The readarray Package

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

Steven B. Segletes
SSegletes@verizon.net

2021/09/17
V3.1

Comments About Version 3.0

Version 3.0 of readarray introduces several new features. Arrays can be initialized with the `\initarray` command, rather than being read from a file. With `\setvalue`, the tokens of individual array cells can be revised on demand. The contents of one array can be merged into another by way of the `\mergearray` command. The `\typesetarray` command allows for versatile ways to format your array output, including by way of `tabular` environments. The way end-of-line characters can be handled on a data-read is made more versatile with this package version. There are many new things here—check them out.

Comments About Version 2.0

Version 2.0 of the readarray package brought major changes, including a *new and improved* syntax. Functionally, the data-reading/parsing code of the package was 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. 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* (1-D)
To parse a data macro and place the results into a new 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)
To create a new array, each cell initialized to a default value:
  `\initarray\`*array-identifier*`[`*rows*`,`*columns*`]` (2-D)
  `\initarray\`*array-identifier*`[`*planes*`,`*rows*`,`*columns*`]` (3-D)
To merge each cell of one array into another:
  `\mergearray\`*from-array-id*`\`*to-array-id*`[`*row*`,`*column*`]` (2-D)
  `\mergearray\`*from-array-id*`\`*to-array-id*`[`*plane*`,`*row*`,`*column*`]` (3-D)
To typeset an array:
  `\typesetarray\`*array-identifier* (2-D/3-D)
Recall (typeset) 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*`[`*row*`,`*column*`]\`*macro* (2-D)
  `\arraytomacro\`*array-identifier*`[`*plane*`,`*row*`,`*column*`]\`*macro* (3-D)
To manually (re)set the value of an existing array cell:
  `\setvalue\`*array-identifier*`[`*row*`,`*column*`]{`*value*`}` (2-D)
  `\setvalue\`*array-identifier*`[`*plane*`,`*row*`,`*column*`]{`*value*`}` (3-D)
To change the array-parsing separator for `\readdef` and/or `\readarray`:
  `\readarraysepchar{`*parsing-separator-char*`}`
To change the default (=9) end-of-line catcode for `\readdef`:
  `\readarrayendlinechar=`*end-of-line-catcode*
To redefine default cell initialization for `\initarray` :
  `\renewcommand\readarrayinitvalue{`*value*`}`
To redefine default cell format in `\typesetarray`:
  `\renewcommand\typesetcell[1]{`*format-for-#1*`}`
To redefine default plane, row, column separator formats for `\typesetarray`:
  `\renewcommand\typesetplanesepchar{`*format*`}`
  `\renewcommand\typesetrowsepchar{`*format*`}`
  `\renewcommand\typesetcolsepchar{`*format*`}`
To select the level of bounds checking on array cell recall:
  `\nocheckbounds` *OR* `\checkbounds` *OR* `\hypercheckbounds`
To ignore blank (or fully commented) records during the course of a record-array

read (this can be undone with a false setting):
  `\ignoreblankreadarrayrecordstrue`

In these commands, *data-macro* is a control sequence into which the contents of `filename` are set into a `\def`. The *array-identifier* is the array into which cell data is inserted. 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 name you have designated a particular array. Their meaning will be discussed later in this document.

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 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:

$$
\begin{array}{llll}
A_{11}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{12}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{13}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & \ldots\ A_{1(\text{columns})} \\
A_{21}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{22}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & \ldots & \\
\vdots & & & \\
A_{(\text{rows})1}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{(\text{rows})2}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{(\text{rows})3}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & \ldots\ A_{(\text{rows})(\text{columns})}
\end{array}
$$

