mxNumeric *Matlab numeric Array Class*

Description

The class "mxNumericArray" represents matlab arrays of all numeric types. These include double, single, int8, uint8, int16, uint16, int32, uint32, int64, uint64. Of these, the last three cannot be represented in R. Numeric arrays may be sparse and/or complex.

Details

Matlab supports a number of different numeric types, not all of which map to R types directly. In general, matlab single and *double* values are convertable to R doubles (or complex), and int8, uint8, int16, uint16, and *int32* values are convertable to R integers. The indicated types are the default for conversions. Currently, other matlab types may not be accessed as doing so would result in a loss of precision.

Sparse numeric arrays are not yet supported. Only mxArray operations will succeed.

Objects from the Class

An empty (all 0) array can be created by `new("mxNumericArray", dim=dim, classID=class)`, where `class` is a "mxClassID" object or a string representation thereof.

Slots

complex: logical representing whether this array contains an imaginary component as well.

sparse: logical representing whether this array is stored sparsely. Sparse arrays are not yet supported by this package.

Extends

Class "mxDataArray", directly. Class "mxArray", by class "mxDataArray", distance 2.

Examples

```
mx <- new("mxNumericArray", dim=c(3,2), classID="uint8")
for (i in seq_len(length(mx))) mx[[i]] <- i
as.R.matlab(mx)
as.matlab(sin(seq(0,2*pi, length.out=50)), mxClassID="single")
```

mxLogical *Matlab Logical Array Class*

Description

The class "mxLogicalArray" represents matlab logical arrays. Character arrays are arrays of logicals.

Details

Logical conversion between matlab and R is trivial, with "mxLogicalArray" values convertable to "logical" values. The only exception is that NA is not representable in matlab, so its use is undefined.

Sparse logical arrays are not yet supported. Only mxArray operations will succeed.

Objects from the Class

An empty (all FALSE) array can be created by new("mxLogicalArray", dim=dim).

Slots

sparse: logical representing whether this array is stored sparsely. Sparse arrays are not yet supported by this package.

Extends

Class "mxDataArray", directly. Class "mxArray", by class "mxDataArray", distance 2.

Examples

```
as.R.matlab(as.matlab(array(c(1:18) %% 2 == 0, dim=c(3,3,2))))
```

matFile *Matlab MAT File Interface*

Description

The function matFile provides a high-level interface to matlab MAT files, while the class "matFile" provides a more direct interface. An instance of the class represents a MAT file on disk, which can contain "mxArray" objects.

Usage

```
matFile(file, mode, value)
# High-level interface:
matFile(file) <- value
## S4 method for signature 'matFile':
x[name,recursive=FALSE,drop=FALSE]
## S4 replacement method for signature 'matFile':
x[name] <- value
## S3 method for class 'matFile':
close(con)
# Low-level interface:
## S4 method for signature 'matFile':
names(x)
## S4 method for signature 'matFile':
x[[name]]
## S4 replacement method for signature 'matFile':
x[[name,global=FALSE]] <- value
## S4 method for signature 'matFile':
x$name
## S4 replacement method for signature 'matFile':
x$name <- value
```

Arguments

`file`	path to file to open. Must exist when `mode="r"`.
`mode`	mode with which to open file, one of `c("r","w","u")` for read, write (truncate), or update (read-write append). If missing, read/write entire file and close.
`value`	replacement `"mxArray"` value to create or replace, or `NULL` to delete from file, or named list of such values when writing an entire file.
`x,con`	matFile to operate on.
`name`	name of variable to operate on.
`global`	specifies whether the variable should be written as global.
`recursive,drop`	passed to value's `[`.

Value

`matFile` is a `"matFile"` when mode is specified, or a named list of contents of the specifed file otherwise.

`names` is the list of matlab variable names stored in the matFile.

`[[` and `$` return an `"mxArray"` object or `NULL` if the variable does not exist (or is the empty array).

`[` converts the result to an array, or if `name` is ommitted a list, just as with mxStructArray.

Objects from the Class

Objects can be created by opening/creating a MAT file using `matFile(file, mode)` or `new("matFile", file, mode)`.

Slots

Slots on matFile objects should never be modified directly, but some may be accessed:

`file`: filename opened.

`mode`: access mode to MAT file.

`ptr`: `externalptr` containing underlying `MATFile*`, for access from external functions.

Note

`"matFile"` wraps `MATFile*`.

`matFile` uses `matOpen`. When reading an entire file, it calls `matGetNextVariable` repeatedly. `close` uses `matClose`.

