Coding Conventions
Image Description ToolBox
(Version 0.8.2, 25.03.2004)

These coding rules are valid for the development of the Local and Global Image Description ToolBox (IDTB) and are established according to previous *MatLab*-Toolboxes developed by Institute for Computer Graphics and Vision (ICG).

1 Toolbox Conventions

1.1 General Rules

- For every (public) function there is exactly one m-file whose name should be chosen in close context to the associated functionality. According to *MatLab* specific syntax, the name of the m-file and the function's name have to be the same.
- The code must readable for others. Therefore the usage of comments is recommended if the code is not self-explaining or if a complicated algorithm needs some explanation. Unnecessary comments must be avoided because they are considered disturbing.
- The names of variables and methods should be chosen in a very meaningful way (self-explaining), even if the code appears to be more complex.
- The code should be divided into meaningful, clearly visible sections.
  - The very first part must be the standard header. (see 1.2)
  - In the following parts default values should be assigned and input parameter should be checked for vality.
  - The algorithmic section should clearly be separated from the rest.
  - Optionally the output parameters can be assigned in the end of the code.
- Functional programming style should be preferred to imperative programming style (i.e. for-loops should be avoided if possible).
- Develop following the KISS (Keep It Small and Simple) principle.
- For every directory in the toolbox - inlcuding the root directory - there must exist a file Content.m (see A.3) that includes a short list of all files included in the directory as well as some version history and a file Readme.m (see A.4) that gives some information about the toolbox, the autors and a general copyright information.

1.2 Standard Header

- The standard-header (see A.1) has to be included in every m-file.
- Using the standard header, the automatic generation of a documentation system in html-format is supported. Therefore the header MUST NOT BE CHANGED ANYWAY !
- The short description (first line) must not exceed the lenght on ONE LINE. This is for layout reasons in the html-files.
- As the standard header is also displayed as help text in the *MatLab* shell, a delimiter line consisting of % (e.g. 75) should be added at the beginning and the end of the header block. This ensures, that a help text can simple be recognised as a help text, respectively.
- The sections *Warnings*, *Notes*, *References*, *See also* and *Called by* may be leaved out, if there are no warnings or no need for additional notes, if no provided code or methods were used or if the functions are not in context to other ones.
- As code provided by others is used, this must be documentated in the section *Notes* or in the section *References* for copyright reasons.
- The section *Known bugs* must be included, even if there are no known bugs to emphasise the fact that there are not any known bugs. For this case the description - *currently none* - may be used.

1.3 Input/Output Interface

- An error code is the minimal required output of every function (see 1.4).
- Every function must have as well an optional input parameter was well an optional output parameter, even if no optional parameters are needed. This is reasonable because if some parameters have to be added later on, the interface needs not to be modified.
- Optional parameters must be stuctures and should be named *isParamIn* and *isParamOut*. The names of the structures' elements should be meaningfull (symbolic names), prefixes are not compelling necessary (see 2).
- The optional output parameter must be the parameter next to last, followed only by the error code. The optional input parameter must be the last parameter.
- All optional parameters have to be predefined with a suitable default value. These default values may be overwritten in the beginning of the code.

According the parameter conventions described above, a function definiton is given in following the from:

```
[iOutputA, isParamOut, iErrorCode] = ICG_functionName(iInputA, isParamIn)
```

1.4 Error Codes and Warnings

1.4.1 Error Codes

- Every function must return an error code that must be specified in the first comment block (standard header) (see A.2).
- Errorcodes are assigned to positive integers. Errorcode=0 is reserved for the NO ERROR case and has to be returned by all functions even if no other error handling is implemented.
- The error code is the last handled parameter in the function's output arguments list and should be named *iErrorCode*.

1.4.2 Warnings

- Warnings should be used if it is not necessary to stop the execution of MatLab script, but if the user should be informed that something extraordinary happened.
- Warnings are assigned to negative integers.

