# The Dicomflex

## Purpose:

It is a Matlab based software template focusing on the creation of tools for systematic, similar image processing of image stacks (e.g.: CT, MRI, …).

## Main Benefits:

- quick creation and high grade of freedom in defining user input, image processing and result generation (Xls, etc.).
- similar appearance (Gui) of developed tools

## Additional Benefits:

- undo
- image zoom
- segmentation methods
- curve fitting methods
- systematic version handling to update session files
- automatic session saving

Image processing functions are not the focus of the framework itself. However, processing libraries or regular Matlab functions can be used to realize processing needs.

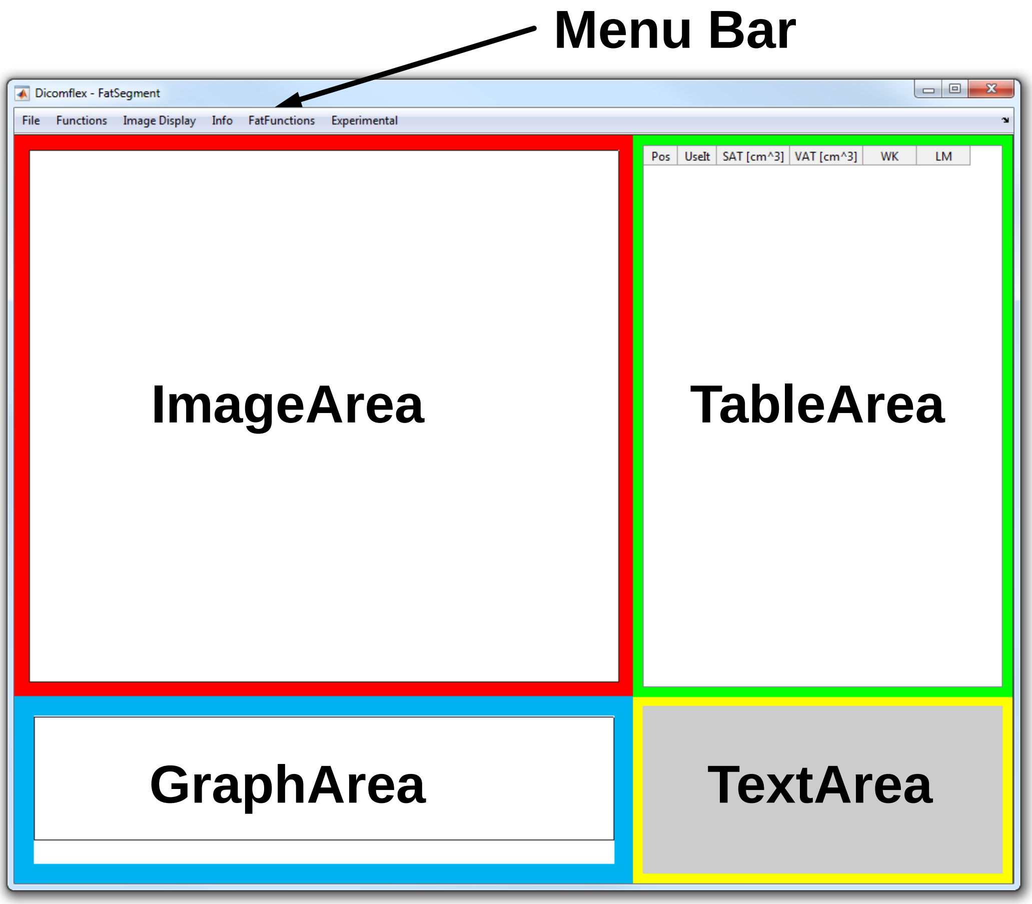
## Content:

The framework consists of a predefined, yet adaptable GUI, which is similar for every developed tool. Menu bar may be modified as well as GUI element callback methods. A simple class hierarchy and two configuration files separate clearly between framework and application specific parts of the Dicomflex.

## Adresses to people that:

- have little to medium Matlab knowledge
- know how object orientation works
- work with medical images
- have systematic image processing needs in the medical research setting

# User Interface



| Pos | UseIt | SAT [cm^3] | VAT [cm^3] | WK | LM |
|---|---|---|---|---|---|

## Table Area:

Each table row represents an image or images of a slice. Columns may vary regarding the application and may display „slice location“ or any result data.

## Image Area:

display of currently selected slice

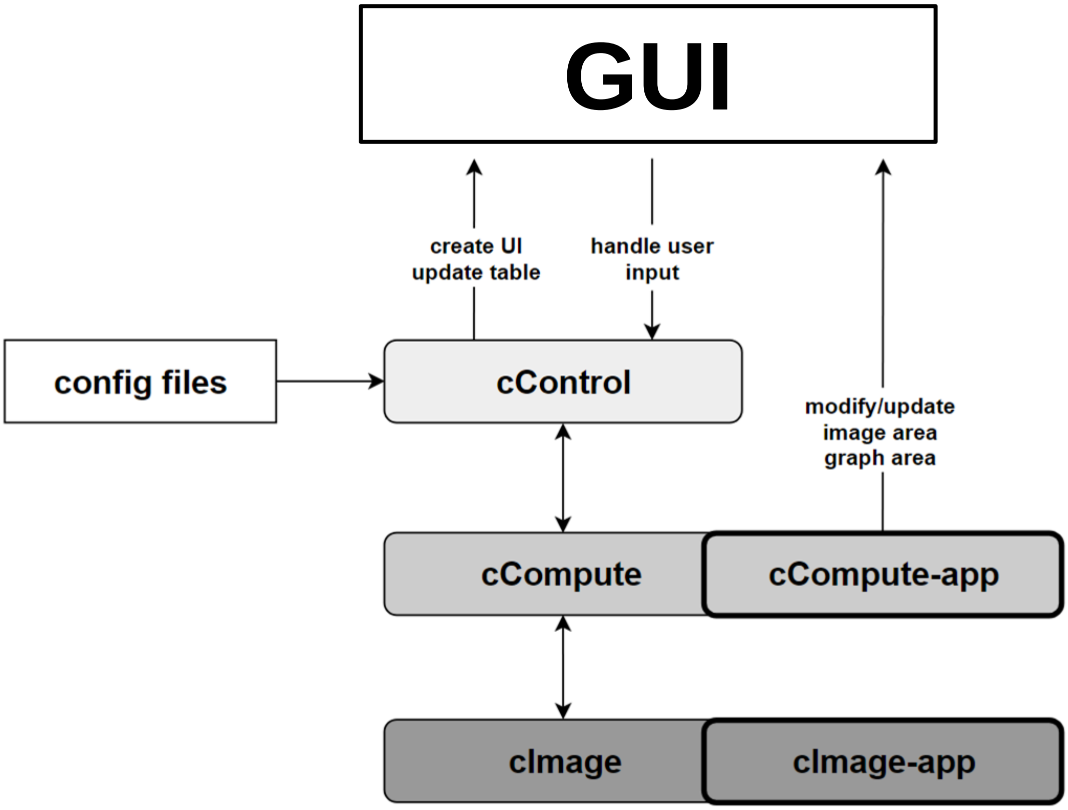
## Graph Area:

display of e.g. histograms, data values as graph, fit functions, …

## Text Area:

any kind of text. E.g. detailed information of the selected slice.