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

$$
\begin{array}{llll}
A_{111}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{112}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{113}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & \ldots\ A_{11(\text{columns})} \\
A_{121}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{122}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & \ldots & \\
\vdots & & & \\
A_{1(\text{rows})1}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{1(\text{rows})2}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{1(\text{rows})3}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & \ldots\ A_{1(\text{rows})(\text{columns})} \\
\text{<blank line>} & & & \\
A_{211}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{212}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{213} & \ldots\ A_{21(\text{columns})} \\
A_{221}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{222}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & \ldots & \\
\vdots & & & \\
A_{2(\text{rows})1}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{2(\text{rows})2}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{2(\text{rows})3}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & \ldots\ A_{2(\text{rows})(\text{columns})} \\
\\
\vdots & & & \\
\\
A_{(\text{planes})11}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{(\text{planes})12}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{(\text{planes})13}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & \ldots\ A_{(\text{planes})1(\text{columns})} \\
A_{(\text{planes})21}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{(\text{planes})22}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & \ldots & \\
\vdots & & & \\
A_{(\text{planes})(\text{rows})1}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{(\text{planes})(\text{rows})2}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & A_{(\text{planes})(\text{rows})3}\left\langle{}^{\text{s}}_{\text{e}}\right\rangle & \ldots\ A_{(\text{planes})(\text{rows})(\text{columns})}
\end{array}
$$

Here, $\left\langle{}^{\text{s}}_{\text{e}}\right\rangle$ is the data separator that is used to parse the input. Terms like $A_{(\text{plane})(\text{row})(\text{column})}$ refers to the LaTeX-formatted data to be associated with the particlar 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

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
`\readdef` 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`.

Regarding the point that end-of-line characters are replaced with a data separator, these are controlled by two separate mechanisms, each with its own control. First, end-of-lines are, by default discarded—this is accomplished internally with readarray setting `\endlinechar` equal to 9, during the `\readdef`. This is not the native LaTeX behavior. In LaTeX, end-of-lines are converted into spaces, unless preempted with a `%` character or a line-ending control sequence. To recover the LaTeX way of treating end-of-lines, the count `\readarrayendlinechar` is provided, which can be set in the manner desired for subsequent invocations of `\readdef`. For example, `\readarrayendlinechar=5` will change subsequent treatment of end-of-line characters to the normal LaTeX treatment.

The second aspect regarding end-of-lines is the data separator inserted when the end-of-line is reached. This is one of the functions of the `\readarraysepchar` macro, the argument to which is the separator inserted into the read, when end-of-line is encountered during a `\readdef`. If you don't want any record separators inserted during the `\readdef`, one may set `\readarraysepchar{}`. Note, however, that an empty readarray sepchar, while valid during a `\readdef`, is meaningless during a `\readarray` invocation, since it would like to use the sepchar to parse the data. Therefore, an empty sepchar should be redefined to a meaningful value prior to invoking `\readarray`. See section 3.2.3 for more information on the separator character.

