Demo hoqc_LOG

Han Oostdijk (han@hanoostdijk.nl)

June 7, 2016

Contents

Introduction

This document shows how to use the hoqc_LOG class. With this class the programmer can write messages to a separate logfile or to standard output (the screen).

Features

- choice between separate logfile or standard output.
- the logfile can be reset (emptied) before the first write if that is desired.
- the logfile can be closed after every write if that is desired. Useful for long running jobs.
- a timestamp can be included in the message if that is desired.
- various message types can be distinguished and (temporarily) shown or hidden.
- the message can be formatted from a template with variable parts.

These options are set when the log object is created (with the exception of the settings of the message types).

Constructor arguments

- Argument 1 of the constructor gives the name of the logfile. The default '' indicates the screen.
- the reset (default false) argument indicates that the logfile will be reset (emptied) before the first write.
- the cl_aw (default false) argument indicates that the logfile will be closed after every write. Useful for long running jobs
- the ts (default true) argument indicates that a timestamp will be included in the message

Examples logging to screen

In this section we will write log messages to the screen. Because we did not specify ts we will use its default (true) and therefore a timestamp is included. By default the standard types (info, debug and error) are shown, but as an example we will now hide the INFO messages. Because the XXX type is non-standard it is by default not shown and therefore only the DEBUG message is shown in this section:

```
mylog  = hoqc_LOG('') ;
mylog.set_types('info',false);
mylog.write('info', ...
        'function %s called with input %7.3f','f1',exp(1));
mylog.write('xxx', ...
        'function %s called with input %7.3f','f2',exp(2));
mylog.write('debug', ...
        'function %s called with input %7.3f','f3',exp(3));
```

```
DEBUG  2016-06-07 23:14:25.656 - function f3 called with input  20.086
```

Only after using the set_types function for INFO and XXX messages of these types are show in the log:

```
mylog.set_types('info',true,'xxx',true);
mylog.write('info', ...
        'function %s called with input %7.3f','f4',exp(4));
mylog.write('xxx', ...
        'function %s called with input %7.3f','f5',exp(5));
```

```
INFO   2016-06-07 23:14:25.672 - function f4 called with input  54.598
XXX    2016-06-07 23:14:25.672 - function f5 called with input 148.413
```

An example of logging without a timestamp and a preformatted message.

```
mylog  = hoqc_LOG('','ts',false) ;
mymsg  = 'function xxx is not called';
mylog.write('info', mymsg);
```

```
INFO    - function xxx is not called
```

Examples logging to external log file

All possibilities of logging to screen are also available for logging to an external file. The only difference is that the first argument of the constructur should contain the name of a file that may or may not exist. In addition to the options already discussed there are two options only applicable for an external log file:

- `reset` : when set to `true` it indicates that the log file will be reset (emptied) before writing the first message. The default is `false` and in that case the messages will be written at the end of (appended to) the log file.
- `cl_aw` : when set to `true` it indicates that the log file will be closed after every write. This is useful for long running jobs. The default is `false` and in that case the log file has to be closed with the `closefile` function to be able to read the messages in the file.

In the first example we use the default options for an external log file by specifying only the filename. Starting with an empty file and looking at the log file just before the 'second close' we will see just 'line 1'. Just after the 'second close' we will see 'line 1' and 'line 2'.

Viewing the log file after the 'third close' we will see only 'line 3': because of the `reset` argument after opening the file it was reset (emptied) before writing 'line 3'. By opening the log file with the `cl_aw` argument the file is closed after each write and therefore we see 'line 4' (and 'line 3') already before 'fourth close': the file was closed immediately after writing the message.

```
mylog1 = hoqc_LOG('log.txt') ;
mylog1.write('info', ...
        'line 1');
mylog1.closefile() ; % first close
mylog1 = hoqc_LOG('log.txt') ;
mylog1.write('info', ...
        'line 2');
% view the log file first time: only 'line 1' visible
mylog1.closefile() ; % second close
% view the log file second time: both 'line 1' and 'line 2' visible
mylog1 = hoqc_LOG('log.txt','reset',true) ;
mylog1.write('info', ...
```

```
        'line 3');
mylog1.closefile() ; % third close
% view the log file third time: only 'line 3' visible
mylog1 = hoqc_LOG('log.txt','cl_aw',true) ;
mylog1.write('info', ...
        'line 4');
% view the log file fourth time: both 'line 3' and 'line 4' visible
mylog1.closefile() ; % fourth close
```

