

Package ‘R.matlab’

February 14, 2012

Version 1.5.2

Date 2011-11-02

Title Read and write of MAT files together with R-to-Matlab connectivity

Author Henrik Bengtsson <henrikb@braju.com>, Jason Riedy <ejr@cs.berkeley.edu>

Maintainer Henrik Bengtsson <henrikb@braju.com>

Depends R (>= 2.5.0), R.oo (>= 1.8.3)

Suggests R.utils (>= 1.8.7), Matrix, SparseM, Rcompression

Description This package provides methods to read and write MAT files.
It also makes it possible to communicate (evaluate code, send
and retrieve objects etc.) with Matlab v6 or higher running locally or on a remote host.

License LGPL (>= 2.1)

URL <http://www.braju.com/R/>

LazyLoad TRUE

Repository CRAN

Date/Publication 2011-11-03 06:50:22

R topics documented:

R.matlab-package	2
1. The Matlab server running in Matlab	5
Matlab	14
readMat	20
writeMat	24

Index	27
--------------	-----------

R.matlab-package

Package R.matlab

Description

This package provides methods to read and write MAT files. It also makes it possible to communicate (evaluate code, send and retrieve objects etc.) with Matlab v6 or higher running locally or on a remote host.

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.00** 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 the "devel" version, see <http://www.braju.com/R/>.

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

A related initiative is *RMatlab* by Duncan Temple Lang and Omegahat. It provides a bi-directional interface between the R and Matlab languages. For more details, see <http://www.omegahat.org/RMatlab/>. To call R from Matlab on Windows (only), see *MATLAB R-link* by Robert Henson available at the Matlab Central File Exchange (<http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=5051>).

How to cite this package

Whenever using this package, please cite [2] as

No citation information available.

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 inbetween.

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!

Acknowledgements

Thanks to the following people who contributed with valuable feedback, suggestions, code etc.:

- Jason Riedy, Computer Science Division, University of California, Berkeley.
- Patrick Drechsler, Biocenter, University of Wuerzburg.
- Andy Jacobson, Atmospheric and Oceanic Sciences Program, Princeton University.
- Chris Sims, Department of Economics, Princeton University.
- Yichun Wei, Department of Biological Sciences, University of Southern California.
- Spencer Graves.
- 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 licence (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. <http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/>
- [2] Henrik Bengtsson, *R.matlab - Local and remote Matlab connectivity in R*, Mathematical Statistics, Centre for Mathematical Sciences, Lund University, Sweden, 2005. (manuscript in progress).

Author(s)

Henrik Bengtsson (<http://www.braju.com/R/>)

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 <http://www.mathworks.com/support/solutions/data/1-16B8X.html> 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, 2002-2010
%
% 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
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Running MatlabServer v1.3.4');

```

```

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

% -----
% Matlab version-dependent setup
% -----
isVersion7 = eval('length(regexpi(version, ''^7'')) ~= 0', '0');
if (~isVersion7)
    disp('Matlab v6.x detected.');
```

% Default save option
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
    disp('Matlab v7.x or higher detected.');
```

% Matlab v7 saves compressed files, which is not recognized by
% R.matlab's readMat(); force saving in old format.
saveOption = '-V6';
disp('Saving with option -V6.');

% In Matlab v7 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'))});
disp('Added InputStreamByteWrapper to dynamic Java CLASSPATH.');

```

end

% -----
% 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('server'))
    close(server);

```

```

    clear server;
end

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

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

fprintf(1, '-----\n');
fprintf(1, 'Matlab server started!\n');
fprintf(1, '-----\n');

% -----
% Initiate server socket to which clients may connect
% -----
port = getenv('MATLABSERVER_PORT');
if (length(port) > 0)
    port = str2num(port);
else
    % Try to open a server socket on port 9999
    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 (port < 1023 | port > 65535)
    error('Cannot not open connection. Port ('MATLABSERVER_PORT') is out of range [1023,65535]: %d', port);
end

fprintf(1, 'Trying to open server socket (port %d)...', port);
server = java.net.ServerSocket(port);
fprintf(1, '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
clientSocket = accept(server);

fprintf(1, 'Connected to client.\n');

% ...client connected.
is = java.io.DataInputStream(getInputStream(clientSocket));
%is = java.io.BufferedReader(InputStreamReader(is0));
os = java.io.DataOutputStream(getOutputStream(clientSocket));

% -----
% The Matlab server state machine
% -----
% Commands
commands = {'eval', 'send', 'receive', 'send-remote', 'receive-remote', 'echo'};

% As long as we receive data, echo that data back to the client.
state = 0;
while (state >= 0),
    if (state == 0)
        cmd = readByte(is);
        fprintf(1, 'Received cmd: %d\n', cmd);
        if (cmd < -1 | cmd > length(commands))
            fprintf(1, 'Unknown command code: %d\n', cmd);
        else
            state = cmd;
        end

%-----
% 'eval'
%-----
elseif (state == strmatch('eval', commands, 'exact'))
    bfr = char(readUTF(is));
    fprintf(1, '"eval" string: "%s"\n', bfr);
    try
        eval(bfr);,
        writeByte(os, 0);
        fprintf(1, 'Sent byte: %d\n', 0);
        flush(os);
    catch,
        fprintf(1, 'EvaluationException: %s\n', lasterr);
        writeByte(os, -1);
        fprintf(1, 'Sent byte: %d\n', -1);
        writeUTF(os, lasterr);
        fprintf(1, 'Sent UTF: %s\n', lasterr);

