Package 'R.matlab'

September 9, 2026

Version 3.8.0

Depends R (>= 2.14.0)

Imports methods, utils, R.methodsS3 (>= 1.7.1), R.oo (>= 1.23.0), R.utils (>= 2.5.0)

Suggests Matrix, SparseM

Title Read and Write MAT Files and Call MATLAB from Within R

Author Henrik Bengtsson [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-7579-5165>), Andy Jacobson [ctb] (Internal MAT v4 reader, ORCID: <https://orcid.org/0000-0002-8501-3382>), Jason Riedy [ctb] (Support for reading compressed files, sparse matrices and UTF-encoded strings., ORCID: <https://orcid.org/0000-0002-4345-4200>)

Maintainer Henrik Bengtsson <henrikb@braju.com>

Description Methods readMat() and writeMat() for reading and writing MAT files. For users with MATLAB v6 or newer installed (either locally or on a remote host), the package also provides methods for controlling MATLAB (trademark) via R and sending and retrieving data between R and MATLAB.

License LGPL (>= 2.1)

URL https://github.com/HenrikBengtsson/R.matlab

BugReports https://github.com/HenrikBengtsson/R.matlab/issues

Language en-US

Encoding UTF-8

LazyLoad TRUE

NeedsCompilation no

Repository CRAN

Date/Publication 2026-09-09 13:00:14 UTC

Contents

R.matlab-package 2
1. The MATLAB server running in MATLAB 5
Matlab 16
readMat 23
writeMat 26

Index **29**

`R.matlab-package` *Package R.matlab*

Description

Methods readMat() and writeMat() for reading and writing MAT files. For user with MATLAB v6 or newer installed (either locally or on a remote host), the package also provides methods for controlling MATLAB (trademark) via R and sending and retrieving data between R and MATLAB.

In brief, this package provides a one-directional interface from R to MATLAB, with communication taking place via a TCP/IP connection and with data transferred either through another connection or via the file system. On the MATLAB side, the TCP/IP connection is handled by a small Java add-on.

The methods for reading and writing MAT files are stable. The R to MATLAB interface, that is the Matlab class, is less prioritized and should be considered a beta version.

For package history, see `showHistory(R.matlab)`.

Requirements

This is a cross-platform package implemented in plain R. This package depends on the **R.oo** package [1].

To use the Matlab class or requesting verbose output messages, the R.utils package is loaded when needed (and therefore required in those cases).

The `readMat()` and `writeMat()` methods do *not* require a MATLAB installation neither do they depend on the `Matlab` class.

To connect to MATLAB, MATLAB v6 or higher is required. It does *not* work with MATLAB v5 or before (because those versions do not support Java). For confirmed MATLAB versions, see the `Matlab` class.

Installation

To install this package do

```
install.packages("R.matlab")
```

To get started

To get started, see:

1. readMat() and writeMat() - For reading and writing MAT files (MATLAB is *not* needed).
2. Matlab - To start MATLAB and communicate with it from R.

Miscellaneous

To call R from MATLAB on Windows (only), see *MATLAB R-link* by Robert Henson available at the MATLAB Central File Exchange (https://www.mathworks.com/matlabcentral/fileexchange/5051-matlab-r-link).

How to cite this package

Whenever using this package, please cite as

Troubleshooting

In general:
For trouble shooting in general, rerun erroneous function with verbose/debug messages turned on. For `readMat()` and `writeMat()` see their help. For communication with a MATLAB server, use

```
matlab <- Matlab()
setVerbose(matlab, threshold = -2)
```

The lower the threshold is the more information you will see.

Cannot connect to MATLAB:
If R fails to connect to MATLAB, make sure to try the example in help(Matlab) first. Make sure that the MATLAB server is running before trying to connect to it from R first. If MATLAB is running but `open()` times out, make sure MATLAB is listening to the same port that R is trying to connect to. If that does not help, try to increase the time-out limit, see help(open.Matlab).

Expected an 'answer' from MATLAB, but kept receiving nothing.:
When launching a really long MATLAB process by `evaluate()`, you may get the above error message.
Reason: This happens because `evaluate()` expect a reply from MATLAB as soon as MATLAB is done. The waiting should be "blocked", i.e. it should wait until it receives something. For unknown reasons, this is not always happening. The workaround we have implemented is to try `readResult/maxTries` waiting `readResult/interval` seconds in-between.
Solution: Increase the total waiting time by setting the above options, e.g.

```
setOption(matlab, "readResult/interval", 10)     # Default is 1 second
setOption(matlab, "readResult/maxTries", 30 * (60 / 10)) # ~30 minutes
```

Wishlist

Here is a list of features that would be useful, but which I have too little time to add myself. Contributions are appreciated.

- Add a function, say, `Matlab$createShortcut()` which creates a Windows shortcut to start the MATLAB server by double clicking it. It should be possible to create it in the current directory or to the Desktop. Maybe it is possible to do this upon installation and even to a Start -> All Programs -> R menu.
- To improve security, update the MatlabServer.m script to allow the user to specify a "password" to be send upon connection from R in order for MATLAB to accept the connection. This password should be possible to specify from the command line when starting MATLAB. If not given, no password is required.
- Add additional methods to the Matlab class. For instance, inline function in MATLAB could have its own method.
- Wrap up common MATLAB commands as methods of the Matlab class, e.g. `who(matlab)`, `clear(matlab)` etc. Can this be done automatically using "reflection", so that required arguments are automatically detected?
- Add access to MATLAB variables via `"$"` and `"$<-"`, e.g. `matlab$A` and `matlab$A <- 1234`. Is this wanted? Maybe the same for functions, e.g. `matlab$dice(1000)`. Is it possible to return multiple return values?

If you consider implement some of the above, make sure it is not already implemented by downloading the latest "devel" version!

Acknowledgments

Thanks to the following people who contributed with valuable feedback, suggestions, code and more:

- Patrick Drechsler, Biocenter, University of Wuerzburg.
- Spencer Graves.
- Andy Jacobson, Atmospheric and Oceanic Sciences Program, Princeton University.
- Jason Riedy, Computer Science Division, University of California, Berkeley.
- Chris Sims, Department of Economics, Princeton University.
- Frank Stephen, National Renewable Energy Laboratory.
- Yichun Wei, Department of Biological Sciences, University of Southern California.
- Wang Yu, ECE Department, Iowa State University.

License

The releases of this package is licensed under LGPL version 2.1 or newer.