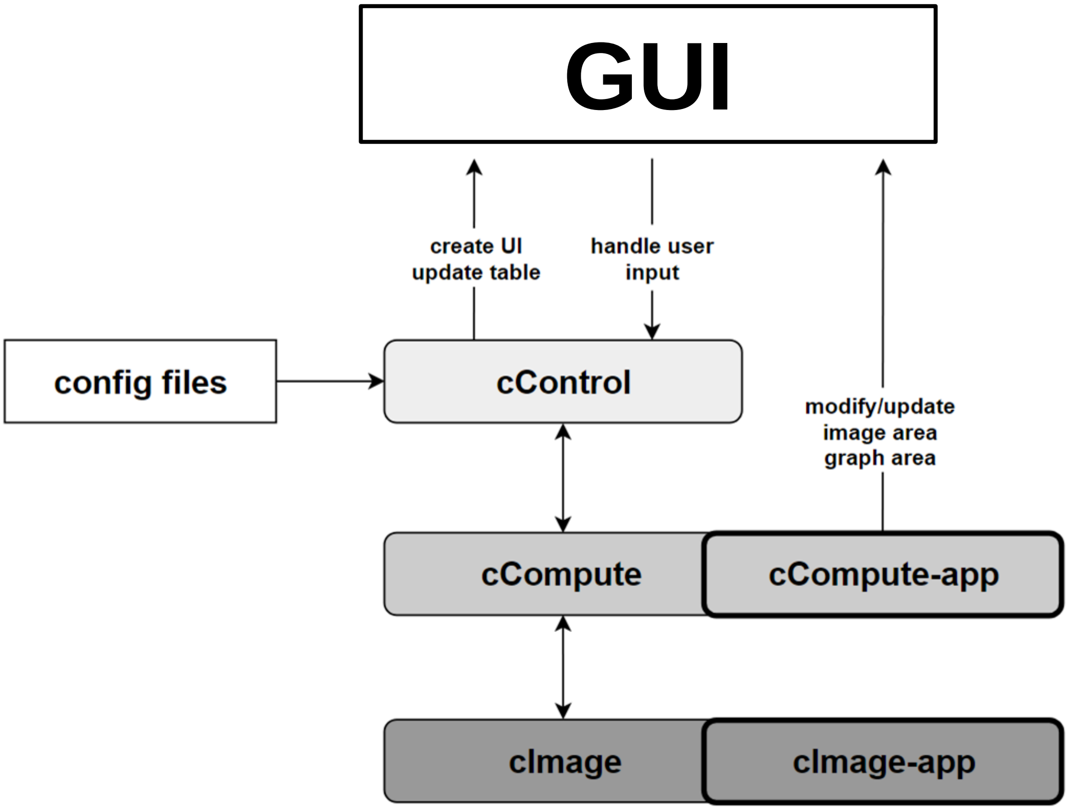
# Software Structure



## General Information:

Classes, methods, objects and properties are indicated by a leading prefix c, m, o and p respectively.
Besides the illustrated Classes there exists a class cBoundary.
All defined properties and methods are described in the Matlab code files.

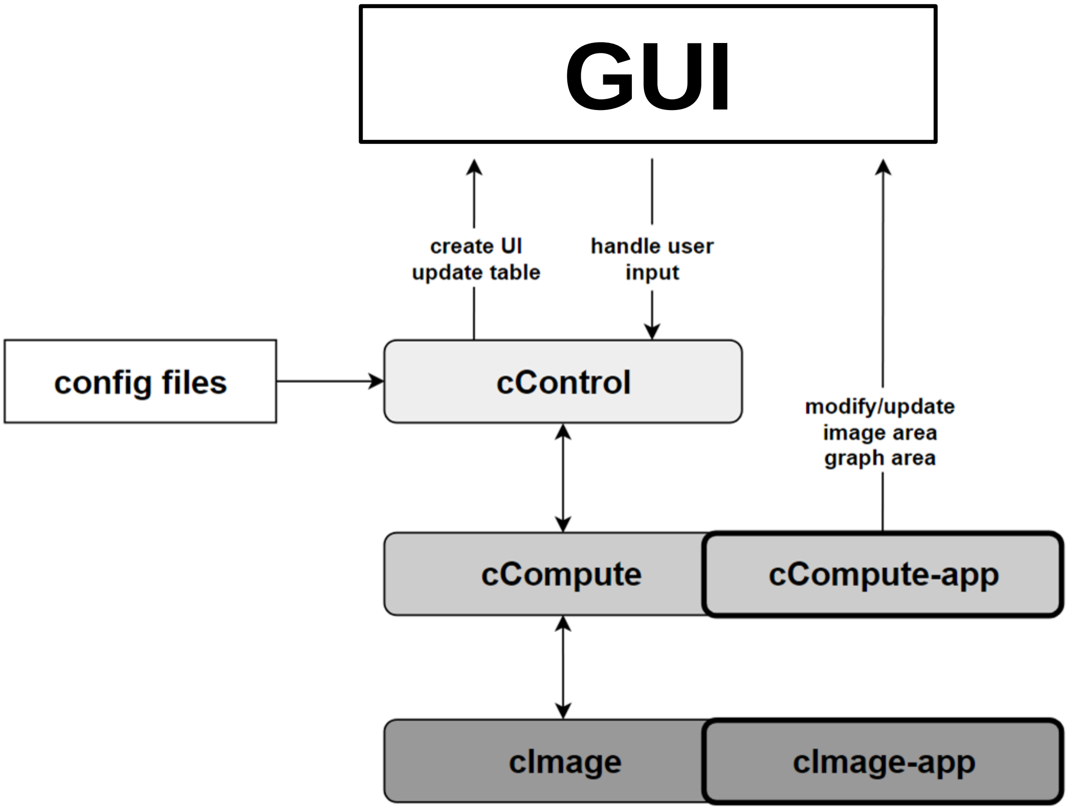
# Software Structure



Classes:

- **cControl**: it is the main framework class. It is not meant to be changed when creating a new tool. It contains methods for GUI creation and user input callback catching and their distribution.
- **cCompute**: each slice/image-data is hosted in a cCompute object. Thus each table row represents the data of a cCompute object. Besides image data and other source data it hosts generic methods (e.g.: image management, boundary/segmentation or fitting methods) or general method parts (e.g.: data loading/saving or GUI update methods) for application specific code.
- **cCompute-app**: contains the mode specific code as well as mandatory mode specific methods called by cControl or cCompute class (e.g.: data loading/saving, GUI update/interaction methods). It is a child class of the cCompute class and thus has access to cCompute methods.

