```
function dynare(fname, varargin)
%       This command runs dynare with specified model file in argument
%       Filename.
%       The name of model file begins with an alphabetic character,
%       and has a filename extension of .mod or .dyn.
%       When extension is omitted, a model file with .mod extension
%       is processed.
%
% INPUTS
%   fname:      file name
%   varargin:   list of arguments following fname
%
% OUTPUTS
%   none
%
% SPECIAL REQUIREMENTS
%   none

% Copyright (C) 2001-2020 Dynare Team
%
% This file is part of Dynare.
%
% Dynare is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% Dynare is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with Dynare.  If not, see <http://www.gnu.org/licenses/>.

if ~nargin || strcmpi(fname,'help')
    skipline()
    disp(['This is Dynare version ' dynare_version() '.'])
    skipline()
    disp('USAGE: dynare FILENAME[.mod,.dyn] [OPTIONS]')
    skipline()
    disp('The dynare command executes instruction included in FILENAME.mod.')
    disp('See the reference manual for the available options.')
    skipline()
    return
end

% The following needs to come early, to avoid spurious warnings (especially under Octave)
warning_config;

% Handle nopathchange option
% Note that it is only handled if it appears on the command-line, and not at
% the top of the .mod file (since the treatment needs to take place very early,
% even before we make the various checks on the filename)
change_path_flag = true;
if nargin>1
    id = ismember(varargin, 'nopathchange');
    if any(id)
        change_path_flag = false;
        varargin(id) = [];
    end
end
check_matlab_path(change_path_flag);

% Detect if MEX files are present; if not, use alternative M-files
dynareroot = dynare_config();

if isoctave
    % The supported_octave_version.m file is not in git nor in the source
    % package, it is manually added in binary packages distributed on dynare.org
    if exist('supported_octave_version', 'file') && ~strcmp(supported_octave_version,↙
version)
        skipline()
        warning(['This version of Octave is not supported. Consider installing ' ...
                 'version %s of Octave\n' ...
                 'from www.octave.org, otherwise m files will be used instead ' ...
                 'of precompiled mex files and some\nfeatures, like solution ' ...
                 'of models approximated at third order, will not be available.'],↙
supported_octave_version())
        skipline()
    elseif octave_ver_less_than('4.2') % Should match the test in↙
mex/build/octave/configure.ac
                                       % and in m4/ax_mexopts.m4
        skipline()
        warning(['This version of Dynare has only been tested on Octave 4.2 and above.↙
Dynare may fail to run or give unexpected result. Consider upgrading your version of↙
Octave.'])
        skipline()
    end
else
    if matlab_ver_less_than('7.9') % Should match the test in mex/build/matlab/configure.↙
ac
        skipline()
        warning('This version of Dynare has only been tested on MATLAB 7.9 (R2009b) and↙
above. Since your MATLAB version is older than that, Dynare may fail to run, or give↙
unexpected results. Consider upgrading your MATLAB installation, or switch to Octave.');
        skipline()
    end
end

% disable output paging (it is on by default on Octave)
more off
```

```
% sets default format for save() command
if isoctave
    save_default_options('-mat')
end

if nargin < 1
    error('Dynare: you must provide the name of the .mod file in argument')
end

if ~ischar(fname)
    error('Dynare: argument of dynare must be a text string')
end

% Testing if filename has more than one period (not allowed).
dot_location=strfind(fname,'.');
if length(dot_location)>1
    error('Dynare: Periods in filenames are only allowed for .mod or .dyn extensions')
end

if dot_location==length(fname)
    error('Dynare: Periods in filenames are only allowed for .mod or .dyn extensions')
end

% Add dyn or mod extension to the file name if not already provided.
if isempty(dot_location)
    fnamelength = length(fname);
    fname1 = [fname '.dyn'];
    d = dir(fname1);
    if length(d) == 0
        fname1 = [fname '.mod'];
    end
    fname = fname1;
else
    % Check provided file extension.