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
```

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.

\ncols
\nrows
\nrecords
\ArrayRecord

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.

`\ifignoreblankreadarrayrecords`

One option (introduced in v3.1) in this regard is the if-condition `\ifignoreblankreadarrayrecords`, which can take on values of either `\ignoreblankreadarrayrecordstrue` or `\ignoreblankreadarrayrecordsfalse`. When set to the true condition, "blank" records in the input file (defined either when the record is actually blank or if the first non-blank token in the record is a catcode-14 `%`) will be ignored. This setting will only affect the record array itself—the result of any `\readdef` will be unaffected by this setting, except as detailed in the next paragraph.

The only warning to heed when setting `\ignoreblankreadarrayrecordstrue` is that the first blank line in a `\readdef`'ed file is used to set the value of `\nrows`, which can be used to set the row-depth of a 3-D array plane within subsequent invocations of `\readarray`. Of course, the array plane's row-depth may always be set manually, in lieu of using `\nrows`.

[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 from Data

\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.

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.

A key setting, either when inputting information from a file by way of \readdef or when parsing a \def'ed argument into an array by way of \readarray, is the
\readarraysepchar macro \readarraysepchar.

The value of \readarraysepchar is used in distinct ways for \readdef and \readarray. For \readdef, the separator is inserted into the \data-macro definition whenever an end-line-char (end of record) is encountered. For the subsequent \readarray (with or without a *), the \readarraysepchar is used to separate the data contained in the \data-macro into individual cell data that will be placed into the \array-identifier.

For example, if your data file is comma separated, but the last item of data in each record is not followed by a trailing comma, setting \readarraysepchar{,} will assure that the comma gets inserted into the \data-macro after each record, during the \readdef. This will preclude confusion of the last item of each record with the first item of the subsequent record when the \readarray is executed.

If the trailing comma were present in the data file, then \readarraysepchar{} could be set for the \readdef, to avoid inserting an extra comma. However, it would have to be reset to a comma before executing the \readarray, in order to parse the data using the comma to separate cell data.

Finally, if one wished to employ lists rather than arrays (wherein each row could have a variable number of columns, for example), then one could set \readarraysepchar{\\} prior to the \readdef, so as to insert a \\ macro between each record. Then, if the file data were otherwise comma separated, one could use the listofitems tools rather than readarray to absorb the data as a

list, rather than an array. In this case, `\setsepchar{\\/,}` would set a two-tier nested parsing for listofitems. Then, `\readlist\`*listname*`{\`*data-macro*`}` would read the data into a list, with a variable column dimension for each row, based on the number of commas in each file record.

Note also, using the facilities of the underlying listofitems package, that compound separators are possible. For example, when employing `\readarray`. *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 read the file into `\dataC`, adding a comma between records. It will then employ a comma separator to parse the file. As a result, it will create a 3-D array using data from the `\dataC` macro, and create a parsed array by the name of `\tobeConjugation`. During the `\readdef`, it will have been gleaned that the file is arranged with 4 columns (stored in `\ncols`) and 3 rows per plane (stored in `\nrows`). These row & column specifications could be manually overridden merely by specifying actual numbers in the bracketed argument to `\readarray`. 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 Initializing an Array Structure

With the `readarray` macro described above, data can be introduced into an array structure that is specifically created to receive the data. However, sometimes, an array placeholder must be defined before knowing exactly what will eventually reside there. For this purpose, that package provides `\initarray`.

`\initarray`

With a command like `\initarray\z[3,4]`, a 2-D array identified as `\z` is created with 3 rows and 4 columns. Each cell is initialized with a standard value (default `-`). However, the standard initialization value may be changed, for example, to a `0` with an invocation of `\renewcommand\readarrayinitvalue{0}`.

`\readarrayinitvalue`

5 Manually Setting Array-Cell Data

While `\readarray` and `\initarray` are used to create new arrays and to populate their cells with content, we here examine a way to update or change the content of a single existing array cell. The command `\setvalue` is used to place content into an existing array cell. Let us assume that the array `\z` was created as a 2-D array of size 3 rows by 4 columns. The command `\setvalue\z[2,3]{xyz}` will replace the existing content of the array's 2nd row and 3rd column with the tokens `xyz`. All other cells in the `\z` array will remain as they were; the pre-existing content of cell `\z[2,3]`, once replaced, will no longer be available.

`\setvalue`

6 Merging Data from Separate Arrays

The command `\setvalue` was described as a way to replace the contents of a single cell in an array. Here, we describe a way to replace the contents of multiple cells at once, in the form of merging two arrays with the macro `\mergearray`. This command sets up a loop of repeated invocations of `\setarray`.

`\mergearray`

Let's assume that we had created arrays `\Q[10,10]` and `\R[3,2]`. The command `\mergarray\R\Q[6,3]` will overlay the contents of `\R`'s 3x2 cells into the `\Q` cells, beginning with `\Q[6,3]`. The overlay will, therefore, include the cells: `[6,3]`, `[6,4]`, `[7,3]`, `[7,4]`, `[8,3]`, and `[8,4]`. If the overlay of the *from-array* causes the bounds in the *to-array* to be exceeded, no copy is made of those cells. However, a warning will be noted in the log file if error checking is enabled.

7 Recalling Cell Data from Array Structures

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`

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`

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.

array-identifier`[...]`

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*".

8 Typesetting Whole Arrays

Often, one wishes not just to recall the contents of a single cell, but to recall a whole array for the purposes of typesetting. The command `\typesetarray` does just that. Let us create an array `\Q[2,2,2]` whose cell `[i,j,k]` contains the tokens "(i, j, k)", where i, j and k denote a particular plane, row and column of `\Q`. If one invokes `\typesetarray\Q`, the default dump of the array will be:

`\typesetarray`

(1,1,1),(1,1,2)
(1,2,1),(1,2,2)
—
(2,1,1),(2,1,2)
(2,2,1),(2,2,2)

The presentation of the array is governed by four defined macros, whose definitions can be renewed:
`\typesetplanesepchar` (default `\\---\\`),
`\typesetrowsepchar` (default `\\`),
`\typesetcolsepchar` (default `,`) and
`\typesetcell`,
the last of which takes 1 argument (the cell content) and, by default, presents it unaltered. One can see that the default treatment separates columns with a comma, performs a linefeed between each row, and inserts an em-dash on a line by itself between each plane, which is exactly how `\Q` was presented above.

However, these defaults can be renewed to produce a different visual format, for example, that obtainable from a `tabular` environment. For example, upon issuing these renewals

