrayij.  Direct
%        access now preferred, e.g., \xyz[2,3,1].
%       -Deprecated most other commands in favor of a more natural syntax.
\RequirePackage{ifthen}
\RequirePackage{listofitems}[2016-10-22]
%
\newcounter{@index}
\newcounter{@plane}
\newcounter{@row}
\newcounter{@col}
\newcounter{use@args}
\newcounter{@record}
\newcounter{arg@index}
\newcounter{break@count}
\newcounter{index@count}
\newcounter{loop@count}
\newtoks\Arg@toks
\newtoks\@arrayident@toks
\newread\rdar@file
%
\newcommand\readdef[2]{\@readdef{#1}{#2}{ArrayRecord}}
%
```

```
\newcommand\readrecordarray[2]{%
  \edef\@arrayident{\rdar@macroname#2}%
  \def\ra@TermA{\@readdef{#1}}%
  \def\ra@TermB{\expandafter\ra@TermA\csname\@arrayident def\endcsname}%
  \expandafter\ra@TermB\expandafter{\@arrayident}%
}
%
\newcommand\readarray{\@ifstar{\read@array@newsyntax[*]}{\read@array@newsyntax[]}}
%
\def\arraytomacro#1[#2]#3{%
  \edef\@arrayident{\rdar@macroname#1[#2]}%
  \@arrayident@toks=\expandafter\expandafter\expandafter{\csname\@arrayident\endcsname}%
  \expandafter\def\expandafter#3\expandafter{\the\@arrayident@toks}%
}
%
\newcommand\readarraysepchar[1]{\def\read@array@sepchar{#1}}
%
\def\nocheckbounds{\def\rootmacro@aux##1##2##3{##1%
 }\typeout{readarray: bounds checking OFF}%
}
%
\def\checkbounds{\def\rootmacro@aux##1##2##3{##1%
  \expandafter\ifx##1\relax\readarrayboundfailmsg%
  \setbox0=\hbox{\typeout{readarray Warning: ##1 out of bounds.}}%
  \fi%
 }\typeout{readarray: bounds checking ON}%
}
%
\def\hypercheckbounds{\def\rootmacro@aux##1##2##3{##1%
  \expandafter\ifx##1\relax\readarrayboundfailmsg%
    \typeout{readarray Warning: ##1 out of bounds:}%
  \fi%
  \setcounter{index@count}{0}%
  \parse@index##3,\relax%
  \setcounter{loop@count}{0}%
  \whiledo{\value{loop@count}<\value{index@count}}{%
    \stepcounter{loop@count}%
    \ifnum\csname parsed@index[\theloop@count]\endcsname<0%
      \relax\typeout{\nonposmessage{##2}{##3}}\fi%
  }%
  \ifnum \value{index@count}=1\relax%
    \ifnum\csname parsed@index[1]\endcsname>%
      \csname##2CELLS\endcsname\relax\typeout{\recordmessage{##2}{##3}}\fi%
  \fi
  \ifnum \value{index@count}=2\relax%
    \ifnum\csname parsed@index[1]\endcsname>%
      \csname##2ROWS\endcsname\relax\typeout{\rowmessage{##2}{%
        \csname parsed@index[1]\endcsname}}\fi%
    \ifnum\csname parsed@index[2]\endcsname>%
      \csname##2COLS\endcsname\relax\typeout{\colmessage{##2}{%
        \csname parsed@index[2]\endcsname}}\fi%
  \fi
  \ifnum \value{index@count}=3\relax%
    \ifnum\csname parsed@index[1]\endcsname>%
      \csname##2PLANES\endcsname\relax\typeout{\planemessage{##2}{%
        \csname parsed@index[1]\endcsname}}\fi%
    \ifnum\csname parsed@index[2]\endcsname>%
```

```
      \csname##2ROWS\endcsname\relax\typeout{\rowmessage{##2}{%
        \csname parsed@index[2]\endcsname}}\fi%
    \ifnum\csname parsed@index[2]\endcsname>%
      \csname##2COLS\endcsname\relax\typeout{\colmessage{##2}{%
        \csname parsed@index[3]\endcsname}}\fi%
  \fi%
 }\typeout{readarray: bounds hyperchecking ON}%
}
%
\def\rdar@macroname{\expandafter\@gobble\string}
%
\def\getArg@toks[#1]{\Arg@toks\expandafter\expandafter\expandafter{\Arg@list[#1]}}
%
\newcommand\@readdef[3]{%
  \clear@array{#3}%
  \edef\former@recordcount{\csname #3CELLS\endcsname}%
  \def\first@row{T}%
  \def\first@plane{T}%
  \catcode\endlinechar=9 %
  \def#2{}%
  \setcounter{@record}{0}%