    if ~strcmpi(fname(dot_location+1:end), 'mod') && ~strcmpi(fname(dot_location+1:end),↙
'dyn')
        error('Dynare: argument must be a filename with .mod or .dyn extensions')
    end
    fnamelength = length(fname) - 4;
end

if fnamelength + length('.set_auxiliary_variables') > namelengthmax()
    error('Dynare: the name of your .mod file is too long, please shorten it')
end

% Workaround for a strange bug with Octave: if there is any call to exist(fname)
% before the call to the preprocessor, then Octave will use the old copy of
% the .m instead of the newly generated one. Deleting the .m beforehand
% fixes the problem.
if isoctave && length(dir([fname(1:(end-4)) '.m'])) > 0
```

```
    delete([fname(1:(end-4)) '.m'])
end

if ~isempty(strfind(fname,filesep))
    fprintf('\nIt seems you are trying to call a .mod file not located in the "Current↙
Folder". This is not possible (the %s symbol is not allowed in the name of the .mod↙
file).\n', filesep)
    [pathtomodfile,basename] = fileparts(fname);
    if exist(pathtomodfile,'dir')
        filesindirectory = dir(pathtomodfile);
        filesindirectory = struct2cell(filesindirectory);
        filesindirectory = filesindirectory(1,:);
        if ~isempty(strmatch([basename '.mod'],filesindirectory)) || ~isempty(strmatch↙
([basename '.dyn'],filesindirectory))
            fprintf('Please set your "Current Folder" to the folder where the .mod file↙
is located using the following command:\n')
            fprintf('\n  >> cd %s\n\n',pathtomodfile)
        else
            fprintf('The file %s[.mod,.dyn] could not be located!\n\n',basename)
        end
    end
    error(['Dynare: can''t open ' fname, '.'])
end

if ~exist(fname,'file') || isequal(fname,'dir')
    fprintf('\nThe file %s could not be located in the "Current Folder". Check whether↙
you typed in the correct filename\n',fname)
    fprintf('and whether the file is really located in the "Current Folder".\n')
    try
        list_of_mod_files = ls('*.mod');
        fprintf('\nCurrent folder is %s, and contains the following .mod files:\n\n',pwd)
        disp(list_of_mod_files)
    catch
        fprintf('\nCurrent folder is %s, and does not contain any .mod files.\n\n',pwd)
    end
    error(['Dynare: can''t open ' fname])
end

if ~isvarname(fname(1:end-4))
    error('Dynare: argument of dynare must conform to MATLAB''s convention for naming↙
functions, i.e. start with a letter and not contain special characters. Please rename↙
your .mod file.')
end

% pre-dynare-preprocessor-hook
if exist(fname(1:end-4),'dir') && exist([fname(1:end-4) filesep 'hooks'],'dir') && exist↙
([fname(1:end-4) filesep 'hooks/priorprocessing.m'],'file')
    run([fname(1:end-4) filesep 'hooks/priorprocessing'])
end

% Parse some options, either for the command-line or from the top of the .mod file
```

```
file_opts = parse_options_line(fname);
preprocessoroutput = ~ismember('nopreprocessoroutput', varargin) && ...