The development code of the packages is under a private license (where applicable) and patches sent to the author fall under the latter license, but will be, if incorporated, released under the "release" license above.

References

[1] H. Bengtsson, *The R.oo package - Object-Oriented Programming with References Using Standard R Code*, In Kurt Hornik, Friedrich Leisch and Achim Zeileis, editors, Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003), March 20-22, Vienna, Austria. https://www.r-project.org/conferences/DSC-2003/Proceedings/

Author(s)

Henrik Bengtsson

---

1. The MATLAB server running in MATLAB

1. The MATLAB server running in MATLAB

Description

This section gives addition details on the MATLAB server. At the end, the MatlabServer.m script and the InputStreamByteWrapper.java code is shown.

Starting the MATLAB server on Windows

Note that you "cannot prevent MATLAB from creating a window when starting on Windows systems, but you can force the window to be hidden, by using " the option -minimize. See https://www.mathworks.com/matlabcentral/answers/102082 for more information.

MatlabServer.m script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MatlabServer
%
% This scripts starts a minimalistic MATLAB "server".
%
% When started, the server listens for connections at port 9999 or the
% port number specified by the environment variable 'MATLABSERVER_PORT'.
%
% Troubleshooting: If not working out of the box, add this will to the
% MATLAB path. Make sure InputStreamByteWrapper.class is in the same
% directory as this file!
%
% Requirements:
% This requires MATLAB with Java support, i.e. MATLAB v6 or higher.
%
% Author: Henrik Bengtsson, 2002-2016
%
% References:
```

```
% [1] http://www.mathworks.com/access/helpdesk/help/techdoc/
%                                      matlab_external/ch_jav34.shtml#49439
% [2] http://staff.science.uva.nl/~horus/dox/horus2.0/user/
%                                                  html/n_installUnix.html
% [3] http://www.mathworks.com/access/helpdesk/help/toolbox/
%                                                modelsim/a1057689278b4.html
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf(2, 'Running MatlabServer v3.5.9-9000\n');

%  addpath R/R_LIBS/linux/library/R.matlab/misc/

% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% MATLAB version-dependent setup
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% Identify major version of Matlab
MatlabServer_tmp_hasMajor = eval('length(regexp(version, ''^[0-9]'')) ~= 0', '0');
if (MatlabServer_tmp_hasMajor)
  MatlabServer_tmp_verParts = sscanf(version, '%d.');
  MatlabServer_tmp_verMajor = MatlabServer_tmp_verParts(1);
  clear MatlabServer_tmp_verParts;
else
  MatlabServer_tmp_verMajor = -1;
end
clear MatlabServer_tmp_hasMajor;

if (MatlabServer_tmp_verMajor < 6)
  % Java is not available/supported
  error('MATLAB v5.x and below is not supported.');
elseif (MatlabServer_tmp_verMajor == 6)
  fprintf(2, 'MATLAB v6.x detected.\n');
  % Default save option
  MatlabServer_saveOption = '';
  % In MATLAB v6 only the static Java CLASSPATH is supported. It is
  % specified by a 'classpath.txt' file. The default one can be found
  % by which('classpath.txt'). If a 'classpath.txt' exists in the
  % current(!) directory (that MATLAB is started from), it *replaces*
  % the global one. Thus, it is not possible to add additional paths;
  % the global ones has to be copied to the local 'classpath.txt' file.
  %
  % To do the above automatically from R, does not seem to be an option.
else
  fprintf(2, 'MATLAB v7.x or higher detected.\n');
  % MATLAB v7 and above saves compressed files, which is not recognized
  % by R.matlab's readMat(); force saving in old format.
  MatlabServer_saveOption = '-V6';
  fprintf(2, 'Saving with option -V6.\n');

  % In MATLAB v7 and above both static and dynamic Java CLASSPATH:s exist.
```

```
  % Using dynamic ones, it is possible to add the file
  % InputStreamByteWrapper.class to CLASSPATH, given it is
  % in the same directory as this script.
  javaaddpath({fileparts(which('MatlabServer'))});
  fprintf(2, 'Added InputStreamByteWrapper to dynamic Java CLASSPATH.\n');
end
clear MatlabServer_tmp_verMajor;


% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% Import Java classes
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
import java.io.*;
import java.net.*;


% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% If an old MATLAB server is running, close it
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% If a server object exists from a previous run, close it.
if (exist('MatlabServer_server'))
  close(MatlabServer_server);
  clear MatlabServer_server;
end

% If an input stream exists from a previous run, close it.
if (exist('MatlabServer_is'))
  close(MatlabServer_is);
  clear MatlabServer_is;
end

% If an output stream exists from a previous run, close it.
if (exist('MatlabServer_os'))
  close(MatlabServer_os);
  clear MatlabServer_os;
end

fprintf(2, '----------------------\n');
fprintf(2, 'MATLAB server started!\n');
fprintf(2, '----------------------\n');

fprintf(2, 'MATLAB working directory: %s\n', pwd);


% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% Initiate server socket to which clients may connect
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
MatlabServer_port = getenv('MATLABSERVER_PORT');
```

```
if (length(MatlabServer_port) > 0)
  MatlabServer_port = str2num(MatlabServer_port);
else
  % Try to open a server socket on port 9999
  MatlabServer_port = 9999;
end

% Ports 1-1023 are reserved for the Internet Assigned Numbers Authority.
% Ports 49152-65535 are dynamic ports for the OS. [3]
if (MatlabServer_port < 1023 | MatlabServer_port > 65535)
 error('Cannot not open connection. Port (''MATLABSERVER_PORT'') is out of range [1023,65535]: %d', Mat
end

fprintf(2, 'Trying to open server socket (port %d)...', MatlabServer_port);
MatlabServer_server = java.net.ServerSocket(MatlabServer_port);
fprintf(2, 'done.\n');


% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% Wait for client to connect
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% Create a socket object from the ServerSocket to listen and accept
% connections.
% Open input and output streams

% Wait for the client to connect
fprintf(2, 'Waiting for client to connect (port %d)...', MatlabServer_port);
MatlabServer_clientSocket = accept(MatlabServer_server);
fprintf(2, 'connected.\n');

% ...client connected.
MatlabServer_is = java.io.DataInputStream(getInputStream(MatlabServer_clientSocket));
MatlabServer_os = java.io.DataOutputStream(getOutputStream(MatlabServer_clientSocket));



% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% The MATLAB server state machine
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% Commands
MatlabServer_commands = {'eval', 'send', 'receive', 'send-remote', 'receive-remote', 'echo', 'evalc'};