  \openin\rdar@file=#1%
  \loop\unless\ifeof\rdar@file%
    \read\rdar@file to\rdar@fileline % Reads a line of the file into \rdar@fileline%
    \addtocounter{@record}{1}%
    \expandafter\g@addto@macro\expandafter#2\expandafter{\rdar@fileline}%
    \ifthenelse{\equal{\rdar@fileline}{}}{}{\expandafter\g@addto@macro%
      \expandafter#2\expandafter{\read@array@sepchar}}%
    \if T\first@row\read@array{#2}\setcounter{@col}{\numexpr(\Arg@listlen-1)}%
      \edef\ncols{\arabic{@col}}\def\first@row{F}\setcounter{@row}{1}%
    \else%
      \if T\first@plane%
        \ifthenelse{\equal{\rdar@fileline}{}}{%
          \edef\nrows{\arabic{@row}}\def\first@plane{F}%
        }{%
          \addtocounter{@row}{1}%
        }%
      \fi%
    \fi%
    \def\record@name{\csname #3[\the@record]\endcsname}%
    \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter%
      \expandafter\def\expandafter\record@name\expandafter{\rdar@fileline}%
  \repeat%
  \edef\nrecords{\arabic{@record}}%
  \expandafter\edef\csname #3PLANES\endcsname{0}%
  \expandafter\edef\csname #3ROWS\endcsname{\nrecords}%
  \expandafter\edef\csname #3COLS\endcsname{0}%
  \expandafter\edef\csname #3CELLS\endcsname{\nrecords}%
  \closein\rdar@file%
  \catcode\endlinechar=5 %
  \define@rootmacro{#3}%
}
%
\def\read@array@newsyntax[#1]#2#3[#4,#5]{%
  \edef\@arrayident{\rdar@macroname#3}%
  \setcounter{index@count}{0}%
  \parse@index#5,\relax%
  \ifnum\value{index@count}=1\relax%
     \def\ra@TermA{\readArrayij#1{#2}}%
     \edef\ra@TermB{{\@arrayident}{\csname parsed@index[1]\endcsname}}%
     \expandafter\ra@TermA\ra@TermB%
  \else
  \ifnum\value{index@count}=2\relax%
     \def\ra@TermA{\readArrayijk#1{#2}}%
     \edef\ra@TermB{{\@arrayident}{\csname parsed@index[1]\endcsname}%
      {\csname parsed@index[2]\endcsname}}%
     \expandafter\ra@TermA\ra@TermB%
  \fi\fi
}
%
\newcommand\read@Arrayijk[5][]{%
  \clear@array{#3}%
  \read@array[#1]{#2}%
  \setcounter{@plane}{\numexpr(\Arg@listlen/#5/#4)}%
  \setcounter{use@args}{\numexpr\arabic{@plane}*#4*#5}%
  \ifthenelse{\arabic{use@args} > \Arg@listlen}{%
    \addtocounter{@plane}{-1}%
    \setcounter{use@args}{\numexpr\arabic{@plane}*#4*#5}%
  }{}%
  \expandafter\edef\csname#3PLANES\endcsname{\arabic{@plane}}%
  \expandafter\edef\csname#3ROWS\endcsname{#4}%
  \expandafter\edef\csname#3COLS\endcsname{#5}%
  \expandafter\edef\csname#3CELLS\endcsname{\arabic{use@args}}%
  \setcounter{@index}{0}%
  \setcounter{@plane}{1}%
  \setcounter{@row}{1}%
  \setcounter{@col}{0}%
  \whiledo{\value{@index} < \value{use@args}}{%
    \addtocounter{@index}{1}%
    \addtocounter{@col}{1}%
    \ifthenelse{\value{@col} > #5}%
      {\addtocounter{@row}{1}%
       \addtocounter{@col}{-#5}}%
      {}%
    \ifthenelse{\value{@row} > #4}%
      {\addtocounter{@plane}{1}%
       \addtocounter{@row}{-#4}}%
      {}%
    \def\arg@name{\csname#3[\the@plane,\the@row,\the@col]\endcsname}%
    \getArg@toks[\the@index]%
    \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter%
      \expandafter\def\expandafter\arg@name\expandafter{\the\Arg@toks}%
  }%
  \define@rootmacro{#3}%
}
%
\newcommand\read@Arrayij[4][]{%
  \clear@array{#3}%
  \read@array[#1]{#2}%
  \setcounter{@row}{\numexpr(\Arg@listlen/#4)}%
  \setcounter{use@args}{\numexpr\arabic{@row}*#4}%
  \ifthenelse{\arabic{use@args} > \Arg@listlen}{%
    \addtocounter{@row}{-1}%
    \setcounter{use@args}{\numexpr\arabic{@row}*#4}%
  }{}%