                     ~ismember('nopreprocessoroutput', file_opts);
nolog = ismember('nolog', varargin) || ismember('nolog', file_opts);
onlymacro = ismember('onlymacro', varargin) || ismember('onlymacro', file_opts);
onlyjson = ismember('onlyjson', varargin) || ismember('onlyjson', file_opts);

if ispc
    arch = getenv('PROCESSOR_ARCHITECTURE');
else
    [~, arch] = system('uname -m');
end

if isempty(strfind(arch, '64'))
    arch_ext = '32';
    if preprocessoroutput
        disp('Using 32-bit preprocessor');
    end
else
    arch_ext = '64';
    if preprocessoroutput
        disp('Using 64-bit preprocessor');
    end
end

if preprocessoroutput
    fprintf(['Starting Dynare (version ' dynare_version() ').\n']);
    fprintf('Calling Dynare with arguments: ');
    if isempty(varargin)
        disp('none')
    else
        disp(strjoin(varargin, ' '));
    end
end

command = ['"' dynareroot 'preprocessor' arch_ext filesep 'dynare_m" ' fname] ;
command = [ command ' mexext=' mexext ' "matlabroot=' matlabroot '"'];
% Properly quote arguments before passing them to the shell
if ~isempty(varargin)
    varargincopy = varargin;
    % Escape backslashes and double-quotes
    varargincopy = strrep(varargincopy, '\', '\\');
    varargincopy = strrep(varargincopy, '"', '\"');
    if ~ispc
        % On GNU/Linux and macOS, also escape dollars and backquotes
        varargincopy = strrep(varargincopy, '$', '\$');
        varargincopy = strrep(varargincopy, '`', '\`');
    end
    % Finally, enclose arguments within double quotes
    dynare_varargin = ['"' strjoin(varargincopy, '" "') '"'];
    command = [command ' ' dynare_varargin];
```

```
end

% Under Windows, make sure the MEX file is unloaded (in the use_dll case),
% otherwise the preprocessor can't recompile it
if isoctave
    clear([fname(1:end-4) '.static'], [fname(1:end-4) '.dynamic'])
else
    clear(['+' fname(1:end-4) '/static'], ['+' fname(1:end-4) '/dynamic'])
end

[status, result] = system(command);
if status ~= 0 || preprocessoroutput
    disp(result)
end
if onlymacro
    if preprocessoroutput
        disp('Preprocessor stopped after macroprocessing step because of ''onlymacro''↙
option.');
    end
    return
end

if onlyjson
    if preprocessoroutput
        disp('Preprocessor stopped after preprocessing step because of ''onlyjson''↙
option.');
    end
    return;
end

% post-dynare-prerocessor-hook
if exist(fname(1:end-4),'dir') && exist([fname(1:end-4) filesep 'hooks'],'dir') && exist↙
([fname(1:end-4) filesep 'hooks/postprocessing.m'],'file')
    run([fname(1:end-4) filesep 'hooks/postprocessing'])
end

% Save preprocessor result in logfile (if `no_log' option not present)
if ~nolog
    logname = [fname(1:end-4) '.log'];
    fid = fopen(logname, 'w');
    fprintf(fid, '%s', result);
    fclose(fid);
end

if status
    % Should not use "error(result)" since message will be truncated if too long
    error('Dynare: preprocessing failed')
end

if ~ isempty(find(abs(fname) == 46))
    fname = fname(:,1:find(abs(fname) == 46)-1) ;
```

```
end

% We need to clear the driver (and only the driver, because the "clear all"
% within the driver will clean the rest)
clear(['+' fname '/driver'])

evalin('base',[fname '.driver']) ;

end

% Looks for an options list in the first non-empty line of the .mod file
% Should be kept in sync with the function of the same name in↙
preprocessor/src/DynareMain.cc
%
% Note that separating options with commas is accepted, but is deprecated (and↙
undocumented)
%
% Also, the parser does not handle correctly some corner cases: for example, it
% will fail on something like -Dfoo="a b,c" (will split at whitespace and comma)
function opts = parse_options_line(fname)
    opts = {};
    fid = fopen(fname, 'r');
    while true
        firstline = fgetl(fid);
        if firstline == -1
            fclose(fid);
            return
        end
        if ~isempty(firstline)
            break
        end
    end
    fclose(fid);
    t = regexp(firstline, '^\s*//\s*--\+\s*options:([^\+]*)\+--', 'tokens');
    if isempty(t)
        return
    end

    opts = regexp(t{1}{1}, '[^,\s]+', 'match');

    if ismember(opts, 'nopathchange')
        warning('The ''nopathchange'' option is not taken into account when it appears at↙
the top of ''.mod'' file. You should rather pass it on the command-line.')
    end
end
```