MatlabServer_lasterr = [];
MatlabServer_variables = [];

% As long as we receive data, echo that data back to the client.
MatlabServer_state = 0;
while (MatlabServer_state >= 0),
```

```
 if (MatlabServer_state == 0)
   MatlabServer_tmp_cmd = readByte(MatlabServer_is);
   fprintf(2, 'Received cmd: %d\n', MatlabServer_tmp_cmd);
  if (MatlabServer_tmp_cmd < -1 | MatlabServer_tmp_cmd > length(MatlabServer_commands))
     fprintf(2, 'Unknown command code: %d\n', MatlabServer_tmp_cmd);
   else
     MatlabServer_state = MatlabServer_tmp_cmd;
   end
   clear MatlabServer_tmp_cmd;

 %------------------
 % 'evalc'
 %------------------
elseif (MatlabServer_state == strmatch('evalc', MatlabServer_commands, 'exact'))
   MatlabServer_tmp_bfr = char(readUTF(MatlabServer_is));
   fprintf(2, '"evalc" string: "%s"\n', MatlabServer_tmp_bfr);
   try
     MatlabServer_tmp_bfr = sprintf(MatlabServer_tmp_bfr);
     MatlabServer_tmp_result = evalc(MatlabServer_tmp_bfr);
     writeByte(MatlabServer_os, 0);
     fprintf(2, 'Sent byte: %d\n', 0);
     writeUTF(MatlabServer_os, MatlabServer_tmp_result);
     fprintf(2, 'Sent UTF: %s\n', MatlabServer_tmp_result);
     flush(MatlabServer_os);
     clear MatlabServer_tmp_result;
   catch
   MatlabServer_lasterr = sprintf('Failed to evaluate expression ''%s''.', MatlabServer_tmp_bfr);
     fprintf(2, 'EvaluationException: %s\n', MatlabServer_lasterr);
     writeByte(MatlabServer_os, -1);
     fprintf(2, 'Sent byte: %d\n', -1);
     writeUTF(MatlabServer_os, MatlabServer_lasterr);
     fprintf(2, 'Sent UTF: %s\n', MatlabServer_lasterr);
     flush(MatlabServer_os);
   end
   flush(MatlabServer_os);
   MatlabServer_state = 0;
   clear MatlabServer_tmp_bfr;

 %------------------
 % 'eval'
 %------------------
elseif (MatlabServer_state == strmatch('eval', MatlabServer_commands, 'exact'))
   MatlabServer_tmp_bfr = char(readUTF(MatlabServer_is));
   fprintf(2, '"eval" string: "%s"\n', MatlabServer_tmp_bfr);
   try
     eval(MatlabServer_tmp_bfr);
     writeByte(MatlabServer_os, 0);
     fprintf(2, 'Sent byte: %d\n', 0);
```

```
    flush(MatlabServer_os);
  catch
  MatlabServer_lasterr = sprintf('Failed to evaluate expression ''%s''.', MatlabServer_tmp_bfr);
    fprintf(2, 'EvaluationException: %s\n', MatlabServer_lasterr);
    writeByte(MatlabServer_os, -1);
    fprintf(2, 'Sent byte: %d\n', -1);
    writeUTF(MatlabServer_os, MatlabServer_lasterr);
    fprintf(2, 'Sent UTF: %s\n', MatlabServer_lasterr);
    flush(MatlabServer_os);
  end
  flush(MatlabServer_os);
  MatlabServer_state = 0;
  clear MatlabServer_tmp_bfr;

%-------------------
% 'send'
%-------------------
elseif (MatlabServer_state == strmatch('send', MatlabServer_commands, 'exact'))
 MatlabServer_tmp_tmpname = sprintf('%s_%d.mat', tempname, MatlabServer_port);
MatlabServer_tmp_expr = sprintf('save(MatlabServer_tmp_tmpname, ''%s''', MatlabServer_saveOption);
  MatlabServer_tmp_ok = 1;
  for MatlabServer_tmp_k=1:length(MatlabServer_variables),
    MatlabServer_tmp_variable = MatlabServer_variables{MatlabServer_tmp_k};
    if (exist(MatlabServer_tmp_variable) ~= 1)
   MatlabServer_lasterr = sprintf('Variable ''%s'' not found.', MatlabServer_tmp_variable);
      fprintf(2, '%s\n', MatlabServer_lasterr);
      MatlabServer_tmp_ok = 0;
      break;
    end;
  MatlabServer_tmp_expr = sprintf('%s, ''%s''', MatlabServer_tmp_expr, MatlabServer_tmp_variable);
  end;

  MatlabServer_tmp_expr = sprintf('%s)', MatlabServer_tmp_expr);
  if (~MatlabServer_tmp_ok)
    writeInt(MatlabServer_os, -1);
    writeUTF(MatlabServer_os, MatlabServer_lasterr);
  else
    fprintf(2, '%s\n', MatlabServer_tmp_expr);
    eval(MatlabServer_tmp_expr);
    writeInt(MatlabServer_os, 0); % Here anything but -1 means "success"
    writeUTF(MatlabServer_os, MatlabServer_tmp_tmpname);
  end

  MatlabServer_tmp_answer = readByte(MatlabServer_is);
  fprintf(2, 'answer=%d\n', MatlabServer_tmp_answer);

  MatlabServer_state = 0;
 clear MatlabServer_tmp_name MatlabServer_tmp_expr MatlabServer_tmp_ok MatlabServer_tmp_answer;
```

```
 %-------------------
 % 'send-remote'
 %-------------------
elseif (MatlabServer_state == strmatch('send-remote', MatlabServer_commands, 'exact'))
  MatlabServer_tmp_tmpname = sprintf('%s_%d.mat', tempname, MatlabServer_port);
  MatlabServer_tmp_expr = sprintf('save(MatlabServer_tmp_tmpname, ''%s''', MatlabServer_saveOption);
   MatlabServer_tmp_ok = 1;
   for MatlabServer_tmp_k=1:length(MatlabServer_variables),
     MatlabServer_tmp_variable = MatlabServer_variables{MatlabServer_tmp_k};
     if (exist(MatlabServer_tmp_variable) ~= 1)
      MatlabServer_lasterr = sprintf('Variable ''%s'' not found.', MatlabServer_tmp_variable);
        fprintf(2, '%s\n', MatlabServer_lasterr);
        MatlabServer_tmp_ok = 0;
        break;
     end;
   MatlabServer_tmp_expr = sprintf('%s, ''%s''', MatlabServer_tmp_expr, MatlabServer_tmp_variable);
   end;
   clear MatlabServer_tmp_k MatlabServer_tmp_variable;