  \expandafter\edef\csname#3PLANES\endcsname{0}%
  \expandafter\edef\csname#3ROWS\endcsname{\arabic{@row}}%
  \expandafter\edef\csname#3COLS\endcsname{#4}%
  \expandafter\edef\csname#3CELLS\endcsname{\arabic{use@args}}%
  \setcounter{@index}{0}%
  \setcounter{@row}{1}%
  \setcounter{@col}{0}%
  \whiledo{\value{@index} < \value{use@args}}{%
    \addtocounter{@index}{1}%
    \addtocounter{@col}{1}%
    \ifthenelse{\value{@col} > #4}%
      {\addtocounter{@row}{1}%
       \addtocounter{@col}{-#4}}%
      {}%
    \def\arg@name{\csname#3[\the@row,\the@col]\endcsname}%
    \getArg@toks[\the@index]%
    \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter%
      \expandafter\def\expandafter\arg@name\expandafter{\the\Arg@toks}%
  }%
  \define@rootmacro{#3}%
}
%
\newcommand\clear@array[1]{%
  \expandafter\ifx\csname #1\endcsname\relax\else%
    \setcounter{@row}{0}%
    \whiledo{\value{@row}<\csname #1ROWS\endcsname}{%
      \stepcounter{@row}%
      \ifnum\csname #1COLS\endcsname=0\relax%
        \expandafter\let\csname #1[\the@row]\endcsname\relax%
      \else
        \setcounter{@col}{0}%
        \whiledo{\value{@col}<\csname #1COLS\endcsname}{%
          \stepcounter{@col}%
          \ifnum\csname #1PLANES\endcsname=0\relax%
            \expandafter\let\csname #1[\the@row,\the@col]\endcsname\relax%
          \else
            \setcounter{@plane}{0}%
            \whiledo{\value{@plane}<\csname #1PLANES\endcsname}{%
              \stepcounter{@plane}%
              \expandafter%
                \let\csname #1[\the@plane,\the@row,\the@col]\endcsname\relax%
            }%
          \fi%
        }%
      \fi%
    }%
  \fi%
}
%
\newcommand\read@array[2][]{%
  \bgroup%
  \expandafter\setsepchar\expandafter{\read@array@sepchar}%
  \greadlist#1\Arg@list{#2}%
  \egroup%
  \edef\Arg@listCELLS{\Arg@listlen}%
}
```

```
%
\def\define@rootmacro#1{%
  \expandafter\def\csname#1\endcsname[##1]{%
    \expandafter\rootmacro@aux\csname #1[##1]\endcsname{#1}{##1}}%
}
%
\def\parse@index#1,#2\relax{%
  \stepcounter{index@count}%
  \expandafter\gdef\csname parsed@index[\theindex@count]\endcsname{#1}%
  \ifx\relax#2\relax\else\parse@index#2\relax\fi%
}
% INITIALIZATION
\readarraysepchar{ }
\nocheckbounds
%
\def\nonposmessage#1#2{Negative index [#2] impermissable for #1.}
\def\recordmessage#1#2{RECORD=#2 exceeds bounds(=\csname#1CELLS\endcsname) for #1.}
\def\planemessage#1#2{PLANE=#2 exceeds bounds(=\csname#1PLANES\endcsname) for #1.}
\def\rowmessage#1#2{ROW=#2 exceeds bounds(=\csname#1ROWS\endcsname) for #1.}
\def\colmessage#1#2{COL=#2 exceeds bounds(=\csname#1COLS\endcsname) for #1.}
%
\def\the@showargs@rule{\kern.2pt\rule{.8ex}{1.6ex}\hspace{.2pt}}%
\def\readarrayboundfailmsg{?}% DISPLAYED WHEN ARRAY CALL OUT OF BOUNDS
%
% SUPPORT/DEBUG ROUTINES
%
% \arraydump INITIALIZATIONS
\def\row@spacer{\\}
\def\row@msg{\the@showargs@rule\hfill{\scriptsize\scshape$<$\row@sign~\arabic{@row}$>$}}
\def\header@msg{{\bfseries\ra@rank:}~}
\def\last@row{\\}
\def\plane@msg{\plane@sign\hrulefill\mbox{}\\}
\def\close@out{}
%
\newcommand\arraydump[1]{%
  \edef\@arrayident{\rdar@macroname#1}%
  \expandafter\ifx\csname\@arrayident\endcsname\relax\else%
    \edef\ra@TmpA{\csname\@arrayident PLANES\endcsname}%
    \edef\ra@TmpB{\csname\@arrayident COLS\endcsname}%
    \def\ra@rank{3-D}%
    \ifnum\ra@TmpA=0\relax\def\ra@TmpA{1}\def\plane@sign{\mbox{}}\def\ra@rank{2-D}%