```

```

        flush(os);
    end
    flush(os);
    state = 0;

%-----
% 'send'
%-----
elseif (state == strmatch('send', commands, 'exact'))
    tmpname = sprintf('%s.mat', tmpname);
    expr = sprintf('save(tmpname, ''%s'', saveOption);
    ok = 1;
    for k=1:length(variables),
        variable = variables{k};
        if (exist(variable) ~= 1)
            lasterr = sprintf('Variable ''%s'' not found.', variable);
            ok = 0;
            break;
        end;
        expr = sprintf('%s, ''%s'',', expr, variable);
    end;
    expr = sprintf('%s)', expr);
    if (~ok)
        writeInt(os, -1);
        writeUTF(os, lasterr);
    else
        disp(expr);
        eval(expr);
        writeUTF(os, tmpname);
    end

    answer = readByte(is);
    fprintf('answer=%d\n', answer);

    state = 0;

%-----
% 'send-remote'
%-----
elseif (state == strmatch('send-remote', commands, 'exact'))
    tmpname = sprintf('%s.mat', tmpname);
    expr = sprintf('save(tmpname, ''%s'', saveOption);
    ok = 1;
    for k=1:length(variables),
        variable = variables{k};
        if (exist(variable) ~= 1)
            lasterr = sprintf('Variable ''%s'' not found.', variable);
            ok = 0;

```

```

        break;
    end;
    expr = sprintf('%s, ''%s''', expr, variable);
end;
expr = sprintf('%s)', expr);
if (~ok)
    writeInt(os, -1);
    writeUTF(os, lasterr);
else
    disp(expr);
    eval(expr);
    file = java.io.File(tmpname);
    maxLength = length(file);
    clear file;
    writeInt(os, maxLength);
    fprintf(1, 'Send int: %d (maxLength)\n', maxLength);
    fid = fopen(tmpname, 'r');
    count = 1;
    while (count ~= 0)
        [bfr, count] = fread(fid, 65536, 'int8');
        if (count > 0)
            write(os, bfr);
%           fprintf(1, 'Wrote %d byte(s).\n', length(bfr));
        end;
    end;
    fclose(fid);
%   fprintf(1, 'Wrote!\n');
    fprintf(1, 'Send buffer: %d bytes.\n', maxLength);
    delete(tmpname);
    clear bfr, count, maxLength, fid, tmpname;
end
flush(os);

answer = readByte(is);
fprintf('answer=%d\n', answer);

state = 0;

%-----
% 'receive-remote'
%-----
elseif (state == strmatch('receive-remote', commands, 'exact'))
    len = readInt(is);
    fprintf(1, 'Will read MAT file structure of length: %d bytes.\n', len);

    reader = InputStreamByteWrapper(4096);
    bfr = [];
    count = 1;

```

```

while (len > 0 & count > 0)
    count = reader.read(is, min(4096, len));
    if (count > 0)
        bfr = [bfr; reader.bfr(1:count)];
        len = len - count;
    end;
end;

clear reader count len;

tmpfile = sprintf('%s.mat', tempname);
% tmpfile = 'tmp2.mat';
% disp(bfr');
% disp(tmpfile);
fh = fopen(tmpfile, 'wb');
fwrite(fh, bfr, 'int8');
fclose(fh);

clear fh, bfr;

load(tmpfile);

delete(tmpfile);
clear tmpfile;
writeByte(os, 0);

state = 0;

%-----
% 'receive'
%-----
elseif (state == strmatch('receive', commands, 'exact'))
    filename = char(readUTF(is));
    fprintf(1, 'Will read MAT file: "%s"\n', filename);
    load(filename);
    clear filename;
    writeByte(os, 0);
    state = 0;
end
end

% -----
% Shutting down the Matlab server
% -----

fprintf(1, '-----\n');
fprintf(1, 'Matlab server shutdown!\n');