Listings

Listing 1: hoqc_LOG.m

```
classdef  hoqc_LOG < handle

    %{
    examples of use:
    mylog = hoqc_LOG('logje.txt','reset',true) ;        % log to file, reset, close after each write
    mylog.set_types('info',false);                      % do not show INFO messages
    mylog.write('info', ...                             % message will not be shown (see prev. line)
        'function %s called input %7.3f','f1',exp(1));
    mylog.write('xxx', ...                              % message will not be shown (unknown type:
        'function %s called input %7.3f','f2',exp(2));  % i.e not one of INFO, DEBUG or ERROR
    mylog.write('debug', ...                            % message will be shown (default for
        'function %s called input %7.3f','f3',exp(3));  % INFO, DEBUG and ERROR is true)
    mylog.set_types('info',true,'xxx',true);            % show INFO and XXX messages
    mylog.write('info', ...                             % message will now be shown
        'function %s called input %7.3f','f4',exp(4));
    mylog.write('xxx', ...                              % message will now be shown
        'function %s called input %7.3f','f5',exp(5));
    mylog.closefile()                                   % close logfile
    %}

    properties (GetAccess = private, Constant = true)
        version = '0.1' ;
    end
```

```
properties (GetAccess = private)
    logfile     = '' ;                          % name logfile (set by constructor)
    logopen     = 0 ;                           % file handle logfile (set by constructor)
    cl_aw       = false ;                       % close after write? (set by constructor)
    ts          = true ;                        % timestamp to insert? (set by constructor)
    msg_types   = {} ;                          % message types (set by set_types)
    msg_tf      = true ;                        % display types? (set by set_types)
end

methods (Access = public)

    function obj = hoqc_LOG(varargin)           % constructor
        p = inputParser;
        addRequired(p,'logfile',@ischar);       % name logfile
        addParameter(p,'reset',false,@islogical) ;  % reset logfile ?
        addParameter(p,'cl_aw',false,@islogical) ;  % close after write?
        addParameter(p,'ts',true,@islogical) ;      % timestamp to insert?
        parse(p,varargin{:})                    % parse inputs
        obj.logfile     = p.Results.logfile ;   % name logfile
        obj.logopen     = 0 ;                   % file handle 0 (not open yet)
        reset           = p.Results.reset ;     % reset logfile ?
        obj.cl_aw       = p.Results.cl_aw ;     % close after write?
        obj.ts          = p.Results.ts ;        % timestamp to insert?
        obj.msg_types   = {'info','debug','error'}; % message types
        obj.msg_tf      = {true, true, true};   % display types ?
        if reset && numel(obj.logfile) > 0
            writefile(obj,'',reset)             % reset (empty) logfile
        end
    end
    function version = getversion(obj)
        version = obj.version;
    end
    function logfile = getlogfile(obj)
        logfile = obj.logfile;
    end
    function set_types(obj,varargin)            % indicate which types will be printed
```

```
        types    = varargin ;
        ntypes   = numel(types);
        if ntypes == 0, return, end
        if mod(numel(types),2) == 1
            error('hoqc_LOG.set_types: not enough arguments')
        end
        for i=1:numel(types)/2
            set_type(obj, ...                          % set set type to false or true
                types{2*i-1},types{2*i});
        end
    end
    function tf = get_type(obj,type)                   % inquire if type will be printed
        type     = lower(type);                        % argument to lower case
        [~,ix]   = ismember(type,obj.msg_types) ;      % lookup
        if ix == 0
            tf   = false ;                             % not found: not to print
        else
            tf   = obj.msg_tf{ix} ;                    % found: return boolean
        end
    end
    function write(obj,type,msg,varargin)              % write message (if type fits)
        if get_type(obj,type) == false
            return;
        end
        if obj.ts == true
            dt   = datestr(now, ...                    % formatted current time
                'yyyy-mm-dd HH:MM:SS.FFF');
        else
            dt   = '' ;                                % no date_time included in message
        end
        msg_out = sprintf('%-7s%s - %s', ...           % combine type, timestamp and message
            upper(type),dt,msg) ;
        if numel(varargin) ~= sum(msg_out=='%')        % arguments correspond with message?
            error(['hoqc_LOG.write: ', ...             % display error if not
                'incorrect number of arguments'])
        else
            msg_out=sprintf(msg_out,varargin{:}) ;     % insert arguments in message
```

```
        end
        if numel(obj.logfile) == 0
            fprintf('%s\n',msg_out);
        else
            writefile(obj,msg_out) ;
        end
    end
    function closefile(obj)
        if obj.logopen > 0
            fclose(obj.logopen);
        end
        obj.logopen = 0;
    end

end

methods (Access = private)

    function set_type(obj,type1,tf1)                % indicate which types will be printed
        [~,ix] = ismember(type1,obj.msg_types) ;
        if ix > 0
            obj.msg_tf{ix} = tf1;                   % type1 found: change boolean
        else
            obj.msg_types = [obj.msg_types, type1]; % add type1 to array
            obj.msg_tf    = [obj.msg_tf, tf1];      % with corresponding boolean
        end
    end
    function writefile(obj,msg,varargin)
        defoptArgs      =  {false} ;                % default values for reset
        optArgs         =  ...                     % merge specified and default values
            hoqc_LOG.setOptArgs(varargin,defoptArgs) ;
        reset           = optArgs{1} ;              % reset file (t) or append (f)
        if reset
            oparm       = 'wt' ;                    % reset
        else
            oparm       = 'at' ;                    % append
        end
```

```
            if obj.logopen == 0
                obj.logopen = fopen(obj.logfile,oparm);
                if obj.logopen == 0
                    error(['hoqc_LOG.write: ', ...      % display error if file could not be opened
                        'file %s could not be opened'], ...
                        obj.logfile)
                end
            end
            if reset == false
                fprintf(obj.logopen,'%s\n',msg);
            end
            if reset || (obj.cl_aw == true)
                closefile(obj);
            end
        end

    end

    methods (Static, Access = private)

        function defArgs = setOptArgs(a,defArgs)
            % set non-specified optional parameters to default values
            % idea from Omid Khanmohamadi on matlabcentral

            % a          : arguments passed with varargin
            % defArgs    : on input default arguments
            %              on output default argument overwritten by the specified ones (if not empty)

            empty_a = cellfun(@(x)isequal(x,[]),a);     % indicate a that are not specified (empty)
            [defArgs{~empty_a}] = a{~empty_a};          % replace defaults by non-empty one
        end

    end

end
```