   MatlabServer_tmp_expr = sprintf('%s)', MatlabServer_tmp_expr);
   if (~MatlabServer_tmp_ok)
     writeInt(MatlabServer_os, -1);
     writeUTF(MatlabServer_os, MatlabServer_lasterr);
   else
     fprintf(2, '%s\n', MatlabServer_tmp_expr);
     eval(MatlabServer_tmp_expr);
     MatlabServer_tmp_file = java.io.File(MatlabServer_tmp_tmpname);
     MatlabServer_tmp_maxLength = length(MatlabServer_tmp_file);
     clear MatlabServer_tmp_file;
   writeInt(MatlabServer_os, MatlabServer_tmp_maxLength); % Here anything but -1 means "success"
     fprintf(2, 'Send int: %d (maxLength)\n', MatlabServer_tmp_maxLength);
     MatlabServer_tmp_fid = fopen(MatlabServer_tmp_tmpname, 'r');
     MatlabServer_tmp_count = 1;
     while (MatlabServer_tmp_count ~= 0)
    [MatlabServer_tmp_bfr, MatlabServer_tmp_count] = fread(MatlabServer_tmp_fid, 65536, 'int8');
       if (MatlabServer_tmp_count > 0)
         write(MatlabServer_os, MatlabServer_tmp_bfr);
       end;
     end;
     fclose(MatlabServer_tmp_fid);
     fprintf(2, 'Send buffer: %d bytes.\n', MatlabServer_tmp_maxLength);
     delete(MatlabServer_tmp_tmpname);
   clear MatlabServer_tmp_bfr MatlabServer_tmp_count MatlabServer_tmp_maxLength MatlabServer_tmp_fi
   end
   flush(MatlabServer_os);

   MatlabServer_tmp_answer = readByte(MatlabServer_is);
```

```
    fprintf(2, 'answer=%d\n', MatlabServer_tmp_answer);

    MatlabServer_state = 0;
   clear MatlabServer_tmp_name MatlabServer_tmp_expr MatlabServer_tmp_ok MatlabServer_tmp_answer;

  %------------------
  % 'receive-remote'
  %------------------
 elseif (MatlabServer_state == strmatch('receive-remote', MatlabServer_commands, 'exact'))
    MatlabServer_tmp_len = readInt(MatlabServer_is);
   fprintf(2, 'Will read MAT file structure of length: %d bytes.\n', MatlabServer_tmp_len);

    MatlabServer_tmp_reader = InputStreamByteWrapper(4096);
    MatlabServer_tmp_bfr = [];
    MatlabServer_tmp_count = 1;
    while (MatlabServer_tmp_len > 0 & MatlabServer_tmp_count > 0)
    MatlabServer_tmp_count = MatlabServer_tmp_reader.read(MatlabServer_is, min(4096, MatlabServer_tmp
      if (MatlabServer_tmp_count > 0)
     MatlabServer_tmp_bfr = [MatlabServer_tmp_bfr; MatlabServer_tmp_reader.bfr(1:MatlabServer_tmp_co
        MatlabServer_tmp_len = MatlabServer_tmp_len - MatlabServer_tmp_count;
      end;
    end;

    clear MatlabServer_tmp_reader MatlabServer_tmp_count MatlabServer_tmp_len;

   MatlabServer_tmp_tmpfile = sprintf('%s_%d.mat', tempname, MatlabServer_port);
    MatlabServer_tmp_fh = fopen(MatlabServer_tmp_tmpfile, 'wb');
    fwrite(MatlabServer_tmp_fh, MatlabServer_tmp_bfr, 'int8');
    fclose(MatlabServer_tmp_fh);

    clear MatlabServer_tmp_fh MatlabServer_tmp_bfr;

    load(MatlabServer_tmp_tmpfile);

    delete(MatlabServer_tmp_tmpfile);
    clear MatlabServer_tmp_tmpfile;
    writeByte(MatlabServer_os, 0);

    MatlabServer_state = 0;

  %------------------
  % 'receive'
  %------------------
 elseif (MatlabServer_state == strmatch('receive', MatlabServer_commands, 'exact'))
    MatlabServer_tmp_filename = char(readUTF(MatlabServer_is));
    fprintf(2, 'Will read MAT file: "%s"\n', MatlabServer_tmp_filename);
    load(MatlabServer_tmp_filename);
    clear MatlabServer_tmp_filename;
```

```
    writeByte(MatlabServer_os, 0);
    MatlabServer_state = 0;
    clear MatlabServer_tmp_filename;
  end
end


% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% Shutting down the MATLAB server
% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

fprintf(2, '-----------------------\n');
fprintf(2, 'MATLAB server shutdown!\n');
fprintf(2, '-----------------------\n');
writeByte(MatlabServer_os, 0);
close(MatlabServer_os);
close(MatlabServer_is);
close(MatlabServer_server);
quit;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HISTORY:
% 2016-04-05 [v3.6.0]
% o All verbose/debug messages are outputted to stream #2 ("stderr").
% 2015-09-11 [v3.3.0]
% o Now temporary files use format <tempname>_<port>.mat.
% o Add 'MatlabServer_' prefix to all variables.
% o Add 'evalc' command.  Thanks to Rohan Shah for this.
% 2015-01-08 [v3.1.2]
% o BUG FIX: Matlab$getVariable() for a non-existing variable would
%   crash the R-to-Matlab communication if remote=FALSE.
% 2014-06-23 [v3.0.2]
% o ROBUSTNESS: Variables 'lasterr' and 'variables' are now always
%   defined. Potential bug spotted by Steven Jaffe at Morgan Stanley.
% o Added more progress/verbose output, e.g. current working directory.
% 2014-01-21 [v2.2.0]
% o BUG FIX: The MatlabServer.m script would incorrectly consider
%   Matlab v8 and above as Matlab v6.  Thanks to Frank Stephen at NREL
%   for reporting on this and providing a patch.
% 2013-07-11 [v1.3.5]
% o Updated messages to use 'MATLAB' instead of 'Matlab'.
% 2010-10-25 [v1.3.4]
% o BUG FIX: The MatlabServer.m script incorrectly referred to the
%   InputStreamByteWrapper class as java.io.InputStreamByteWrapper.
%   Thanks Kenvor Cothey at GMO LCC for reporting on this.
% 2010-08-28
% o Now the MatlabServer script reports its version when started.
% 2010-08-27
```