```
\renewcommand\typesetplanesepchar{\\\hline}
\renewcommand\typesetrowsepchar{\\}
\renewcommand\typesetcolsepchar{&}
\renewcommand\typesetcell[1]{\scriptsize\textbf{#1}}
```

one may insert the `\typesetarray` inside a `\tabular` and it will format according to those rules:

```
\begin{tabular}{cc}
\hline
\typesetarray\Q\\
\hline
\end{tabular}
```

yielding the following output:

(1,1,1)	**(1,1,2)**
(1,2,1)	**(1,2,2)**
(2,1,1)	**(2,1,2)**
(2,2,1)	**(2,2,2)**

.

9 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.

10 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.

11 Transfer Array Cell Tokens to Macro

\arraytomacro

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, 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, the macro `\thiscell` contains "`had \textit {been}`".

12 Deprecated and Defunct Features

Deprecated

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

`\arraydump`
`\scalardump`

To replace the function of these, `\typesetarray` is the preferred embodiment.

Defunct

The following macros are no longer supported.

`\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*`}` `\getargsC{\`*macro* or *string*`}`
`\arg`*index*
`\narg`
`\showargs` `\converttilde`
`\record`*index*

13 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.

14 Code Listing

```
\def\readarrayPackageVersion{3.1}
\def\readarrayPackageDate{2021/09/17}
\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.
%
\RequirePackage{forloop}
\RequirePackage{listofitems}[2016-10-22]
%
\newcounter{@index}
\newcounter{@plane}
\newcounter{@row}
\newcounter{@col}
\newcounter{use@args}
\newcounter{@record}
\newcounter{index@count}
\newtoks\Arg@toks
\newtoks\@arrayident@toks
\newread\rdar@file
\newcount\readarrayendlinechar
\newif\ifignoreblankreadarrayrecords
\ignoreblankreadarrayrecordsfalse
\edef\readarraybackslash{\expandafter\@firstoftwo\string\\}
%
\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{%
  \@arrayident@toks=\expandafter\expandafter\expandafter
    {\csname\rdar@macroname#1[#2]\endcsname}%
  \expandafter\def\expandafter#3\expandafter{\the\@arrayident@toks}%
}
%
\newcommand\readarraysepchar[1]{\def\read@array@sepchar{#1}}
%
\def\nocheckbounds{\def\rootmacro@aux##1##2{\csname##1[##2]\endcsname%
 }\typeout{readarray: bounds checking OFF}%
}
%
\def\checkbounds{\def\rootmacro@aux##1##2{%
  \ifcsname##1[##2]\endcsname\csname##1[##2]\endcsname\else%
  \readarrayboundfailmsg%
  \typeout{readarray Warning: \readarraybackslash##1[##2] undefined.}%
  \fi%
 }\typeout{readarray: bounds checking ON}%
}
%
\def\hypercheckbounds{\def\rootmacro@aux##1##2{%
  \ifcsname##1[##2]\endcsname\csname##1[##2]\endcsname\else
    \readarrayboundfailmsg%
    \typeout{readarray Warning: \readarraybackslash##1[##2] undefined:}%
    \setcounter{index@count}{0}%
    \parse@index##2,\relax%
    \forloop{@index}{1}{\value{@index}<\numexpr\theindex@count+1}{%
      \ifnum\parsed@index[\the@index]<1%
        \relax\typeout{ \nonpos@message{##1}{##2}}\fi%
    }%
    \ifnum \value{index@count}=1\relax%
      \ifnum\parsed@index[1]>\csname##1CELLS\endcsname\relax
        \typeout{ \record@message{##1}{##2}}\fi%
    \fi
    \ifnum \value{index@count}=2\relax%
      \ifnum\parsed@index[1]>\csname##1ROWS\endcsname\relax
        \typeout{ \row@message{##1}{\parsed@index[1]}}\fi%
      \ifnum\parsed@index[2]>\csname##1COLS\endcsname\relax
        \typeout{ \col@message{##1}{\parsed@index[2]}}\fi%
    \fi
    \ifnum \value{index@count}=3\relax%
      \ifnum\parsed@index[1]>\csname##1PLANES\endcsname\relax
        \typeout{ \plane@message{##1}{\parsed@index[1]}}\fi%
      \ifnum\parsed@index[2]>\csname##1ROWS\endcsname\relax
        \typeout{ \row@message{##1}{\parsed@index[2]}}\fi%
      \ifnum\parsed@index[3]>\csname##1COLS\endcsname\relax
        \typeout{ \col@message{##1}{\parsed@index[3]}}\fi%
    \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]}}
```