# Software Structure

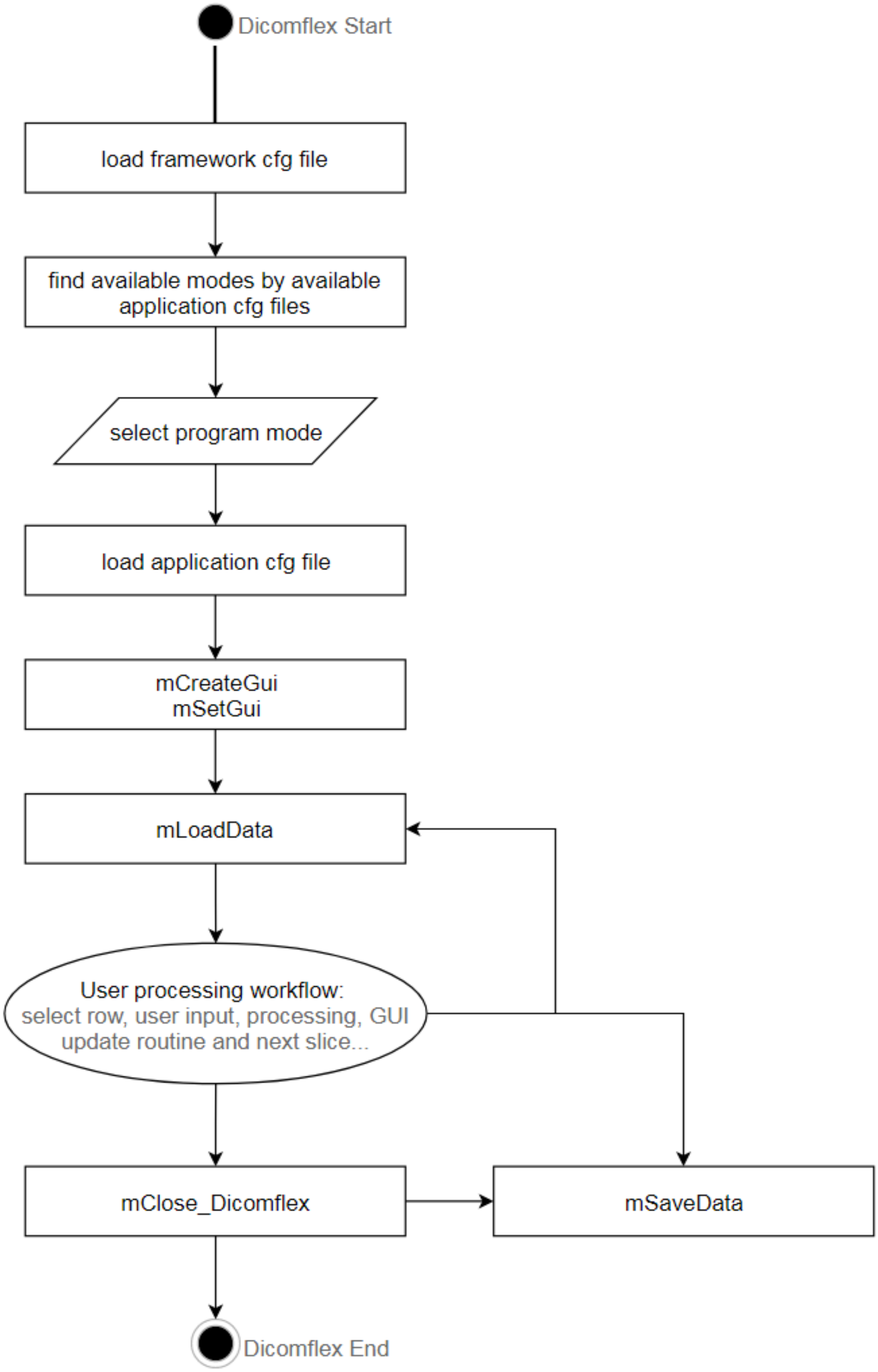


## Config files:

two files in the JSON format host configuration values. They include slight GUI modification, menu bar definition and callback definitions mainly. Creation of those (cfg_framework.json and cfg_application_*.json) files is comfortably possible by using matlab scripts (ConfigFramework_template.m and ConfigApplication_template.m).
The application config file is separated in two parts: „ mandatory app values” and „app specific values”. Description of entries is available in the *template.m files.

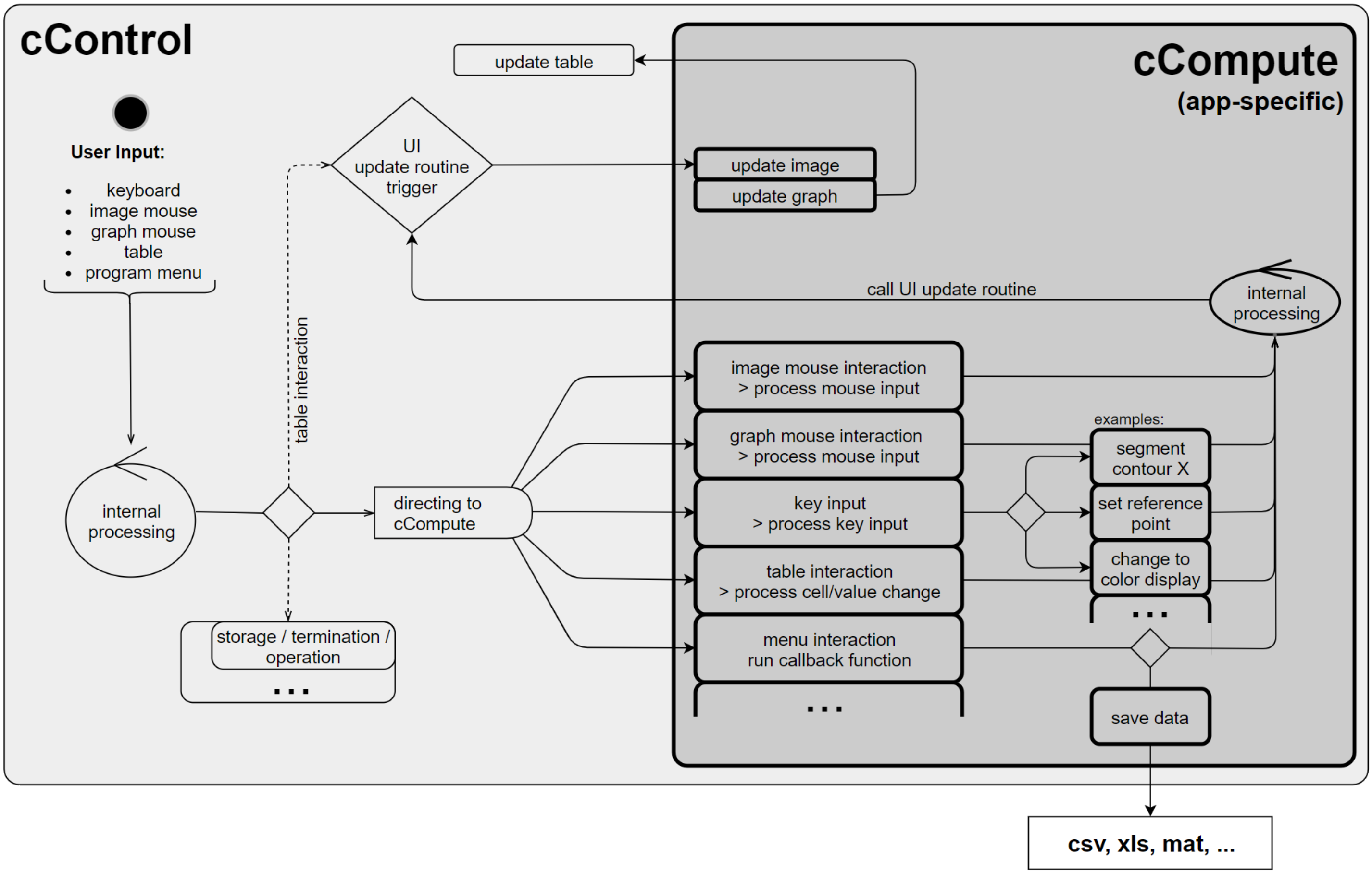
# Software Workflow

## Dicomflex FlowChart


Dicomflex Start
load framework cfg file
find available modes by available application cfg files
select program mode
load application cfg file
mCreateGui
mSetGui
mLoadData
User processing workflow:
select row, user input, processing, GUI update routine and next slice...
mClose_Dicomflex
mSaveData
Dicomflex End