`[[` and related functions use `matGetVariable`, and `[[<-` uses `matPutVariable`, `matPutVariableAsGlobal`, or `matDeleteVariable`.

Examples

```
file <- "tmp.mat"
matFile(file) <- list(val=1)

mat <- matFile(file, mode="u")
mat$val[[1]] <- mat$val[[1]] + 1
close(mat)

matFile(file)
file.remove(file)
```

---

mxClassID *Matlab Datatype Classes*

Description

The class `"mxClassID"` represents the possible types or classes of matlab data as represented by `"mxArray"` objects.

Objects from the Class

Objects can be created by calls of the form `new("mxClassID", class)`, although character strings may be used instead in most places they are expected.

Extends

Class `"factor"`, directly.

Examples

```
as("double", "mxClassID")
```

---

mxData *Matlab Generic Array Class*

Description

The class "mxDataArray" represents most matlab arrays of scalars. This class is abstract, and no objects should inherit directly from it, although it provides most of the common useful methods.

Usage

```
## S4 method for signature 'mxDataArray':
x[[i, j, ...]]
## S4 replacement method for signature 'mxDataArray':
x[[i, j, ...]] <- value
```

Arguments

x	mxDataArray object to be manipulated.
i,j,...	indices of element to extract or replace. These follow matlab semantics with 1-based indices, each representing an offset scaled by the product of the lower dimensions.
value	the replacement element, which must be coercible to the correct type.

Details

Single array elements can be accessed and updated just as in matlab. Since arrays are references, updating an element modifies the underlying array, and so other references to the same array will reflect this as well. The treatment of elements themselves depend on the particular subclass.

mxDataArray values are also coercible to (and, depending on the type, from) "vector" or "array" using as. However, use of the more general [or as.R.matlab (and as.matlab) is preferred.

Objects from the Class

Objects should not be created directly, but rather though constructors and conversion on subclasses.

Note

[[And [[<- use mxCalcSingleSubscript.

`as.matlab` *Convert To/From Matlab Data*

Description

Marshalls matlab arrays to or from equivalent R data structures. If any non-trival access to or manipulation of matlab data is required, use of these functions is recommended.

Usage

```
as.R.matlab(x, recursive=TRUE)
as.matlab(x, mxClassID=attr(x, "mxClassID"))
## S4 method for signature 'mxDataArray':
x[i, j, ..., recursive=FALSE, drop=FALSE]
## S4 replacement method for signature 'mxDataArray':
x[i, j, ...] <- value
## S4 method for signature 'mxStructArray':
x[i, j, ..., field=NULL, recursive=FALSE, drop=FALSE]
## S4 replacement method for signature 'mxStructArray':
x[i, j, ..., field] <- value
```

Arguments

`x`	data to convert.
`mxClassID`	class of mxArray to create (defaults to automatic selection).
`recursive`	recursively convert all contained matlab data as well.
`drop`	apply `drop` to the resulting array.
`i,j,...,field`	indices of element to convert and assign just as with `[[`, or empty for the entire array.
`value`	value to convert and assign to the specified element.

Details

These functions convert between equivalent matlab and R representations of data.

Matlab data may be a null, cell, struct, logical, char, (complex) double, (complex) single, int8, uint8, int16, uint16, or int32 array. (Other data types are not yet supported or not representable in R without loss of precision.)

R data may be NULL or a logical, integer, double, complex, character, or list (containing any of these) vector, matrix, or array.

`as.R.matlab` always returns NULL or an array with the `"mxClassID"` attribute set. This allows `as.matlab` to retore the original data class when passed data from `as.R.matlab`.

`[` and `[<-` provide simpler alternatives, combining some of the functionality of the `as` functions with the `[[` and `[[<-` methods. As a special case, `x[]` can be used to convert or assign the entire array, and follows recycling rules on assignment. Otherwise, indexing rules are just as with `[[`, and in particular only single array elements can be accessed.

Value

`as.R.matlab` and `[` return a logical, integer, double, complex, character, list, or NULL array.

`as.matlab` returns a newly created `"mxArray"` object.

Examples

```
l <- list(a=1:3, b="abc", z=matrix(c(0,pi,pi,2*pi), 2, 2))
struct <- as.matlab(l)
struct$z[] <- c(0,2) # recycled
a <- struct[recursive=TRUE]
a[['z',1,1]][,2]
```

---

mxStruct *Matlab Structure Class*

Description

The class `"mxStructArray"` represents matlab struct arrays. Structs are much like cell arrays with an extra, named dimension of fields, which for the purpose of this class is considered to be the first dimension.