```
%
\def\@readdef#1#2#3{%
  \clear@array{#3}%
  \edef\former@recordcount{\csname #3CELLS\endcsname}%
  \def\nrows{0}%
  \def\first@row{T}%
  \def\first@plane{F}%
  \catcode\endlinechar=\readarrayendlinechar\relax %
  \def#2{}%
  \setcounter{@record}{0}%
  \openin\rdar@file=#1%
  \ifignoreblankreadarrayrecords
    \def\rdar@iftest{\rdar@record\empty}\else\def\rdar@iftest{01}\fi
  \loop\unless\ifeof\rdar@file%
    \read\rdar@file to\rdar@record % Reads record into \rdar@record%
    \expandafter\ifx\rdar@iftest\else
      \stepcounter{@record}%
      \expandafter\g@addto@macro\expandafter#2\expandafter{\rdar@record}%
      \ifx\rdar@record\empty\else\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}{0}%
          \def\first@plane{T}%
        \fi
      \fi
      \if T\first@plane
        \ifx\rdar@record\empty
          \edef\nrows{\arabic{@row}}%
          \def\first@plane{F}%
        \else
          \stepcounter{@row}%
        \fi
      \fi
      \def\record@name{\csname #3[\the@record]\endcsname}%
      \expandafter\expandafter\expandafter\expandafter\expandafter\expandafter
        \expandafter\def\expandafter\record@name\expandafter{\rdar@record}%
    \fi
  \repeat
  \ifnum\nrows=0 \edef\nrows{\arabic{@row}}\fi
  \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]{%
  \setcounter{index@count}{0}%
  \parse@index#5,\relax%
  \ifnum\value{index@count}=1\relax%
```

```
      \def\ra@TermA{\read@Arrayij[#1]{#2}}%
      \edef\ra@TermB{{\rdar@macroname#3}{\parsed@index[1]}}%
      \expandafter\ra@TermA\ra@TermB%
  \else
  \ifnum\value{index@count}=2\relax%
      \def\ra@TermA{\read@Arrayijk[#1]{#2}}%
      \edef\ra@TermB{{\rdar@macroname#3}{\parsed@index[1]}%
       {\parsed@index[2]}}%
      \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}%
  \ifnum\arabic{use@args} > \Arg@listlen\relax
    \addtocounter{@plane}{-1}%
    \setcounter{use@args}{\numexpr\arabic{@plane}*#4*#5}%
  \fi%
  \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{@plane}{1}%
  \setcounter{@row}{1}%
  \setcounter{@col}{0}%
  \forloop{@index}{1}{\value{@index} < \numexpr\value{use@args}+1}{%
    \addtocounter{@col}{1}%
    \ifnum\value{@col} > #5\relax
      \addtocounter{@row}{1}%
      \addtocounter{@col}{-#5}%
    \fi
    \ifnum\value{@row} > #4\relax
      \addtocounter{@plane}{1}%
      \addtocounter{@row}{-#4}%
    \fi
    \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}%
  \ifnum\arabic{use@args} > \Arg@listlen\relax
    \addtocounter{@row}{-1}%
    \setcounter{use@args}{\numexpr\arabic{@row}*#4}%
  \fi
  \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{@row}{1}%
  \setcounter{@col}{0}%
  \forloop{@index}{1}{\value{@index} < \numexpr\value{use@args}+1}{%
    \addtocounter{@col}{1}%
    \ifnum\value{@col} > #4\relax
      \addtocounter{@row}{1}%
      \addtocounter{@col}{-#4}%
    \fi
    \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\read@array[2][]{%
  \bgroup%
  \ifx\empty\read@array@sepchar
    \setsepchar{\empty}%
  \else
    \expandafter\setsepchar\expandafter{\read@array@sepchar}%
  \fi
  \greadlist#1\Arg@list{#2}%
  \egroup%
  \edef\Arg@listCELLS{\Arg@listlen}%
}
%
\def\clear@array#1{%
  \ifcsname #1ROWS\endcsname%
    \forloop{@row}{1}{\value{@row}<\numexpr\csname #1ROWS\endcsname+1}{%
      \ifnum\csname #1COLS\endcsname=0\relax%
        \expandafter\let\csname #1[\the@row]\endcsname\undefined%
      \else
        \forloop{@col}{1}{\value{@col}<\numexpr\csname #1COLS\endcsname+1}{%
          \ifnum\csname #1PLANES\endcsname=0\relax%
            \expandafter\let\csname #1[\the@row,\the@col]\endcsname
              \undefined%
          \else
            \forloop{@plane}{1}{\value{@plane}<\numexpr\csname #1PLANES\endcsname+1}{%
              \expandafter%
                \let\csname #1[\the@plane,\the@row,\the@col]\endcsname
                  \undefined%
            }%
          \fi%
        }%
      \fi%
    }%
    \expandafter\let\csname #1PLANES\endcsname\undefined
    \expandafter\let\csname #1ROWS\endcsname\undefined
    \expandafter\let\csname #1PLANES\endcsname\undefined
    \expandafter\let\csname #1\endcsname\undefined
  \fi%
}
%
```