# Software Workflow

## User Input Processing

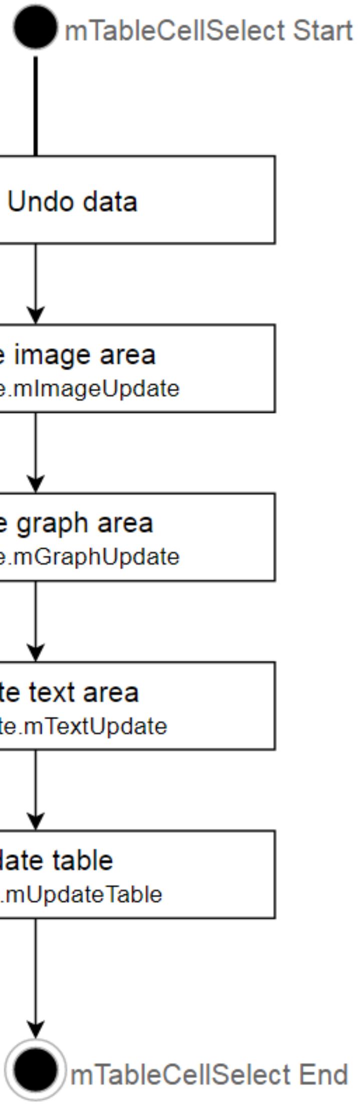


All user input is caught by the cControl class. If not directly processed there (e.g. when selecting another slice in the TableArea), it is redirected to the appropriate method of the cCompute class. The called cCompute method might do some general things and might forward to the cCompute-app class. After processing the GUI-update routine is triggered by the cControl.

Out of framework definitions of GUI element callbacks by the cCompute-app directly to methods in cCompute-app are possible to e.g. create and control additional GUI elements available for a certain application only.

# Software Workflow

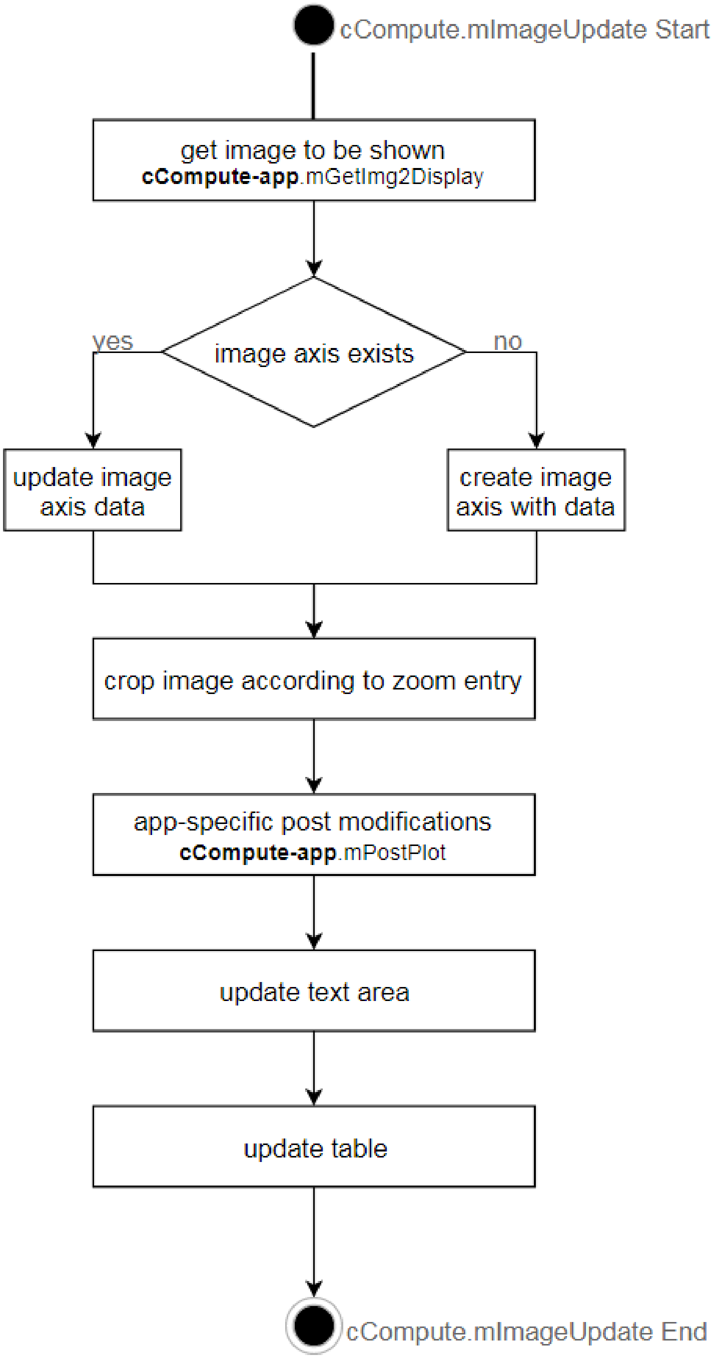
## GUI update routine



The GUI update routine may be triggered from anywhere in the Dicomflex by calling the method cControl.mTableCellSelect

# Software Workflow

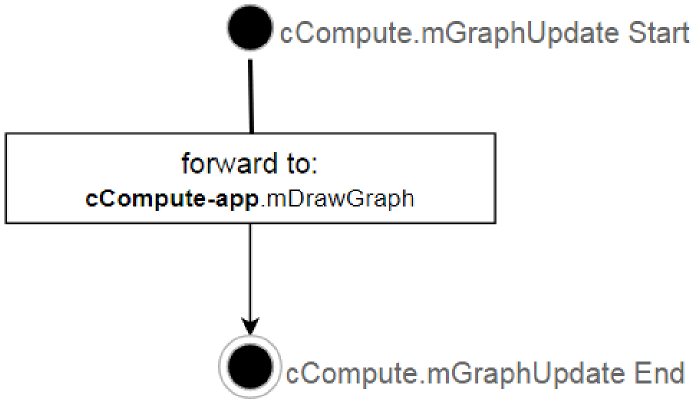
## ImageUpdate


cCompute.mImageUpdate Start
get image to be shown
cCompute-app.mGetImg2Display
image axis exists
yes
no
update image axis data
create image axis with data
crop image according to zoom entry
app-specific post modifications
cCompute-app.mPostPlot
update text area
update table
cCompute.mImageUpdate End

The image update is commonly called by the GUI update routine from cControl.mTableCellSelect. It is a framework method that calls two app-specific mandatory methods to be hosted in cCompute-app (mGetImg2Display and mPostPlot).