Usage

```
## S4 method for signature 'mxStructArray':
dim(x)
## S4 replacement method for signature 'mxStructArray':
dim(x) <- value
## S4 method for signature 'mxStructArray':
dimnames(x)
mxAddField(mx, field)
mxRemoveField(mx, field)
## S4 method for signature 'mxStructArray':
x[[i, j, ..., field=NULL]]
## S4 replacement method for signature 'mxStructArray':
x[[i, j, ..., field]] <- value
## S4 method for signature 'mxStructArray':
x$field
## S4 replacement method for signature 'mxStructArray':
x$field <- value
```

Arguments

`mx,x`	mxStructArray object to be manipulated.
`field`	name (or index number) of field to be manipulated.
`i,j,...`	indices of element to extract or replace. If `field` is not specified and the first or last index is a character, it is used as the field. For access only, if `field` is `NULL`, a named list is returned with all fields for that index.
`value`	the replacement element, which must be either `NULL` or `"mxArray"`.

Details

Structures are represented as n+1-dimensional arrays, where the first dimension is the field. Accordingly, dim is c(length(fields), SIZE).

The field names and count may only be modified by mxAddField and mxRemoveField. Removing a field may leak non-NULL elements in that field.

Assigning an element will explicitly destroy the previous contents, invalidating any existing references to it. Likewise, when an mxStructArray is no longer referenced, all its contained mxArrays are freed along with it. If this presents a problem, see mxCopy.

$ and $<- work in the obvious way on scalar mxStructArrays only.

Value

dimnames associates the field names to the first dimension, so rownames is often preferable.

mxAddField and mxRemoveField return the updated mx. The underlying data is modified in-place, so the old value of mx is invalid after the call.

Objects from the Class

An empty (all NULL) struct can be created by new("mxStructArray", dim=dim, fields=fields).

Extends

Class "mxArray", directly.

Slots

fields: character vector of field names, like FIELDNAMES. rownames may also be used and is preferred.

Note

mxAddField and mxRemoveField use the matlab functions of the same name.

[[And [[<- use mxGetFieldByNumber and mxSetFieldByNumber respectively.

Examples

```
# create a 2-element row-vector struct array with three fields
mx <- new("mxStructArray", dim=c(1,2), fields=c("a","b","c"))

# reference mx(1).b three ways:
mx[1,field="b"] <- 3.14
mx[[1,1]]$b
mx[['b']]

mx <- mxAddField(mx, "d")
mx <- mxRemoveField(mx, "a")
# remove field "c" (the second field):
mx <- mxRemoveField(mx, 2)
rownames(mx)
```

`matlabEngine` *Matlab Engine Interface*

Description

This class provides an interface to matlab, allowing arbitrary matlab functions and expressions to be evaluated by running and communicating with a separate matlab process through the matlab engine library.

The function `matlabEngine` creates a new matlab process which can be terminated with `close`.

Usage

```
matlabEngine(cmd = getOption("matlabBin"), args = c("-nosplash","-nojvm"))
## S3 method for class 'matlabEngine':
close(con)
## S4 method for signature 'matlabEngine':
x[[name]]
## S4 replacement method for signature 'matlabEngine':
x[[name]] <- value
## S4 method for signature 'matlabEngine':
x[name,recursive=FALSE,drop=FALSE]
## S4 replacement method for signature 'matlabEngine':
x[name] <- value
## S4 method for signature 'matlabEngine':
evaluate(object, command, ..., output=FALSE, nargout=1, tmp.var.prefix="Reval_")
## S4 method for signature 'matlabEngine':
x$name
## S4 replacement method for signature 'matlabEngine':
x$name <- value
```

Arguments

`cmd`	path to matlab binary to start.
`args`	command line arguments to pass to matlab command.
`x,con,object`	matlabEngine to interface with.
`name`	name of variable to operate on.
`value`	replacement "`mxArray`" value to assign to variable.
`recursive,drop`	passed to value's `[`.
`command`	function name or matlab expression string to evaluate.
`...`	arguments to pass to matlab function (after passing through `as.matlab`). If these are passed, it is assumed `command` is a simple function name, and will be assigned to variables of the form `paste(tmp.var.prefix, "in", 1:N, sep="")` to be passed as arguments. If an argument is a name, the matlab variable by this name will be passed instead.

`output` number of bytes of standard output to capture and return (`TRUE` defaults to a reasonable value. By default, output generated by the command is lost.

`nargout` number of result values to assign and return. If this value or arguments are specified, the (first command in the) expression or function will be assigned to variables of the form `paste(tmp.var.prefix, "out", 1:nargout, sep="")` to be returned.