```
% o BUG FIX: Now MatlabServer.m saves variables using the function form,
%   i.e. save().  This solves the problem of having single quotation marks
%   in the pathname. Thanks Michael Q. Fan at NC State University for
%   reporting this problem.
% 2009-08-25
% o BUG FIX: Started to get the error "Undefined function or method
%   'ServerSocket' for input arguments of type 'double'.".  It seems like
%   import java.net.* etc does not work. A workaround is to specify the
%   full path for all Java classes, e.g. java.net.ServerSocket.
%   Thanks Nicolas Stadler for reporting this issue.
% 2006-12-28
% o Extended the accepted range of ports from [1023,49151] to [1023,66535].
% 2006-05-08
% o BUG FIX: The error message string for reporting port out of range
%   was invalid and gave the error '... Line: 109 Column: 45 ")" expected,
%   "identifier" found.'.  Thanks Alexander Nervedi for reporting this.
% 2006-01-21
% o Now an error is thrown if port number is out of (safe) range.
% o Added option to specify the port number via the system environment
%   variable MATLABSERVER_PORT, after request by Wang Yu, Iowa State Univ.
% 2005-03-08
% o BUG FIX: substring() is not recognized by MATLAB v7. Using regexp()
%   which works in MATLAB 6.5 and 7. Workaround eval('try', 'catch').
%   Thanks Patrick Drechsler, University of Wuerzburg for the bug report.
% 2005-02-24
% o Now the dynamic Java classpath is set for MATLAB v7 or higher. This
%   will simplify life for MATLAB v7 users.
% 2005-02-22
% o Added javaaddpath() to include InputStreamByteWrapper.class.
%   Thanks Yichun Wei for feedback and great suggestions.
% 2005-02-11
% o If MATLAB v7 or higher is detected, all MAT structures are saved with
%   option '-V6' so readMat() in R.matlab can read them.
% 2002-09-02 [or maybe a little bit earlier]
% o Created.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

InputStreamByteWrapper.(class|java) script

The Java class InputStreamByteWrapper is needed in order for MATLAB to *receive data* via a data stream. R sends data via a data stream if, and only if, the connection was setup for "remote" communication, that is, with argument `remote = TRUE`).

```
import java.io.*;

/**************************************************************************
```

```
% Compile from within MATLAB with:
% !javac InputStreamByteWrapper.java

% MATLAB example that reads a file using Java code and writes it
% back to a temporary file using MATLAB code. Finally the contents
% of the new file is displayed.

reader = InputStreamByteWrapper;  % Default buffer size is 4096 bytes.

in = java.io.FileInputStream('InputStreamByteWrapper.java');

bfr = [];
len = 1;
while (len > 0)
  len = reader.read(in, 16);  % Read 16 bytes at the time (offset=0).
  if (len > 0)
    bfr = [bfr; reader.bfr(1:len)];  % Add bytes to my MATLAB buffer.
  end
end

close(in);
clear in, reader;

disp(bfr');

tmpfile = tempname;
fh = fopen(tmpfile, 'wb');
fwrite(fh, bfr, 'char');
fclose(fh);

type(tmpfile);
*********************************************************************/
public class InputStreamByteWrapper {
  public static byte[] bfr = null;

  public InputStreamByteWrapper(int capasity) {
    bfr = new byte[capasity];
  }

  public InputStreamByteWrapper() {
    this(4096);
  }

  public int read(InputStream in, int offset, int length) throws IOException {
    return in.read(bfr, offset, length);
  }

  public int read(InputStream in, int length) throws IOException {
```

```
    return read(in, 0, length);
  }

  public int read(InputStream in) throws IOException {
    return in.read(bfr);
  }
}

/**********************************************************************
 HISTORY:
2013-07-11
 o Updated comments to use 'MATLAB' instead of 'Matlab'.
2002-09-02 [or maybe a little bit earlier]
 o Created.
**********************************************************************/
```

`Matlab` *MATLAB client for remote or local MATLAB access*

Description

Package: R.matlab
Class Matlab

```
Object
~~|
~~+--Matlab
```

Directly known subclasses:

public static class **Matlab**
extends Object

Usage

```
Matlab(host="localhost", port=9999, remote=!(host %in% c("localhost", "127.0.0.1")))
```

Arguments

`host`	Name of host to connect to.
`port`	Port number on host to connect to.

remote — If TRUE, all data to and from the MATLAB server will be transferred through the socket connection, otherwise the data will be transferred via a temporary file.

Fields and Methods

Methods:

```
as.character  -
close         -
evaluate      -
finalize      -
getOption     -
getVariable   -
isOpen        -
open          -
readResult    -
setFunction   -
setOption     -
setVariable   -
setVerbose    -
startServer   -
writeCommand  -
```

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Requirements

In order for R to communicate with MATLAB, MATLAB v6 or higher is needed. It will *not* work with previous versions, because they do not support Java.

We use the term *server* to say that MATLAB acts like a server with regard to R. Note that it a standard MATLAB session that runs.

Also, the starting of the MatlabServer is simpler from MATLAB v7, although it is pretty straightforward for MATLAB v6 too. It is easier in MATLAB v7 and above, because the Java class required for remote-data-transfer can be automatically/dynamically added to the MATLAB Java classpath, whereas for MATLAB v6 it has to be added manually (see below).

Remote and non-remote connections

When a remote connection (argument `remote = TRUE`) is used, data is send to and from MATLAB via a data stream. This is needed when R is running on a host with a separated file system than the one MATLAB is running on.

If not connection "remotely" (`remote = FALSE`), data is communicated via the file system, that is, by saving and reading it to temporary MAT files.

Troubleshooting: If "remote" transfers are used, the InputStreamByteWrapper Java class must be found by MATLAB, otherwise an error will occur in MATLAB as soon as data is send from R to MATLAB. In all other cases, the above Java class is *not* needed.

Starting the MATLAB server from within R

The MATLAB server may be started from within R by calling `Matlab$startServer()`. By default 'matlab' is called if named differently set `options(matlab = "matlab6.5")`, say. *The method is experimental and may not work on your system.* By default the MATLAB server listens for connections on port 9999. For other ports, set argument `port`, e.g. `Matlab$startServer(port = 9998)`.