# Software Workflow

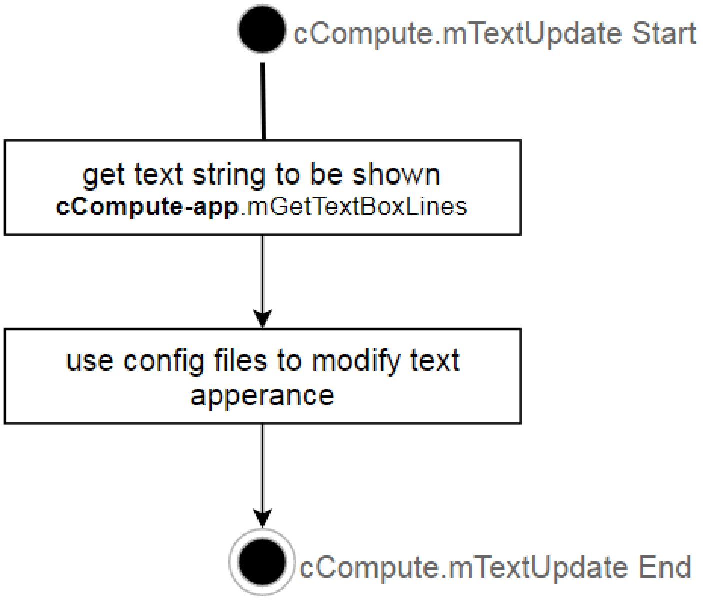
## GraphUpdate



The graph update is commonly called by the GUI update routine from cControl.mTableCellSelect. It is directly forwarded by the cControl to the cControl-app method mDrawGraph.

# Software Workflow

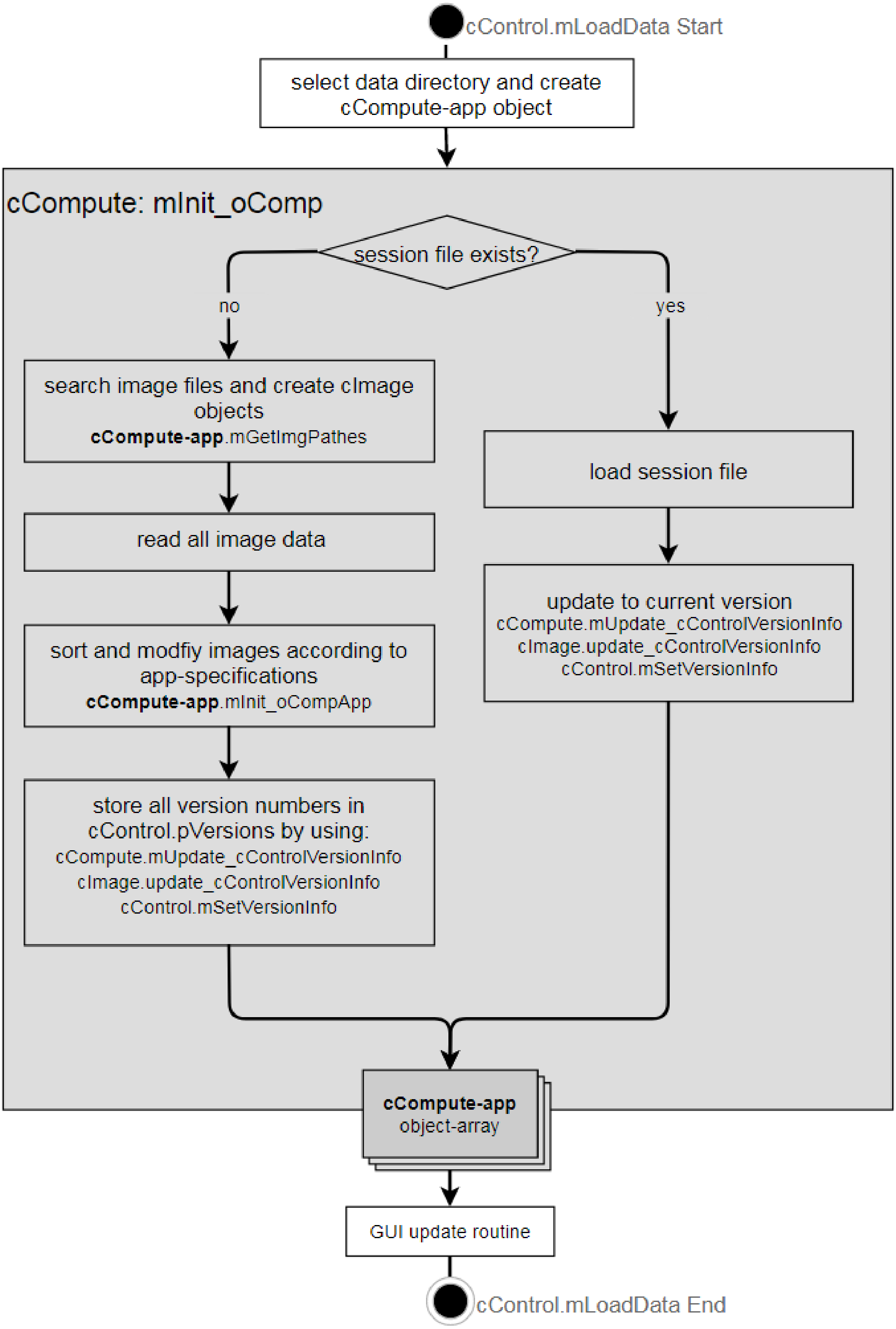
## TextUpdate


cCompute.mTextUpdate Start
get text string to be shown
**cCompute-app**.mGetTextBoxLines
use config files to modify text apperance
cCompute.mTextUpdate End

The text update is commonly called by the GUI update routine from cControl.mTableCellSelect. It uses only one app-specific method from the cCompute-app class to get the string to be displayed.

# Software Workflow

## Load Data



cControl.mLoadData Start

select data directory and create cCompute-app object

cCompute: mInit_oComp

session file exists?

no

yes

search image files and create cImage objects
**cCompute-app**.mGetImgPathes

read all image data

sort and modfiy images according to app-specifications
**cCompute-app**.mInit_oCompApp

store all version numbers in cControl.pVersions by using:
cCompute.mUpdate_cControlVersionInfo
cImage.update_cControlVersionInfo
cControl.mSetVersionInfo

load session file

update to current version
cCompute.mUpdate_cControlVersionInfo
cImage.update_cControlVersionInfo
cControl.mSetVersionInfo