```
\def\setvalue#1[#2]#3{%
  \ifcsname\rdar@macroname#1[#2]\endcsname
    \expandafter\def\csname\rdar@macroname#1[#2]\endcsname{#3}%
  \else
    \typeout{readarray Warning (setvalue = #3):
      \readarraybackslash\rdar@macroname#1[#2] undefined.}%
  \fi%
}
%
\def\readarray@initializedata#1[#2]#3{%
  \expandafter\def\csname
    \rdar@macroname#1[#2]\expandafter\endcsname\expandafter{#3}%
}
%
\def\initarray#1[#2]{%
  \edef\@tmp{\rdar@macroname#1}%
  \expandafter\clear@array\expandafter{\@tmp}%
  \expandafter\define@rootmacro\expandafter{\@tmp}%
  \setcounter{index@count}{0}%
  \parse@index#2,\relax
  \ifnum\value{index@count}=2\relax
    \setcounter{use@args}{\numexpr\parsed@index[1]*\parsed@index[2]}
    \expandafter\def\csname\@tmp PLANES\endcsname{0}
    \expandafter\edef\csname\@tmp ROWS\endcsname{\parsed@index[1]}
    \expandafter\edef\csname\@tmp COLS\endcsname{\parsed@index[2]}
    \expandafter\edef\csname\@tmp CELLS\endcsname{\theuse@args}
    \forloop{@row}{1}{\value{@row}<\numexpr\parsed@index[1]+1}{%
      \forloop{@col}{1}{\value{@col}<\numexpr\parsed@index[2]+1}{%
        \readarray@initializedata#1[\the@row,\the@col]{\readarrayinitvalue}}}
  \else
    \ifnum\value{index@count}=3\relax
      \setcounter{use@args}{\numexpr\parsed@index[1]*
        \parsed@index[2]*\parsed@index[3]}
      \expandafter\edef\csname\@tmp PLANES\endcsname{\parsed@index[1]}
      \expandafter\edef\csname\@tmp ROWS\endcsname{\parsed@index[2]}
      \expandafter\edef\csname\@tmp COLS\endcsname{\parsed@index[3]}
      \expandafter\edef\csname\@tmp CELLS\endcsname{\theuse@args}
      \forloop{@plane}{1}{\value{@plane}<\numexpr\parsed@index[1]+1}{%
        \forloop{@row}{1}{\value{@row}<\numexpr\parsed@index[2]+1}{%
          \forloop{@col}{1}{\value{@col}<\numexpr\parsed@index[3]+1}{%
            \readarray@initializedata#1[\the@plane,\the@row,\the@col]{%
                                              \readarrayinitvalue}}}}
    \else
      [initarray ERROR: 2-D or 3-D arrays only]
    \fi
  \fi
}
%
\def\mergearray#1#2[#3]{%
  \setcounter{index@count}{0}%
  \parse@index#3,\relax
  \ifnum\value{index@count}=2\relax
    \forloop{@row}{1}{\value{@row}<
                  \numexpr\csname\rdar@macroname#1ROWS\endcsname+1}{%
      \forloop{@col}{1}{\value{@col}<
                  \numexpr\csname\rdar@macroname#1COLS\endcsname+1}{%
        \edef\tmpA{\the\numexpr\the@row+\parsed@index[1]-1,%
```