Note that the code will *not* halt and wait for MATLAB to get started. Thus, you have to make sure you will wait long enough for the server to get up and running before the R client try to connect. By default, the client will try once a second for 30 seconds before giving up. Moreover, on non-Windows systems, the above command will start MATLAB in the background making all MATLAB messages be sent to the R output screen. In addition, the method will copy the MatlabServer.m and InputStreamByteWrapper.class files to the current directory and start MATLAB from there.

Starting the MATLAB server without R

If the above does not work, the MATLAB server may be started manually from MATLAB itself. Please follow the below instructions carefully.

To be done once:
In MATLAB, add the path to the directory where MatlabServer.m sits. See `help pathtool` in MATLAB on how to do this. In R you can type `system.file("externals", package = "R.matlab")` to find out the path to MatlabServer.m.

For MATLAB v6 only: Contrary to MATLAB v7 and above, MATLAB v6 cannot find the InputStreamByteWrapper class automatically. Instead, the so called Java classpath has to be set manually. In MATLAB, type `which('classpath.txt')` to find where the default MATLAB classpath.txt file is located. Copy this file to the *current directory*, and append the *path* (the directory) of InputStreamByteWrapper.class to the end of classpath.txt. The path of InputStreamByteWrapper.class can be identified by `system.file("java", package = "R.matlab")` in R.

Lazy alternative: Instead of setting path and classpaths, you may try to copy the MatlabServer.m and InputStreamByteWrapper.class to the current directory from which MATLAB is then started.

To start the server:
In order to start the MATLAB server, type

```
matlab -nodesktop -nosplash -r MatlabServer
```

If using MATLAB v6, make sure your `classpath.txt` is the current directory!

This will start MATLAB and immediately call the MatlabServer(.m) script. Here is how it should look like when the server starts:

```
                    < M A T L A B (R) >
          Copyright 1984-2012 The MathWorks, Inc.
            R2012a (7.14.0.739) 64-bit (glnxa64)
                      February 9, 2012


To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

Running MatlabServer v3.0.2
MATLAB v7.x or higher detected.
Saving with option -V6.
Added InputStreamByteWrapper to dynamic Java CLASSPATH.
----------------------
MATLAB server started!
----------------------
MATLAB working directory: /home/AwesomeUser/FabulousProject/
Trying to open server socket (port 9999)...done.
```

Alternatively you can start MATLAB and type `MatlabServer` at the prompt.

By default the MATLAB server listens for connections on port 9999. For other ports, set environment variable `MATLABSERVER_PORT`.

Confirmed MATLAB versions

This package has been confirmed to work *successfully* out of the box together with the following MATLAB versions: MATLAB v6.1.0.450 (R12.1) [Jun 2001], MATLAB v6.5.0.180913a (R13) [Jul 2002], MATLAB v7.0.0.19901 (R14) [Jun 2004], MATLAB v7.0.1.24704 (R14SP1) [Oct 2004], MATLAB v7.0.4.365 (R14SP2) [Mar 2005], MATLAB v7.2.0.232 (R2006a) [Mar 2006], MATLAB v7.4.0 (R2007a) [Mar 2007]], MATLAB v7.7.0.471 (R2008b) [Oct 2008], MATLAB v7.10.0.499 (R2010a) [Mar 2010], MATLAB v7.11.0.584 (R2010b) [Sep 2010], MATLAB v7.14.0.739 (R2012a) [Mar 2012], MATLAB v8.2.0.701 (R2013b) [Sep 2013], and MATLAB v8.4.0 (R2014b) [Oct 2014]. If you successfully use a different/higher MATLAB version, please tell us, so we can share it here.

It does *not* work with MATLAB v5 or before.

Security

There is *no* security in the communication with the MATLAB server. This means that if you start the MATLAB server, it will wait for requests via the connection at the specified port. As long as your R session has not connected to this port, others may be able to steal the connection and send malicious commands (if they know the R.matlab protocol). The MATLAB server only allows one connection. In other words, if you are connected it is not possible for others to connect to the MATLAB server.

MATLAB server is timing out

It might be that an `*evaluate()` call to the MATLAB server takes a long time for the server to finish resulting in a time-out exception. By default this happens after 30 seconds, but it can be changed by modifying options, cf. `setOption()`.

Multiple parallel MATLAB instances

You can launch multiple parallel MATLAB instance using this interface. This can be done in separate R sessions or in a single one. As long as each MATLAB server/session is communicating on a separate port, there is no limitation in the number of parallel MATLAB instances that can be used. Example:

```
> library('R.matlab')
## Start two separate MATLAB servers
> Matlab$startServer(port = 9997)
> Matlab$startServer(port = 9999)

## Connect to each of them
> matlab1 <- Matlab(port = 9997); open(matlab1)
> matlab2 <- Matlab(port = 9999); open(matlab2)

## Evaluate expression in each of them
> evaluate(matlab1, "x = 1+2; x")
> evaluate(matlab2, "y = 1+2; y")
```

Note that the two MATLAB instance neither communicate nor share variables.

Author(s)

Henrik Bengtsson

See Also

Stand-alone methods `readMat()` and `writeMat()` for reading and writing MAT file structures.

Examples

```
## Not run:
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# This example will try to start the MATLAB server on the local machine,
# and then setup a Matlab object in R for communicating data between R
# and MATLAB and for sending commands from R to MATLAB.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# 1.  Load R.matlab
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
library(R.matlab)
```

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# 2.  Start MATLAB
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# 2.1.  Start MATLAB from R?
# Start MATLAB server on the local machine (if this fails,
# see help(Matlab) for alternatives).
Matlab$startServer()

# 2.2.  OR start MATLAB externally,
#       THEN add 'externals' subdirectory to the MATLAB path

#  (Where is the 'externals' subdirectory?)
print(system.file("externals", package = "R.matlab"))

#       THEN from within MATLAB,
#            issue MATLAB command "MatlabServer"
# Note: If issued from a MATLAB command line, this last command
#       prevents further MATLAB 'command line' input
#       until something like close(matlab) at the end of this script

# 2.3.  If both these options fail, see help(Matlab) for alternatives.


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# 3. Create a MATLAB client object used to communicate with MATLAB
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
matlab <- Matlab()

# 3.1 Check status of MATLAB connection (not yet connected)
print(matlab)

# 3.2 If you experience any problems, ask for detailed outputs
#     by uncommenting the next line
# setVerbose(matlab, -2)

# 3.3 Connect to the MATLAB server.
isOpen <- open(matlab)

# 3.4 Confirm that the MATLAB server is open, and running
if (!isOpen)
  throw("MATLAB server is not running: waited 30 seconds.")