**cCompute-app**
object-array

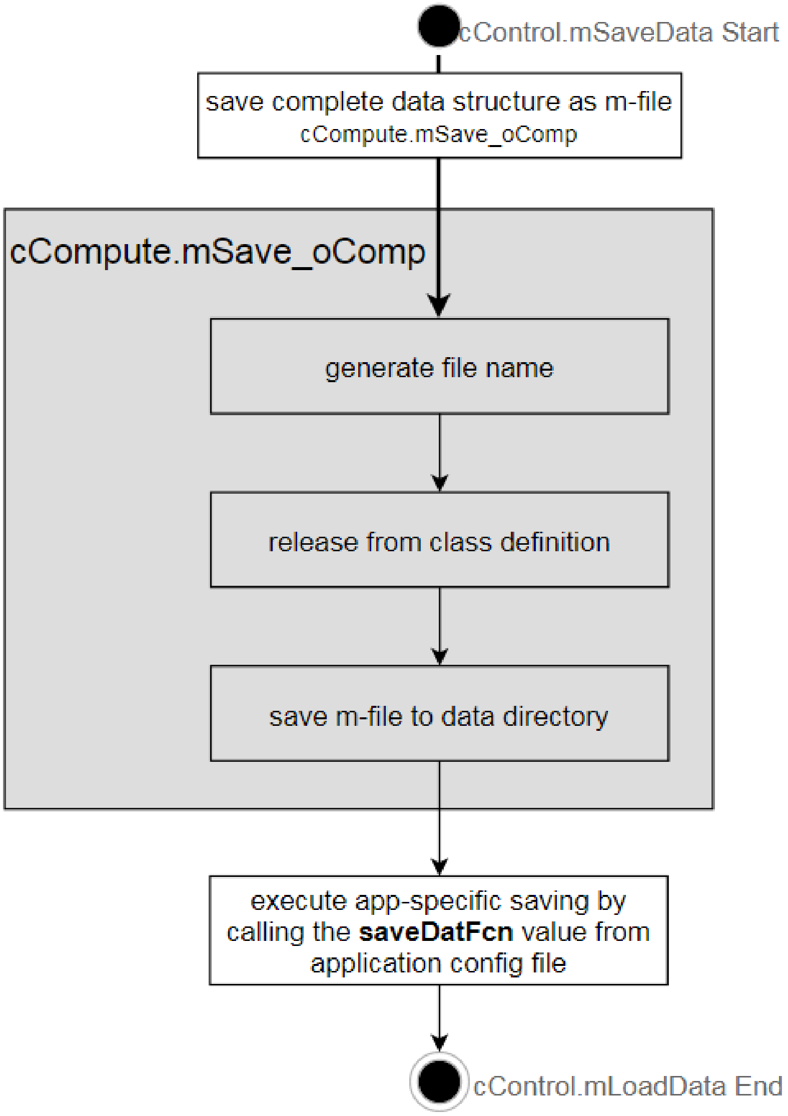
GUI update routine

cControl.mLoadData End

mLoadData is called after program start as well after clicking the „Load data“ button of the menu bar. App-specification needs to be done in cCompute-app methods mGetImgPathes and mInit_oCompApp.

# Software Workflow

## Save Data



cControl.mSaveData Start
save complete data structure as m-file
cCompute.mSave_oComp
cCompute.mSave_oComp
generate file name
release from class definition
save m-file to data directory
execute app-specific saving by calling the **saveDatFcn** value from application config file
cControl.mLoadData End

mSaveData is called clicking the „Save data“ button of the menu bar. Automatic saving is repeatedly done after the time specified by the „datAutoSaveTime” value in the framework config. Additionally mSaveData may be chosen when clicking x-close at the program window. App-specification needs to be done in cCompute-app methods mGetImgPathes and mInit_oCompApp.

# Software Workflow

## Versionhandling/SessionFile Update

Background info to Session files:

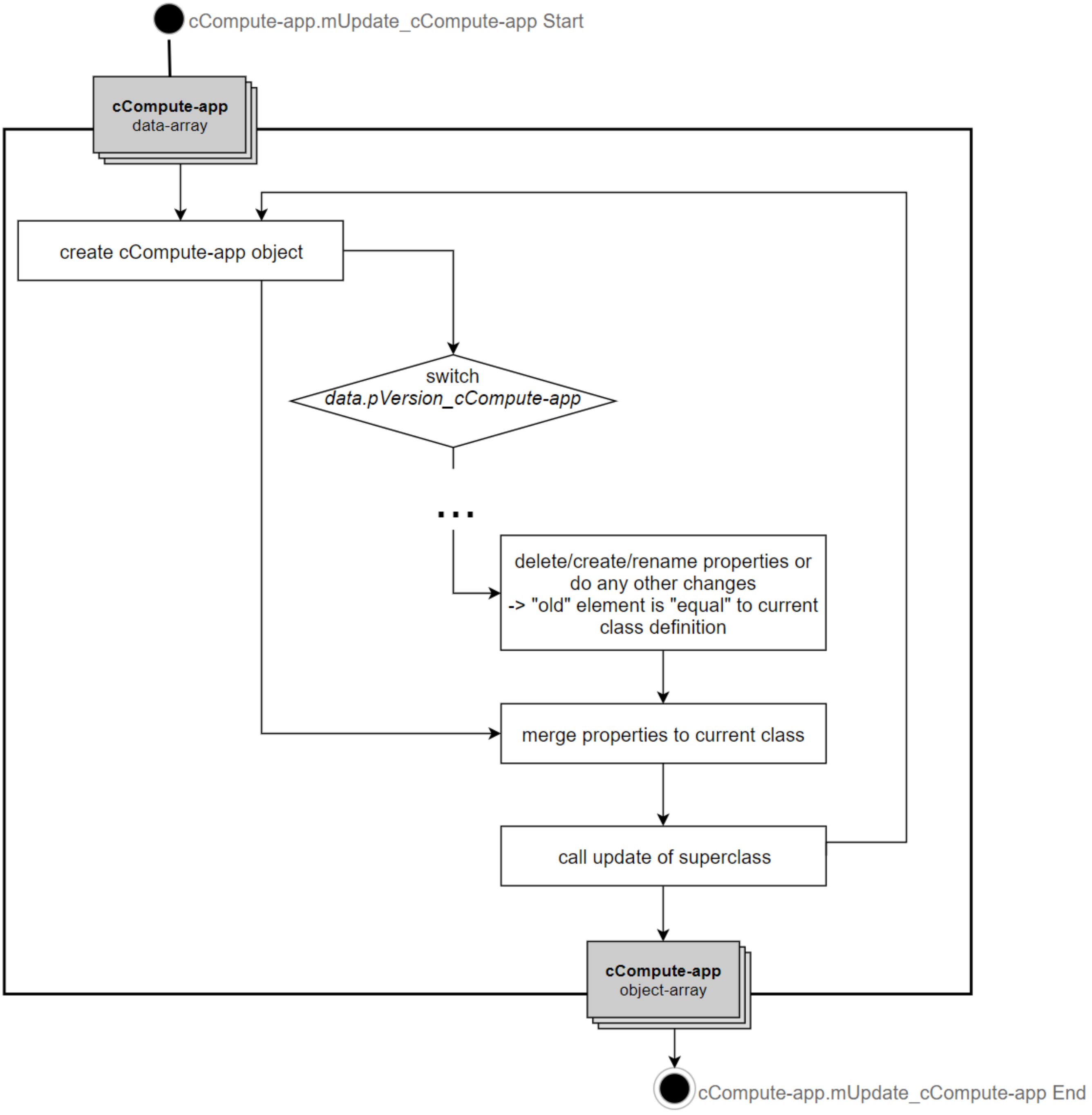
A session file consists of the following data:

- *saveDate* - the creation date of the file
- *versions* - a struct with all class-names, their versions and the corresponding updateFcn, as well as the version numbers of the two config files.
- *data* - a struct array containing all oCompute-app elements as they where used in the application. They are disconnected from their class definition.

When changes where made to the Dicomflex (cControl, cCompute) and/or any application (cCompute-app) prior session files saved with a Dicomflex application might not be loadable. To overcome this, we chose the following handling of session files: If loading a session file in the cCompute.mInit_oComp, the update to current version is called by oComp = oComp.(['mUpdate_' class(oComp)])(data, saveDate);. Meaning, that in each cCompute-app class exists a method called mUpdate_cCompute-*app*, in which a switch-case is used to discriminate between versions of the classes used in the "old" session file. In the cases the update to the current version and the merging in the cCompute-app class as an object is done.