`tmp.var.prefix`
prefix for temporary matlab variable names.

Value

`matlabEngine` returns the new `"matlabEngine"` object representing the connection to the matlab process.

`[[` returns an `"mxArray"` object or `NULL` if the variable does not exist (or is the empty array). `[` converts the result to an array.

`evaluate` returns the `nargout` `"mxArray"` objects resulting from the evaluation. If `output` is specified, the output buffer string will be returned as the last (named `"output"`) item of this list. If there is only one item to return (object or output), it will not be wrapped in a list.

`$` returns either the value of the named parameter if it exists, as with `[[`, or a function representing the named call that can be called with all the same arguments as `evaluate` from `...` on.

Slots

Slots on matlabEngine objects should never be modified directly, but some may be accessed:

`cmd`: command line used to start matlab.

`ptr`: `externalptr` containing underlying `Engine*`, for access from external functions.

Note

`"matlabEngine"` wraps `Engine*`.

`matlabEngine` uses `engOpen`, and `close` uses `engClose`.

`[[` and similar use `engGetVariable` and `[[<-` uses `engPutVariable`.

`evaluate` uses `engEvalString` with `engOutputBuffer`.

Examples

```
eng <- matlabEngine()
eng['x'] <- as.double(3:8)
s <- eng$std(quote(x), 1)
close(eng)
```

mxArray *Matlab Array Root Class*

Description

The class "mxArray" represents all matlab data (arrays). Every matlab object is an instance of this class, which is a wrapper around the underlying matlab representation. Most matlab objects are arrays, much like R arrays, with a certain number of dimensions (at least 2) and a size for each dimension. Most objects should not inherit directly from this class, however some unusual array types that are not representable in R may.

In general, it is preferable to work with the equivalent R representation of mxArray objects. See as.R.matlab and as.matlab.

Usage

```
## S4 method for signature 'mxArray':
dim(x)
## S4 replacement method for signature 'mxArray':
dim(x) <- value
## S4 method for signature 'mxArray':
length(x)
mxGlobal(mx, global)
```

Arguments

mx,x	mxArray to operate on.
value	integer vector of new array dimensions, which will be padded with 1s if its length is less than 2.
global	logical value used to set global variable flag if specified.

Details

Most operations conform to the matlab semantics. In particular, indexing is done as with subscripts in matlab, with each index representing a 1-based offset in that dimension, wrapping into the next dimensions as indices exceed the size. Also, NA values are in general not representable in matlab, so attempts to convert them may have undefined results.

Making an mxCellArray or mxStructArray smaller may leak non-NULL elements beyond the new end.

Objects may be coerced to "raw" to get the underlying data.

Value

dim, like SIZE, returns an integer vector of dimensions of length at least 2.

length, like NUMEL (not LENGTH), returns the total number of elements.

mxGlobal returns a logical of the current setting of the variable's global workspace flag.

Objects from the Class

Objects should not be created directly, but rather though constructors and conversion on subclasses.

Every matlab object is a type of mxArray. Those that contain data accessible from R are further sub-classed according to their format and contents. Note that all of these objects are only references to the underlying data, so modifications will affect (and possibly invalidate) other references to the same data (see `mxCopy`).

The emtpy array (`[]`) is represented by `NULL` rather than an mxArray object.

Slots

Slots on mxArray objects should never be modified directly, but some may be accessed:

`classID:` matlab class of array, of class `"mxClassID"`, like `CLASS`.

`global:` logical representing state of global variable flag, like `ISGLOBAL`.

`ptr:` `externalptr` containing underlying `mxArray*`, for access from external functions.

Note

`"mxArray"` wraps `mxArray*`.

`dim` and `dim<-` use `mxGetDimensions` and `mxSetDimensions` respectively.

`length` uses `mxGetNumberOfElements`.

`mxGlobal` uses `mxIsFromGlobalWS` and possibly `mxSetFromGlobalWS`.

Coercing to `"raw"` returns `mxGetElementSize*mxGetNumberOfElements` bytes from `mxGetData`.

---

mxCell *Matlab Cell Array Class*

Description

The class `"mxCellArray"` represents matlab cell arrays. Cell arrays can contain any other mxArray type.

Details

Cell arrays can contain only NULL and `"mxArray"` values. In R, they are represented as list arrays of these values, and are convertible to/from lists using `as.R.matlab` and `as.matlab`.

Assigning an element will explicitly destroy the previous contents, invalidating any existing references to it. Likewise, when an mxCellArray is no longer referenced, all its contained mxArrays are freed along with it. If this presents a problem, see `mxCopy`.