1.4.3 Additional Recommendations

- Enhanced error handling may be usefull (i.e. for debugging). Therefore each function should itself generate an interactive error message/warning either for the MatLab console (e.g. *error*-command, *warning*-command, *disp*-command, ...) or using an error dialog (popup window).
- For reasons of conistency same errors resulting from different functions should return the same error codes/warnings. Furthermore it would be a good idea to reserve ranges of error codes/warnings for single functions. For instance, if error codes [100...200] are reserved for function DUMMY and an error X is assigned to error code 13 in a underlying function, function DUMMY would return error code 113, if error X occurs in the underlying function.

1.5 Version Number Conventions

The version number conventions of IDTB are inspired by the Linux kernel version numbering conventions. The version number of a ITDB-function follows a <major>.<minor>.<patchlevel> scheme. Even-numbered values of the minor number are deemed stable, while odd values are beta or experimental code. The numbers <major>, <minor> and <patchlevel> consist of one or more digits, where <patchlevel> may be leaved out. A version number of the form 0.y.z describes a prerelease version of a function. Therefore e.g. 0.8, 1.2 and 1.4.9 describe a stable (prerelease) version whereas e.g 0.7, 1.3 and 4.5.1 describe a unstable (prerelease) version.

2 Coding Conventions

- Every m-file starts with the definition of the function followed by a blank line and the standard header (see A.2).
- The language for code is English. This applies for comments as well as for names and identifiers in the code.
- For reasons of readability spaces should be used when writing binary operators (+, -, =, ...).
- The indent-size for code-blocks is 4 spaces[1], tabs must not be used for indentions.
- The length of lines in the code should be restricted to 75[1].
- Variables have to be named in capital letters (optional prefixes for interface-variables have to be taken into account). Exception: Counter variables or indices may be called *i, j, k, ...*
- The following prefixes have to be used for variables:

m	Matrix
v	Vector
i	Interface parameter
c	Strings or characters
s	Structures

  If more than one prefix is used (i.e. interface parameter), the interface prefix *i* must be the first one.
- Because structures are mostly used for input and output parameters the elements of a structure should have self explaining names - prefixes are not compelling necessary.
- Function-names have to start with lower letters. The prefix „ICG_“ has to be used for all public functions, since only .m-files with the prefix „ICG_“ are included in the automatic help-system.
- If the name of a variable or function is a connection of more than one word, subsequent words start with capital letters.
- The order of the parameters in function interfaces should be made with respect to the magnitude of the handled data types. That is, start structures followed by matrices, vectors (strings) and scalars (characters).
- Special characters (e.g. umlauts etc.) must not be used.
- The code should be uniform.
  - Variables that stand for the same should always called the same.

[1] *Default value by MatLab 6.5*

  - Methods that perform the same task should always called the same and take the same parameters.
  - For methods that take the same parameters the order of the parameters should always be the same.

3 Additional Design Patterns

- Integration of sources to a source code control system (CVS, Sourcesafe) should only be done, if the provided method has been proved.
- The design of every function must allow to run it directly from the console without any graphical user interface (GUI).
- As a GUI is required, the GUI should only act as interface between the user and underlying functionality, i.e. no algorithmic functionality should be handled within the GUI.
- Every function should have an own implementation for the visualisation of results (images, showing a result matrix, ...) that can be activated by an optional parameter.
- Every function should have a verbose mode (e.g. for debugging), where the algorithmic progress is indicated and status messages are generated on the console.
- As code provided by others, especially in binary form, is used, it is not necessary and of course not meaningful to adapt the whole code to these coding rules. Instead of that, a wrapper for the included code should be written that follows the coding rules. On the one hand side a non-developer can use a new functionality, because the interfaces are in the familar style. On the other hand side this approach makes it possible to use methods in a very early stage without explicitly understanding the underlying implementation details. In a later step this wrapping functions could be used for verification and evaluation of the own implementation in comparison to the original one.