```

```
fprintf(1, '-----\n');
writeByte(os, 0);
close(os);
close(is);
close(server);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HISTORY:
% 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 it's 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 capacity) {
        bfr = new byte[capacity];
    }

    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:
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	Gets a string describing the current Matlab connection.
close	Closes connection to Matlab server.
evaluate	Evaluates a Matlab expression.
finalize	Finalizes the object if deleted.
getOption	Gets the value of an option.
getVariable	Gets one or several Matlab variables.
isOpen	Checks if connection to the Matlab server is open.
open	Tries to open a connection to the Matlab server.
readResult	Reads results from the Matlab server.
setFunction	Defines a Matlab function.
setOption	Sets the value of an option.
setVariable	Sets one or several Matlab variables.
setVerbose	Sets the verbose level to get more details about the Matlab access.
startServer	Static method which starts a Matlab server.
writeCommand	Writes (sends) a command to the Matlab server.

Methods inherited from [Object](#):

\$, \$<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, gc, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, registerFinalizer, 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 is 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, 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 sent 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 sent 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` and `InputStreamByteWrapper` 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, 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 should be the same as the path of the MatlabServer.m that you identified above.

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 >
    Copyright 1984-2004 The MathWorks, Inc.
    Version 7.0.1.24704 (R14) Service Pack 1
                September 13, 2004

```

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

```

Matlab v7.x or higher detected.
Saving with option -V6.
Added InputStreamByteWrapper to dynamic Java CLASSPATH.
-----
Matlab server started!
-----
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: Matlab v6.1.0.450 (R12.1), Matlab v6.5.0.180913a (R13), Matlab v7.0.0.19901 (R14), Matlab v7.0.1.24704 (R14SP1), Matlab v7.0.4.365 (R14SP2), Matlab v7.2.0.232 (R2006a), Matlab 7.4.0 (R2007a), Matlab 7.7.0.471 (R2008b), Matlab version 7.10.0.499 (R2010a), and Matlab version 7.11.0.584 (R2010b). 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()`.

Author(s)

Henrik Bengtsson (<http://www.braju.com/R/>)

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. 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)

# 3.5 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 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.

Both the MAT version 4 and MAT version 5 file formats are supported. The implementation is based on [1].

From Matlab v7, *compressed* MAT version 5 files are used by default [3]. This function supports reading such files, if running R v2.10.0 or newer. For older versions of R, the **Rcompression** package is used. To install that package, please see instructions at <http://www.omegahat.org/cranRepository.html>. As a last resort, use `save -v6` in Matlab to write MAT files that are compatible with Matlab v6, that is, to write non-compressed MAT version 5 files.

Note: Do not mix up version numbers for the Matlab software and the Matlab file formats.

Recent versions of Matlab store some strings using Unicode encodings. If the R installation supports [iconv](#), these strings will be read correctly. Otherwise non-ASCII codes are converted to NA. Saving to an earlier file format version may avoid this problem as well.

Usage

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

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).
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.
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).
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.
...	Not used.

Details

For the MAT v5 format, *cell* structures are read into R as a [list](#) structure.

Value

Returns a named [list](#) structure containing all variables in the MAT file structure.

Author(s)

Henrik Bengtsson, Mathematical Statistics, Lund University. The internal MAT v4 reader was written by Andy Jacobson at Program in Atmospheric and Oceanic Sciences, Princeton University. Support for reading compressed files via **Rcompression**, sparse matrices and UTF-encoded strings was added 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, http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matfile_format.pdf

See Also

`writeMat()`.

Examples

```
path <- system.file("mat-files", package="R.matlab")

# -----
# Reading all example files
# -----
for (version in 4:5) {
  cat("Loading all MAT v", version, " example files in ",
      path, "...\\n", sep="")

  pattern <- sprintf("-v%d[.]mat$", version)
  pathnames <- list.files(pattern=pattern, path=path, full.names=TRUE)

  for (pathname in pathnames) {
    cat("Reading MAT file: ", basename(pathname), "\\n", sep="")
    mat <- readMat(pathname)
    if (interactive()) {
      cat("Press ENTER to view data:")
      readline()
    }
    print(mat)
  }
}

# -----
# Assert that signed and unsigned integers are read correctly
# -----
bs <- readMat(file.path(path, "unsignedByte.mat"))
if (!identical(as.vector(bs$A), as.double(126:255)))
  stop("Error reading unsigned bytes saved by Matlab.")

is <- readMat(file.path(path, "unsignedInt.mat"))
if (!identical(as.vector(is$B), as.double(127:256)))
  stop("Error reading unsigned ints saved by Matlab.")