Objects from the Class

An empty cell array can be created by `new("mxCellArray", dim=dim)`.

Extends

Class "mxDataArray", directly. Class "mxArray", by class "mxDataArray", distance 2.

Note

[[And [[<- use mxGetCell and mxSetCell respectively.

Examples

```
mx <- new("mxCellArray", dim=c(3,2))
mx[[1,1]] <- as.matlab(3.14)
mx[[2]] <- as.matlab('abc')
# no-op: already NULL
mx[[3,1]] <- as.matlab(NULL)
mx[[4]] <- as.matlab(c(1,2))
mx[[2,2]] <- as.matlab(list(3.14, 'abc'))
as.R.matlab(mx, recursive=FALSE)
```

---

mxChar *Matlab Character Array Class*

Description

The class "mxCharArray" represents matlab character arrays. Character arrays are simply that: arrays of single (possibly unicode) characters. Note that matlab handles strings very differently than R, where character arrays are actually arrays of strings.

Details

Because of differences between the R and matlab representation of strings, the exact handling of mxCharArrays may depend on the content.

Single element access, and explicit coersion to/from "array" is done by treating the R array as an array of characters, where each element has at most 1 character (length(mx[[i]]) == nchar(mx[[i]]) == 1), or 0 if the character is '\0'. Since matlab characters may be unicode characters, this may result in truncation.

On the other hand, scalar strings in R and simple row-vectors of characters in matlab are compatible, so coersion to/from "character" results in length(to.R.matlab(mx)) == 1 and nchar(to.R.matlab(mx)) == length(mx) in this case. Here, some unicode translation is done on the matlab side, so these lengths might not match exactly.

Objects from the Class

An empty (all '\0') array can be created by new("mxCharArray", dim=dim).

Extends

Class "mxDataArray", directly. Class "mxArray", by class "mxDataArray", distance 2.

Examples

```
mx <- new("mxCharArray", dim=c(1,3))
mx[[1]] <- 'a'
# 'xxx' is discarded
mx[[2]] <- 'bxxx'
mx[[1,3]] <- 'c'
# abc
as.R.matlab(mx)

dim(mx) <- c(3,1)
# now an array
as.R.matlab(mx)
```

---

mxCopy *Duplicate mxArray*

Description

Duplicates and copies the memory associated with an mxArray. Usually used for the purpose of assigning a local mxArray to an element of another one which will have a shorter lifetime.

Usage

```
mxCopy(mx)
```

Arguments

mx	mxArray object to be copied.

Details

mxArray objects are all references. Normally, when the R garbage collection determines than an mxArray is no longer referenced, it will also free the underlying storage associated with it, as well as that for all mxArrays contained therein (e.g., in cells or structures). This can present a problem in the case where an element of this mxArray is still associated with an R symbol, either because the mxArray was subscripted or assigned to. In this case, the element should be duplicated to prevent it being freed along with its container and the likely resultant crash.

Value

mxCopy returns an mxArray that is identical to mx but that has its own copy of the underlying storage.

Note

Uses mxDuplicateArray.

Examples

```
## extracting an element:
x <- as.matlab(list(1:3, 'abc'))
# bad, corrupts 'y': y <- x[[2]] ; rm(x)
# copy to prevent deletion with 'x'
y <- mxCopy(x[[2]])
rm(x)
y

## replacing an element:
mxPi <- as.matlab(pi)
x <- new('mxCellArray', dim=1)
# bad, corrupts mxPi: x[[1]] <- mxPi ; rm(x)
# copy to prevent deletion with 'x'
x[[1]] <- mxCopy(mxPi)
rm(x)
mxPi
```

`Rmatlab-package` *Matlab bindings and interface.*

Description

This package aims to provide a comprehensive interface to the MathWorks MATLAB(R) libraries and native data structures, including complete matrix access, MAT-format files, linking and execution of runtime libraries and engine. Requires MATLAB for full functionality. This has been tested with MATLAB 7.7-7.8 and should work with other versions as well.

Details

Package:	Rmatlab
Type:	Package
Version:	0.1
Date:	2009-07-25
License:	BSD3
LazyLoad:	yes

The primary interface to matlab data is the `"mxArray"` class and its children.

Access to MATLAB MAT save files is handled through `matFile`.

Note

When referencing matlab functions in documentation, `ALLCAPS` format is used.

The package R.matlab provides some overlapping functionality to this package, but in a pure R implementation rather than relying on the MATLAB libraries.

Author(s)

Dylan Simon <dylan@dylex.net>