Documentation Workflow

The documentation workflow describes how to document new m-file scripts or functions and where they must be registrated into the publishing process of the documentation. So the published m-file is available in the Matlab help browser of this project.

1. Create a new m-file in the project structure
2. Use the script or function template for initial edit and fill the template with new content.
3. Make introducing documentation entries. If it is a new module, so introduce the module with its own doc where all scripts, functions and classes are listed. If this document already exist, make a new entry.
4. Make help entry in the helptoc.xml via tocitem tag. List all sections of the doc comment as sub tocitems.
5. Introduce the new file to the publish script and make an entry under a fitting section or make a new one if it is a new module or folder.
6. Introduce the new file to export published files script and do toc entries into script file generate pdf-manual.
7. Commit the done work.

See Also

- Project Structure.
- Display Custom Documentation
- publishProjectFilesToHTML
- exportPublishedToPdf

Script Template

```
%% scriptName
% Detailed description of the script task and summary description of
% underlaying script sections.
%
%
%% Requirements
% * Other m-files required: None
% * Subfunctions: None
% * MAT-files required: None
%
%
%% See Also
% * Reference1
% * Reference2
% * Reference3
%
%
% Created on Month DD. YYYY by Creator. Copyright Creator YYYY.
%
% <html>
% <!--
% Hidden Clutter.
% Edited on Month DD. YYYY by Editor: Single line description.
% -->
% </html>
%
%
%% First Script Section
% Detailed section description of step by step executed script code.
disp("Prompt current step or meaningful information of variables.")
Enter section source code
```

```
%% Second Script Section
% Detailed section description of step by step executed script code.
disp("Prompt current step or meaningful information of variables.")
Enter section source code
```

Function Template

```
%% functionName
% Single line summary.
%
%% Syntax
%   outputArg = functionName(positionalArg)
%   outputArg = functionName(positionalArg, optionalArg)
%
%
%% Description
% *outputArg = functionName(positionalArg)* detailed use case description.
%
% *outputArg = functionName(positionalArg, optionalArg)* detailed use case
% description.
%
%
%% Examples
%   Enter example matlab code for each use case.
%
%
%% Input Argurments
% *positionalArg* argurment description.
%
% *optionalArg* argurment description.
%
%
%% Output Argurments
% *outputArg* argurment description.
%
%
%% Requirements
% * Other m-files required: None
% * Subfunctions: None
% * MAT-files required: None
%
%
%% See Also
% * Reference1
% * Reference2
% * Reference3
%
%
% Created on Month DD. YYYY by Creator. Copyright Creator YYYY.
%
% <html>
% <!--
% Hidden Clutter.
% Edited on Month DD. YYYY by Editor: Single line description.
% -->
% </html>
%
function [outputArg] = functionName(possitionalArg, optionalArg)
    arguments
        % validate possitionalArg: dim class {validator}
        possitionalArg (1,:) double {mustBeNumeric}
        % validate optionalArg: dim class {validator} = defaultValue
        optionalArg (1,:) doubel {mustBeNumeric, mustBeEqualSize(positionalArg, optionalArg)} = 4
    end
    outputArg = positionalArg + optionalArg;
end

% Custom validation function
function mustBeEqualSize(a,b)
    % Test for equal size
    if ~isequal(size(a),size(b))
        eid = 'Size:notEqual';
        msg = 'Size of first input must equal size of second input.';
        throwAsCaller(MException(eid,msg))
    end
end
```