# -----
# Assert that sparse matrices are read identically in MAT v4 and v5
# -----
mat4 <- readMat(file.path(path, "SparseMatrix3-v4.mat"))
```

```

mat5 <- readMat(file.path(path, "SparseMatrix3-v5.mat"))
diff <- sum(abs(mat4$sparseM - mat5$sparseM))
if (diff > .Machine$double.eps)
  stop("Failed to read identical MAT v4 and MAT v5 sparse matrices.")

# -----
# Assert that sparse matrices can be read as 'Matrix' and 'SparseM'
# -----
pathname <- file.path(path, "SparseMatrix3-v4.mat")
mat4a <- readMat(pathname, sparseMatrixClass="matrix")

if (require("Matrix")) {
  mat4b <- readMat(pathname, sparseMatrixClass="Matrix")
  diff <- sum(abs(as.matrix(mat4b$sparseM) - mat4a$sparseM))
  if (diff > .Machine$double.eps)
    stop("Failed to read MAT v4 sparse matrix by class 'Matrix'.")
}

if (require("SparseM")) {
  mat4c <- readMat(pathname, sparseMatrixClass="SparseM");
  diff <- sum(abs(as.matrix(mat4c$sparseM) - mat4a$sparseM))
  if (diff > .Machine$double.eps)
    stop("Failed to read MAT v4 sparse matrix by class 'SparseM'.")
}

# -----
# Assert that compressed files can be read
# -----
hasMemDecompress <- (getRversion() >= "2.10.0" && exists("memDecompress", mode="function"));
if (hasMemDecompress || require("Rcompression")) {
  # A particular compressed file
  pathname <- file.path(path, "StructWithSparseMatrix-v4,compressed.mat")
  mat4 <- readMat(pathname, sparseMatrixClass="matrix")

  # All compressed files
  pattern <- ",compressed[.]mat$"
  pathnames <- list.files(pattern=pattern, path=path, full.names=TRUE)
  for (pathname in pathnames) {
    cat("Reading MAT file: ", basename(pathname), "\n", sep="")
    mat <- readMat(pathname)
    if (interactive()) {
      cat("Press ENTER to view data:")
      readline()
    }
    print(mat)
  }
}

# -----
# Example of a Matlab struct

```

```

# -----
# File was created by
# s = struct('type',{'big','little'}, 'color','red', 'x',{3,4})
# 1x2 struct array with fields:
#     type
#     color
#     x
# save structLooped.mat s -v6
mat <- readMat(file.path(path, "structLooped.mat"))

# Extract the structure
s <- mat$s

# Field names are always in the first dimension
fields <- dimnames(s)[[1]]
cat("Field names: ", paste(fields, collapse=" "), "\n", sep="");

print(s)

# Get field 'type'
print(s["type",,])

# Get substructure s(:,2)
print(s[:,2])

# -----
# Example of verbose output
# -----
bs <- readMat(file.path(path, "unsignedByte.mat"), verbose=TRUE)

```

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.

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

Usage

```

## Default S3 method:
writeMat(con, ..., 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).
...	<i>Named</i> variables to be written where the names must be unique.
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 <i>not</i> 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.

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 (<http://www.braju.com/R/>)

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:", getMessage(ex), "\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:", getMessage(ex), "\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

*Topic **IO**

readMat, [20](#)
writeMat, [24](#)

*Topic **classes**

Matlab, [14](#)

*Topic **documentation**

1. The Matlab server running in
Matlab, [5](#)

*Topic **file**

readMat, [20](#)
writeMat, [24](#)

*Topic **package**

R.matlab-package, [2](#)

*evaluate, [18](#)

1. The Matlab server running in
Matlab, [5](#)

as.character, [15](#)

character, [25](#)

close, [15](#)

connection, [15](#), [21](#), [25](#)

evaluate, [15](#)

FALSE, [21](#), [25](#)

finalize, [15](#)

getOption, [15](#)

getVariable, [15](#)

iconv, [21](#)

isOpen, [15](#)

list, [21](#), [25](#)

logical, [21](#), [25](#)

Matlab, [2](#), [14](#)

matrix, [21](#)

numeric, [21](#), [25](#)

Object, [14](#), [15](#)

open, [15](#)

R.matlab (R.matlab-package), [2](#)

R.matlab-package, [2](#)

R.utils, [2](#), [21](#), [25](#)

readMat, [2](#), [18](#), [20](#), [25](#)

readResult, [15](#)

setFunction, [15](#)

setOption, [15](#), [18](#)

setVariable, [15](#)

setVerbose, [15](#)

startServer, [15](#)

TRUE, [15](#), [21](#), [25](#)

Verbose, [21](#), [25](#)

writeCommand, [15](#)

writeMat, [2](#), [18](#), [22](#), [24](#)