```
                \the\numexpr\the@col+\parsed@index[2]-1}%
        \edef\tmpB{\csname\rdar@macroname#1[\the@row,\the@col]\endcsname}%
        \def\tmpC{\setvalue#2[\tmpA]}%
        \expandafter\tmpC\expandafter{\tmpB}}}%
  \else
    \ifnum\value{index@count}=3\relax
      \forloop{@plane}{1}{\value{@plane}<
                  \numexpr\csname\rdar@macroname#1PLANES\endcsname+1}{%
        \forloop{@row}{1}{\value{@row}<
                  \numexpr\csname\rdar@macroname#1ROWS\endcsname+1}{%
          \forloop{@col}{1}{\value{@col}<
                  \numexpr\csname\rdar@macroname#1COLS\endcsname+1}{%
            \edef\tmpA{\the\numexpr\the@plane+\parsed@index[1]-1,%
                       \the\numexpr\the@row+\parsed@index[2]-1,%
                       \the\numexpr\the@col+\parsed@index[3]-1}%
            \edef\tmpB{\csname\rdar@macroname#1%
                         [\the@plane,\the@row,\the@col]\endcsname}%
            \def\tmpC{\setvalue#2[\tmpA]}%
            \expandafter\tmpC\expandafter{\tmpB}}}}%
    \else
      [mergearray ERROR: 2-D or 3-D arrays only]
    \fi
  \fi
}
%
\def\addtoArg@toks#1{\Arg@toks\expandafter{\the\Arg@toks#1}}
\def\xaddtoArg@toks#1{\expandafter\addtoArg@toks\expandafter{#1}}
\def\xxaddtoArg@toks#1{\expandafter\xaddtoArg@toks\expandafter{#1}}
%
\def\typesetarray#1{\noindent\Arg@toks{}%
  \ifnum\csname\rdar@macroname#1PLANES\endcsname>0\relax
    \forloop{@plane}{1}{\value{@plane}<
                \numexpr\csname\rdar@macroname#1PLANES\endcsname+1}{%
      \ifnum\the@plane=1 \else\xaddtoArg@toks{\typesetplanesepchar}\fi%
      \forloop{@row}{1}{\value{@row}<
                \numexpr\csname\rdar@macroname#1ROWS\endcsname+1}{%
        \ifnum\the@row=1 \else\xaddtoArg@toks{\typesetrowsepchar}\fi%
        \forloop{@col}{1}{\value{@col}<
                \numexpr\csname\rdar@macroname#1COLS\endcsname+1}{%
          \ifnum\the@col=1 \else\xaddtoArg@toks{\typesetcolsepchar}\fi%
          \xaddtoArg@toks{\expandafter\typesetcell\expandafter
            {\csname\rdar@macroname#1[\the@plane,\the@row,\the@col]\endcsname}}}%
      }%
    }%
  \else
    \forloop{@row}{1}{\value{@row}<
                \numexpr\csname\rdar@macroname#1ROWS\endcsname+1}{%
      \ifnum\the@row=1 \else\xaddtoArg@toks{\typesetrowsepchar}\fi%
      \forloop{@col}{1}{\value{@col}<
                \numexpr\csname\rdar@macroname#1COLS\endcsname+1}{%
        \ifnum\the@col=1 \else\xaddtoArg@toks{\typesetcolsepchar}\fi%
        \xaddtoArg@toks{\expandafter\typesetcell\expandafter
          {\csname\rdar@macroname#1[\the@row,\the@col]\endcsname}}%
      }%
    }%
  \fi
  \the\Arg@toks
```