# 3.5 Check status of MATLAB connection (now connected)
print(matlab)


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# 4.  Sample uses of the MATLAB server
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# 4.1 Run MATLAB expressions on the MATLAB server
evaluate(matlab, "A = 1+2;", "B = ones(2, 20);")
```

```
# 4.2 Ask MATLAB to display a value (without transferring it to R)
evaluate(matlab, "A")

# 4.3 Get MATLAB variables
data <- getVariable(matlab, c("A", "B"))
cat("Received variables:\n")
str(data)

# 4.4 Set variables in MATLAB
ABCD <- matrix(rnorm(10000), ncol = 100)
str(ABCD)
setVariable(matlab, ABCD = ABCD)

# 4.5 Retrieve what we just set
data <- getVariable(matlab, "ABCD")
cat("Received variables:\n")
str(data)

# 4.6 Create a function (M-file) on the MATLAB server
setFunction(matlab, "            \
  function [win, aver] = dice(B) \
  %Play the dice game B times    \
  gains = [-1, 2, -3, 4, -5, 6]; \
  plays = unidrnd(6, B, 1);      \
  win = sum(gains(plays));       \
  aver = win/B;                  \
")

# 4.7 Use the MATLAB function just created
evaluate(matlab, "[w, a] = dice(1000);")
res <- getVariable(matlab, c("w", "a"))
print(res)


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# 5. Exception handling
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# 5.1 Try to get non-existing MATLAB variable
#     (will result in an informative error)
tryCatch({
  data <- getVariable(matlab, "unknown")
  cat("Received variables:\n")
  str(data)
}, error = function(ex) {
  print(ex)
})
# Confirm that things still work
data <- getVariable(matlab, "A")
cat("Received variables:\n")
str(data)


# 5.2 Try to evaluate a MATLAB expression that fails
```

```
#     (will result in an informative error)
tryCatch({
  res <- evaluate(matlab, "C = 1+unknown;")
  res
}, error = function(ex) {
  print(ex)
})
# Confirm that things still work
res <- evaluate(matlab, "C = 1+2;")
print(res)
data <- getVariable(matlab, "C")
cat("Received variables:\n")
str(data)


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# 6.  Done:  close the MATLAB client
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# When done, close the MATLAB client, which will also shutdown
# the MATLAB server and the connection to it.
close(matlab)

# 6.1 Check status of MATLAB connection (now disconnected)
print(matlab)

## End(Not run)
```

---

readMat *Reads a MAT file structure from a connection or a file*

Description

Reads a MAT file structure from a connection or a file. Both the MAT version 4 and MAT version 5 file formats are supported. The implementation is based on [1-5]. Note: Do not mix up version numbers for the MATLAB software and the MATLAB file formats.

Usage

```
## Default S3 method:
readMat(con, maxLength=NULL, fixNames=TRUE, drop=c("singletonLists"),
  sparseMatrixClass=c("Matrix", "SparseM", "matrix"), verbose=FALSE, ...)
```

Arguments

con	Binary connection from which the MAT file structure should be read. If a character string, it is interpreted as filename, which then will be opened (and closed afterwards). If a raw vector, it will be read via as a raw binary connection.
maxLength	The maximum number of bytes to be read from the input stream, which should be equal to the length of the MAT file structure. If NULL, data will be read until End Of File has been reached.

fixNames — If TRUE, underscores within names of MATLAB variables and fields are converted to periods.

drop — A character vector specifying cases when one or more dimensions of elements should be dropped in order to decrease the amount of "nestedness" of the returned data structure. This only applies to the MAT v5 file format.

sparseMatrixClass — If "matrix", a sparse matrix is expanded to a regular matrix. If either "Matrix" (default) or "SparseM", the sparse matrix representation by the package of the same name will be used. These packages are only loaded if the a sparse matrix is read.

verbose — Either a logical, a numeric, or a Verbose object specifying how much verbose/debug information is written to standard output. If a Verbose object, how detailed the information is is specified by the threshold level of the object. If a numeric, the value is used to set the threshold of a new Verbose object. If TRUE, the threshold is set to -1 (minimal). If FALSE, no output is written.

... — Not used.

Value

Returns a named list structure containing all variables in the MAT file structure.

Speed performance

This function uses a MAT file parser implemented completely using pure R. For MAT files containing large vectorized objects, for instance long vectors and large matrices, the R implementation is indeed fast enough because it can read and parse each such objects in one go.

On the other hand, for MAT files containing a large number of small objects, e.g. a large number of cell structures, there will be a significant slowdown, because each of the small objects has to be parsed individually. In such cases, if possible, try to (re)save the data in MATLAB using larger ("more vectorized") objects.

MAT cell structures

For the MAT v5 format, *cell* structures are read into R as a list structure.

Unicode strings

Recent versions of MATLAB store some strings using Unicode encodings (miUTF8, miUTF16, miUTF32), which are read as UTF-8. UTF-16 surrogate pairs (Unicode code points above 0xFFFF) are not reassembled.

Reading compressed MAT files

From MATLAB v7, *compressed* MAT version 5 files are used by default [3-5], which is supported by this function.

If for some reason it fails, use save -V6 in MATLAB to write non-compressed MAT v5 files (sic!).

About MAT files saved in MATLAB using '-v7.3'

MAT v7.3 files, saved using for instance `save('foo.mat', '-v7.3')`, stores the data in the Hierarchical Data Format (HDF5) [6, 7], which is a format not supported by this function/package. However, there exist other R packages that can parse HDF5, e.g. CRAN package **h5** and Bioconductor package **rhdf5**.

Reading MAT file structures input streams

Reads a MAT file structure from an input stream, either until End of File is detected or until `maxLength` bytes has been read. Using `maxLength` it is possible to read MAT file structure over socket connections and other non-terminating input streams. In such cases the `maxLength` has to be communicated before sending the actual MAT file structure.

Author(s)

Henrik Bengtsson. The internal MAT v4 reader was written by Andy Jacobson (Princeton University). Support for reading sparse matrices, UTF-encoded strings and compressed files, was contributed by Jason Riedy (UC Berkeley).

References

[1] The MathWorks Inc., *MATLAB - MAT-File Format, version 5*, June 1999.
[2] The MathWorks Inc., *MATLAB - Application Program Interface Guide, version 5*, 1998.
[3] The MathWorks Inc., *MATLAB - MAT-File Format, version 7*, September 2009.
[4] The MathWorks Inc., *MATLAB - MAT-File Format, version R2012a*, September 2012.
[5] The MathWorks Inc., *MATLAB - MAT-File Format, version R2015b*, September 2015.
[6] The MathWorks Inc., *MATLAB - MAT-File Versions*, December 2015. https://www.mathworks.com/help/matlab/import_export/mat-file-versions.html
[7] Undocumented Matlab, *Improving save performance*, May 2013. https://undocumentedmatlab.com/articles/improving-save-performance/
[8] J. Gilbert et al., Sparse Matrices in MATLAB: Design and Implementation, SIAM J. Matrix Anal. Appl., 1992. https://www.mathworks.com/help/pdf_doc/otherdocs/simax.pdf
[9] J. Burkardt, *HB Files: Harwell Boeing Sparse Matrix File Format*, Apr 2010. https://people.sc.fsu.edu/~jburkardt/data/hb/hb.html

See Also

`writeMat`().

Examples