A Appendix

A.1 Example: Standard Header

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ICG_exampleFunction - short description here (max one line!!)
%
% [iOut1, iOut2, iErrorCode] = ICG_exampleFunction(iIn1, iIn2)
%
% Functionality:
%     enter the detailed description here
%
% Parameters [IN]:
%     iIn1:    description of parameter iIn1
%
% Returns [OUT]:
%     iOut1:    description of parameter iOut1
%
% Error Codes [OUT]:
%     0:    No Error
%
% Warnings [OUT]:
%     -1:    Warning description
%
% Notes:
%     additional, important notes
%
% Known bugs:
%     known bugs should be recorded here
%
% References:
%     basic code or method providers
%
% See also:
%     function1, ... (functions in close context)
%
% Called by:
%      function1, ... (functions that call ICG_exampleFunction)
%
% Authors:            authors
% Created:            date of creation
% Version:            version of script
% Matlab version:     version (Windows), version (Linux), ...
%
% Copyright (c) 2004 ICG, Graz University of Technology, Austria
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

A.2 Example: m.file

This not very meaningful file gives a short overview of the coding convention from above. To place emphasis on the code (e.g. names of variables) parts of the the header are cut.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ICG_add   Returns a diagonal matrix with sin(iA) and sin (iB)
%
% function [imResult, iErrorCode] = ICG_sinExample(iA, iB);
%
% Error Codes [OUT]
%     0:    No Error
%     1:    Wrong number of input parameters
%     2:    At least one input parameter is not a numeric
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialization of variables %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

iErrorCode = 0;
sHeading = 'Diagonal Matrix of Sinus and Cosinus';

for i = 1:2
    for j = 1:2
        imResult(i,j) = 0;
    end
end


%%%%%%%%%%%%%%%%%%%%%%
% main calculation %
%%%%%%%%%%%%%%%%%%%%%%

% test if the input parameters are correct
if (nargin ~= 2)
    iErrorCode = 1;
elseif ((isnumeric(iA) == 0) || (isnumeric(iB) == 0))
    iErrorCode = 2;
else
    % Generation of the diagonal matrix
    vSin = [sin(iA) sin(iB)];
    mDiag = diag(vSin);
end;


%%%%%%%%%%%%%%%%%%%%%%%%%%
% set output parameters %
%%%%%%%%%%%%%%%%%%%%%%%%%%

display(sHeading);
imResult = mDiag;
```

A.3 Example: Content.m

```
% Contents of ICG Utils Toolbox.
% Version 0.1 12.03.2002
%
% Release information.
%     Readme - Display information about current and previous versions.
%
% Administrative Utils.
%     ICG_exampleFunction       Example for an ICG_VisionTools function.
%     ICG_installVisionTools    Installation function for entire toolbox.
%
% Copyright (c) 2004 ICG, Graz University of Technology, Austria
```

A.4 Example: Readme.m

```
% ICG_Utils important information
%
% Toolbox for utility functions not directly related to computer vision.
%
% This toolbox was created to collect and share the work of the members of
% the Institute of Computer Graphics and Vision, Graz University of
% Technology.
%
% Web:               http://www.icg.tu-graz.ac.at
% E-mail address:    {autors}@icg.tu-graz.ac.at
% CVS repository:    fcggsg06.icg.tu-graz.ac.at/mount/sem/MatCvsRep
%                    module: ICG_VisionTools
%
% These MATLAB functions are the property of the authors stated in the
% m-files. They are not a product of The MathWorks, Inc. and The MathWorks
% assumes no responsibility for any errors that may exist in these
% functions.
%
% While a reasonable effort has been made to create error-free functions,
% the authors of these functions assume no responsibility for any errors
% that may exist in these functions. Moreover, the authors do not accept
% any liability for errors that may result from the use of these functions.
%
% ------------------------------------------------------------------------
%
% For toolbox contents, see Contents.m
```