```
}
%
\def\define@rootmacro#1{%
  \expandafter\def\csname#1\endcsname[##1]{\rootmacro@aux{#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%
}
%
\def\parsed@index[#1]{\csname parsed@index[#1]\endcsname}
%
% INITIALIZATION
% ON \readdef, SEP CHAR INSERTED AFTER EACH RECORD;
% ON \readarray, SEP CHAR SERVES AS DATA-FIELD SEPARATOR
\readarraysepchar{ }
% ON \readdef, IGNORE END-LINE CHARS BY DEFAULT (NORMAL LaTeX MODE = 5)
\readarrayendlinechar=9
% DEFAULT CELL DATA FOR \initarray
\def\readarrayinitvalue{-}
% DEFAULT FIELD SEPARATORS FOR \typesetarray
\def\typesetplanesepchar{\\---\\}
\def\typesetrowsepchar{\\}
\def\typesetcolsepchar{,}
% DEFAULT CELL FORMATTING ON \typesetarray
\def\typesetcell#1{#1}
%
\nocheckbounds% DEFAULT IS NO BOUNDS CHECKING
%
\def\nonpos@message#1#2{Nonpositive index [#2] prohibited for \ra@nm#1.}
\def\record@message#1#2{%
  RECORD=#2 exceeds bounds(=\csname#1CELLS\endcsname) for \ra@nm#1.}
\def\plane@message#1#2{%
  PLANE=#2 exceeds bounds(=\csname#1PLANES\endcsname) for \ra@nm#1.}
\def\row@message#1#2{%
  ROW=#2 exceeds bounds(=\csname#1ROWS\endcsname) for \ra@nm#1.}
\def\col@message#1#2{%
  COL=#2 exceeds bounds(=\csname#1COLS\endcsname) for \ra@nm#1.}
%
\def\readarrayboundfailmsg{?}% WHEN ARRAY CALL OUT OF BOUNDS, IF BOUNDS CHECKING ON
\def\ra@nm#1.{\readarraybackslash#1.}
%
% SUPPORT/DEBUG ROUTINES
% (THESE ARE DEPRECATED...CONSIDER USING \typesetarray AS AN ALTERNATIVE)
%
\def\the@showargs@rule{\kern.2pt\rule{.8ex}{1.6ex}\hspace{.2pt}}%
% \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]{%
```

```
  \expandafter\ifx\csname\rdar@macroname#1\endcsname\relax\else%
    \edef\ra@TmpA{\csname\rdar@macroname#1PLANES\endcsname}%
    \edef\ra@TmpB{\csname\rdar@macroname#1COLS\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%
    \forloop{@plane}{1}{\value{@plane}<\numexpr\ra@TmpA+1}{%
      \plane@msg%
      \forloop{@row}{1}{\value{@row}<
              \numexpr\csname\rdar@macroname#1ROWS\endcsname+1}{%
        \ifnum\value{@row}=1\relax\else\row@spacer\fi%
        \forloop{@col}{1}{\value{@col}<
              \numexpr\ra@TmpB+1}{%
          \the@showargs@rule%
          \ifnum\csname\rdar@macroname#1COLS\endcsname=0\relax%
            #1[\the@row]%
          \else%
            \ifnum\csname\rdar@macroname#1PLANES\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\rdar@macroname####1CELLS\endcsname\ ELEMENTS:}%
    ~\hrulefill\mbox{}\\}%
  \def\last@row{}%
  \def\plane@msg{}%
  \def\close@out{\the@showargs@rule\\}%
  \arraydump#1\egroup%
}

\endinput

% Revisions:
% v1.0  -Initial release.
% 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.
% v3.0  (2021-08-05)
%       -Added features: \setvalue, \initarray, \mergearray, \typesetarray.
%       -Allowed for \readarrayendlinechar to be set other than 9.
% v3.1  (2021-09-17)
%       -Bug fix to not break if first line of file subject to \readdef
%        is blank/comment line.
%       -Introduce \ifignoreblankreadarrayrecords
%       -Bug fix in \clear@array, if selected array name is defined,
%        but not defined as an array.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
COMMANDS THAT WERE DEPRECATED IN v2.0 (2016-11-10), NOT BEING IN THE PREFERRED
PACKAGE SYNTAX, THAT HAVE BEEN ELIMINATED in v3.0 (2021)
CODE PROVIDED BELOW AS A COPY/PASTE LAST RESORT FOR STRAGGLERS

\usepackage{ifthen}
%
% 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}]}%
  }%
}
```