      \else\def\plane@sign{{\scriptsize\scshape Plane \arabic{@plane}}}\fi%
    \ifnum\ra@TmpB=0\relax\def\ra@TmpB{1}\def\row@sign{Record}\def\ra@rank{1-D}%
      \else\def\row@sign{Row}\fi%
    \par\noindent\header@msg%
    \setcounter{@plane}{0}%
    \whiledo{\value{@plane}<\ra@TmpA}{%
      \stepcounter{@plane}%
      \plane@msg%
      \setcounter{@row}{0}%
      \whiledo{\value{@row}<\csname\@arrayident ROWS\endcsname}{%
        \ifnum\value{@row}=0\relax\else\row@spacer\fi%
        \stepcounter{@row}%
        \setcounter{@col}{0}%
        \whiledo{\value{@col}<\ra@TmpB}{%
          \the@showargs@rule%
          \stepcounter{@col}%
          \ifnum\csname\@arrayident COLS\endcsname=0\relax%
            #1[\the@row]%
          \else%
            \ifnum\csname\@arrayident PLANES\endcsname=0\relax%
              #1[\the@row,\the@col]%
            \else%
              #1[\the@plane,\the@row,\the@col]%
            \fi%
          \fi%
        }\row@msg%
      }\last@row%
    }\close@out\mbox{}\hrulefill\mbox{}\par%
  \fi%
}
%
\newcommand\scalardump[1]{\bgroup%
  \def\row@spacer{}%
  \def\row@msg{}%
  \def\header@msg{{\bfseries\csname\@arrayident CELLS\endcsname\ ELEMENTS:}%
    ~\hrulefill\mbox{}\\}%
  \def\last@row{}%
  \def\plane@msg{}%
  \def\close@out{\the@showargs@rule\\}%
  \arraydump#1\egroup%
}
%
% DEPRECATED COMMANDS (NOT PREFERRED EMBODIMENT OF PACKAGE SYNTAX)
%
\newcommand\readArrayijk{\@ifstar{\read@Arrayijk[*]}{\read@Arrayijk}}
\newcommand\readArrayij{\@ifstar{\read@Arrayij[*]}{\read@Arrayij}}
\newcommand\arrayijk[4]{\csname#1[#2,#3,#4]\endcsname}
\newcommand\arrayij[3]{\csname#1[#2,#3]\endcsname}
\newcommand\Arrayijk[5][\relax]{%
  \bgroup%
  \ifx\relax#1\else\def\readarrayboundfailmsg{#1}\fi\csname#2\endcsname[#3,#4,#5]%
  \egroup%
}
\newcommand\Arrayij[4][\relax]{%
  \bgroup%
  \ifx\relax#1\else\def\readarrayboundfailmsg{#1}\fi\csname#2\endcsname[#3,#4]%
  \egroup%
}
\newcommand\copyrecords[1]{%
  \clear@array{#1}%
  \edef\former@recordcount{\csname #1CELLS\endcsname}%
  \setcounter{@record}{0}%
  \whiledo{\value{@record} < \nrecords}{%
    \addtocounter{@record}{1}%
    \def\arg@name{\csname#1[\the@record]\endcsname}%
    \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter%
      \expandafter\def\expandafter\arg@name\expandafter{%
        \csname ArrayRecord[\the@record]\endcsname}%
  }%
  \expandafter\edef\csname#1PLANES\endcsname{0}%
  \expandafter\edef\csname#1ROWS\endcsname{\nrecords}%
  \expandafter\edef\csname#1COLS\endcsname{0}%
  \expandafter\edef\csname#1CELLS\endcsname{\nrecords}%
  \define@rootmacro{#1}%
}
\newcommand\showargs[1][0]{\bgroup%
  \def\Arg@listPLANES{0}%
  \def\Arg@listCOLS{0}%
  \let\Arg@listROWS\Arg@listCELLS%
  \scalardump\Arg@list\egroup%
}
\newcommand\showrecord[2][\relax]{%
  \bgroup\ifx\relax#1\else\def\readarrayboundfailmsg{#1}\fi\ArrayRecord[#2]\egroup%
}
% The support routine \getargs{} is provided for backward compatibility.
% It is preferable to directly use facilities of the
% listofitems package to accomplish these tasks.
\def\getargsC#1{%
  \bgroup%
  \expandafter\setsepchar\expandafter{\read@array@sepchar}%
  \greadlist\Arg@list{#1}%
  \egroup%
  \edef\narg{\Arg@listlen}%
  \let\Arg@listCELLS\narg%
  \setcounter{@index}{0}%
  \whiledo{\value{@index}<\narg}{%
    \stepcounter{@index}%
    \expandafter\edef\csname arg\romannumeral\value{@index}\endcsname{%
      \Arg@list[\value{@index}]}%
  }%
}
\endinput
```