# Software Workflow

## Versionhandling/SessionFile Update


cCompute-app.mUpdate_cCompute-app Start
cCompute-app
data-array
create cCompute-app object
switch
data.pVersion_cCompute-app
...
delete/create/rename properties or do any other changes -> "old" element is "equal" to current class definition
merge properties to current class
call update of superclass
cCompute-app
object-array
cCompute-app.mUpdate_cCompute-app End

# App Creation

## overview

To create a unique, new application the following parts need to be modified:

1. cCompute-app.m file
    - customize mandatory methods called from cControl
        - mInit_oCompApp
        - mGetImg2Display
        - mPostPlot
        - mDrawGraph
        - mGetTextBoxLines
        - mTableEdit
        - mShowHotkeyInfo
        - mKeyPress
        - mKeyRelease
        - mImgAxisButtonDown
        - mImgAxisButtonMotion
        - mImgAxisButtonUp
        - mUpdate_oComputeT1Mapper or mUpdate_oComputeFatSegment
        - mTabDat2oCompApp
    - implement app specific properties and methods
2. applicationConfig.JSON:
   use ConfigApplication_template.m for creation.

# App Creation

## cCompute-app.m

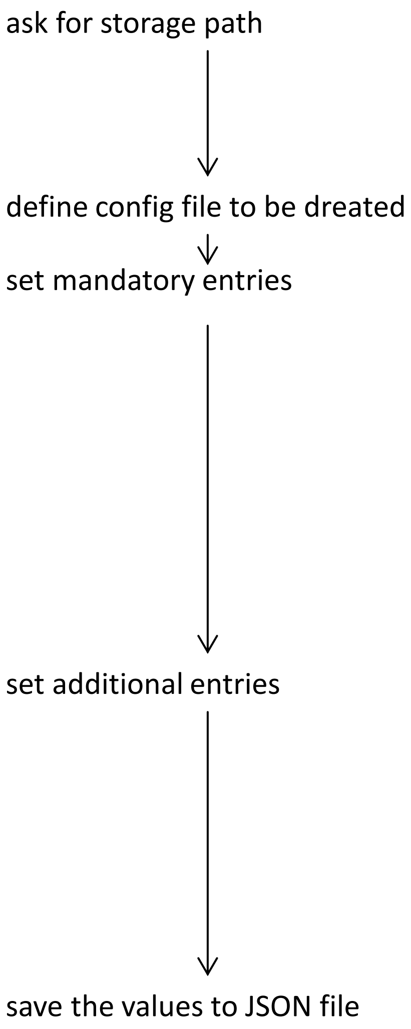
1. define application name. For this tutorial: *Dummy*
2. make copy of cCompute_template.m and rename to cComputeDummy.m
3. open in Matlab and rename classname at line 1:
   classdef cComputeDummy<cCompute
4. change property name pVersion_cCompute_template:
   pVersion_cComputeDummy
5. change method name of mUpdate_cCompute_template to:
   mUpdate_cComputeDummy
   1. change object creation line in for loop to:
      oComp(i) = cComputeDummy;
   2. change switch discriminator to:
      oCompTmp.pVersion_cComputeDummy;
6. change class creation method cCompute_template name to:
   cComputeDummy
7. Insert application specific code at all points indicated with %-%-%
8. Note, that the cCompute_template.m:
   - uses sliceLocations value to find images and sort them to oComp objects in mInit_oCompApp
   - has some example code in commented form
   - use already existing cCompute-app.m files to reuse stuff (e.g. Boundary management or data fitting)

# App Creation

## ConfigApplication_template.m

Modifications in the ConfigApplication_template.m are necessary in the *mandatory app values* section. A switch statement is used to discriminate between the different applications. To create a new config file do the following:

1. copy the *_template_* case and rename it to the new application name *Dummy*
2. change the value cfg.applicationName to *Dummy*
3. customize all mandatory entrys to suite your application
4. additional entries may be stored in the switch statement *app specific* values under the *Dummy* case.