```
path <- system.file("mat-files", package = "R.matlab")
pathname <- file.path(path, "ABC.mat")
data <- readMat(pathname)
print(data)
```

writeMat *Writes a MAT file structure*

Description

This function takes the given variables (...) and places them in a MAT file structure, which is then written to a binary connection.

Usage

```
## Default S3 method:
writeMat(con, ..., fixNames=TRUE, matVersion="5", onWrite=NULL, verbose=FALSE)
```

Arguments

con	Binary connection to which the MAT file structure should be written to. A string is interpreted as filename, which then will be opened (and closed afterwards).
...	*Named* variables to be written where the names must be unique.
fixNames	If TRUE, periods within names of R variables and fields are converted to underscores.
matVersion	A character string specifying what MAT file format version to be written to the connection. If "5", a MAT v5 file structure is written. No other formats are currently supported.
onWrite	Function to be called just before starting to write to connection. Since the MAT file structure does not contain information about the total size of the structure this argument makes it possible to first write the structure size (in bytes) to the connection.
verbose	Either a logical, a numeric, or a Verbose object specifying how much verbose/debug information is written to standard output. If a Verbose object, how detailed the information is is specified by the threshold level of the object. If a numeric, the value is used to set the threshold of a new Verbose object. If TRUE, the threshold is set to -1 (minimal). If FALSE, no output is written (and neither is the R.utils package required).

Note that ... must *not* contain variables with names equal to the arguments matVersion and onWrite, which were chosen because we believe they are quite unique to this write method.

Value

Returns (invisibly) the number of bytes written. Any bytes written by any onWrite function are *not* included in this count.

Limitations

Currently only the uncompressed MAT version 5 file format [6] is supported, that is, compressed MAT files cannot be written (only read).

Moreover, the maximum variable size supported by the MAT version 5 file format is 2^31 bytes [6]. In R, this limitation translates to 2^31-1 bytes, which corresponds to for instance an integer object with 536870912 elements or double object with 268435456 elements.

Details on onWrite()

If specified, the `onWrite()` function is called before the data is written to the connection. This function must take a `list` argument as the first argument. This will hold the element `con` which is the opened `connection` to be written to. It will also hold the element `length`, which specified the number of bytes to be written. See example for an illustration.

Note, in order to provide the number of bytes before actually writing the data, a two-pass procedure has to be taken, where the first pass is imitating a complete writing without writing anything to the connection but only counting the total number of bytes. Then in the second pass, after calling `onWrite()`, the data is written.

Author(s)

Henrik Bengtsson

References

[1] The MathWorks Inc., *MATLAB - MAT-File Format, version 5*, June 1999.
[2] The MathWorks Inc., *MATLAB - Application Program Interface Guide, version 5*, 1998.
[3] The MathWorks Inc., *MATLAB - MAT-File Format, version 7*, September 2009.
[4] The MathWorks Inc., *MATLAB - MAT-File Format, version R2012a*, September 2012.
[5] The MathWorks Inc., *MATLAB - MAT-File Format, version R2015b*, September 2015.
[6] The MathWorks Inc., *MATLAB - MAT-File Versions*, December 2015. https://www.mathworks.com/help/matlab/import_export/mat-file-versions.html

See Also

`readMat()`.

Examples

```
A <- matrix(1:27, ncol = 3)
B <- as.matrix(1:10)
C <- array(1:18, dim = c(2, 3, 3))

filename <- paste(tempfile(), ".mat", sep = "")

writeMat(filename, A = A, B = B, C = C)
data <- readMat(filename)
str(data)
```

```
X <- list(A = A, B = B, C = C)
stopifnot(all.equal(X, data[names(X)]))

unlink(filename)


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# All objects written must be named uniquely
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
tryCatch({
  # Named
  writeMat(filename, A = A)
  # Not named
  writeMat(filename, A)
}, error = function(ex) {
  cat("ERROR:", ex$message, "\n")
})


tryCatch({
  # Uniquely named
  writeMat(filename, A = A, B = B, C = C)
  # Not uniquely named
  writeMat(filename, A = A, B = B, A = C)
}, error = function(ex) {
  cat("ERROR:", ex$message, "\n")
})


## Not run:
# When writing to a stream connection the receiver needs to know on
# beforehand how many bytes are available. This can be done by using
# the 'onWrite' argument.
onWrite <- function(x)
  writeBin(x$length, con = x$con, size = 4, endian = "little")
  writeMat(con, A = A, B = B, onWrite = onWrite)

## End(Not run)
```

Index

∗ **IO**
    readMat, 23
    writeMat, 26
∗ **classes**
    Matlab, 16
∗ **documentation**
    1. The MATLAB server running in MATLAB, 5
∗ **file**
    readMat, 23
    writeMat, 26
∗ **package**
    R.matlab-package, 2
*evaluate, *20*
1. The MATLAB server running in MATLAB, 5

character, *23*, *24*, *26*
connection, *17*, *23*, *26*, *27*

FALSE, *24*, *26*

list, *24*, *27*
logical, *24*, *26*

Matlab, *2*, *3*, 16
matrix, *24*

numeric, *24*, *26*

Object, *16*

R.matlab *(*R.matlab-package*)*, 2
R.matlab-package, 2
R.utils, *2*, *26*
raw, *23*
readMat, *3*, *20*, 23, *27*

setOption, *20*

TRUE, *17*, *24*, *26*

vector, *23*, *24*
Verbose, *24*, *26*

writeMat, *3*, *20*, *25*, 26