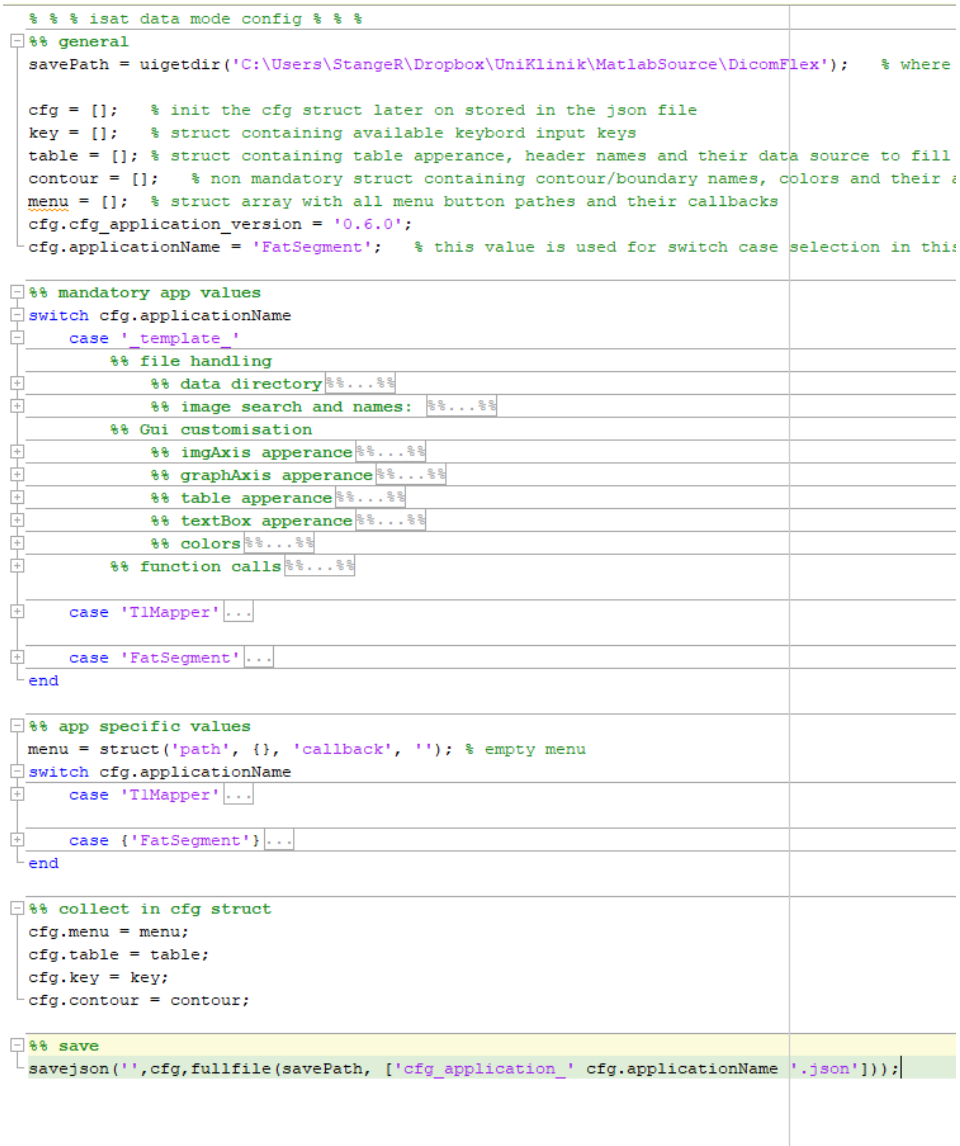
ask for storage path

define config file to be dreated

set mandatory entries

set additional entries

save the values to JSON file

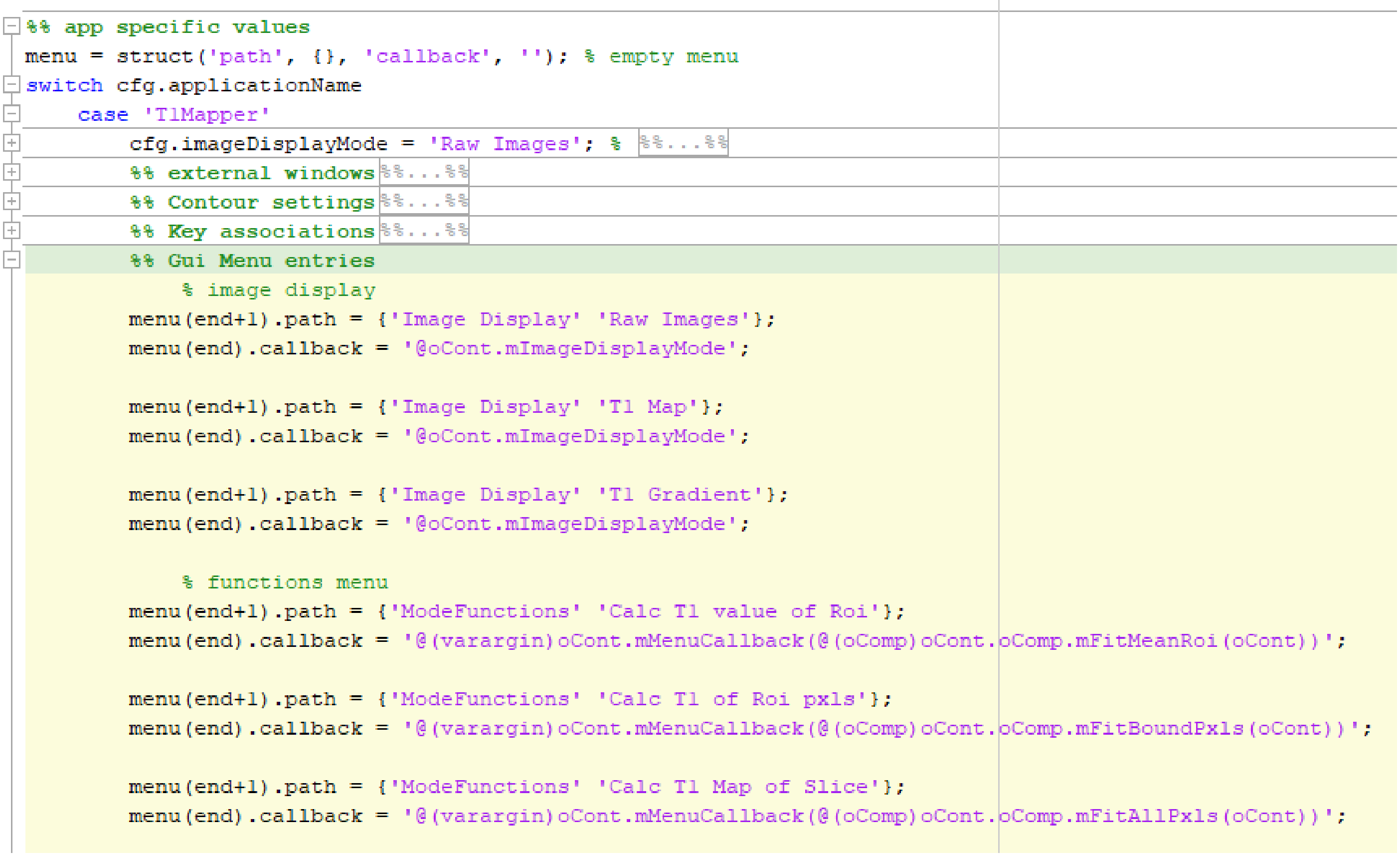


```
% % % isat data mode config % % %
%% general
savePath = uigetdir('C:\Users\StangeR\Dropbox\UniKlinik\MatlabSource\DicomFlex');   % where

cfg = [];   % init the cfg struct later on stored in the json file
key = [];   % struct containing available keybord input keys
table = []; % struct containing table apperance, header names and their data source to fill
contour = [];   % non mandatory struct containing contour/boundary names, colors and their a
menu = [];  % struct array with all menu button pathes and their callbacks
cfg.cfg_application_version = '0.6.0';
cfg.applicationName = 'FatSegment';   % this value is used for switch case selection in this

%% mandatory app values
switch cfg.applicationName
    case '_template_'
        %% file handling
            %% data directory%%...%%
            %% image search and names: %%...%%
        %% Gui customisation
            %% imgAxis apperance%%...%%
            %% graphAxis apperance%%...%%
            %% table apperance%%...%%
            %% textBox apperance%%...%%
            %% colors%%...%%
        %% function calls%%...%%

    case 'T1Mapper'...

    case 'FatSegment'...
end

%% app specific values
menu = struct('path', {}, 'callback', ''); % empty menu
switch cfg.applicationName
    case 'T1Mapper'...

    case {'FatSegment'}...
end

%% collect in cfg struct
cfg.menu = menu;
cfg.table = table;
cfg.key = key;
cfg.contour = contour;

%% save
savejson('',cfg,fullfile(savePath, ['cfg_application_' cfg.applicationName '.json']));
```

# App Creation

## Menue Bar

There exist some menu bar entries defined in the framework config. Despite those it is possible to define more in the application config in a similar manner. To do so, use the ConfigApplication_template.m, in which all application configs are stored for creation of a config.json file. Each element of the struct array "menu" represents a menu bar entry. Each element contains a cell with strings called "path" defining the path of the menu bar button (e.g.: {'File' 'Load'} would create a button "Load" in the "File" menu). For each button exists a "callback" value stored per array element. It may directly point to a cCompute method or may be caught by the cControl.mMenuCallback method and be redirected there further.



```
%% app specific values
menu = struct('path', {}, 'callback', ''); % empty menu
switch cfg.applicationName
    case 'T1Mapper'
        cfg.imageDisplayMode = 'Raw Images'; % %%...%%
        %% external windows%%...%%
        %% Contour settings%%...%%
        %% Key associations%%...%%
        %% Gui Menu entries
            % image display
        menu(end+1).path = {'Image Display' 'Raw Images'};
        menu(end).callback = '@oCont.mImageDisplayMode';

        menu(end+1).path = {'Image Display' 'T1 Map'};
        menu(end).callback = '@oCont.mImageDisplayMode';

        menu(end+1).path = {'Image Display' 'T1 Gradient'};
        menu(end).callback = '@oCont.mImageDisplayMode';

            % functions menu
        menu(end+1).path = {'ModeFunctions' 'Calc T1 value of Roi'};
        menu(end).callback = '@(varargin)oCont.mMenuCallback(@(oComp)oCont.oComp.mFitMeanRoi(oCont))';

        menu(end+1).path = {'ModeFunctions' 'Calc T1 of Roi pxls'};
        menu(end).callback = '@(varargin)oCont.mMenuCallback(@(oComp)oCont.oComp.mFitBoundPxls(oCont))';

        menu(end+1).path = {'ModeFunctions' 'Calc T1 Map of Slice'};
        menu(end).callback = '@(varargin)oCont.mMenuCallback(@(oComp)oCont.oComp.mFitAllPxls(oCont))';
```

Menu bar entries may be created and organized therewith easily. The callbacks simply need to be computable by Matlab.