The Dicomflex

Purpose:

It is a Matlab (R2016b) 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
- contour tracking
- 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.

Addresses 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

Quick Start

start the program

- To get started include the main folder, subfun, and FitTool to your Matlab directories and type cControl to start.
- A list dialog will appear to choose from the available modes.
  - Available modes are determined by the existence of cfg_application_*.json files in the main directory.
  - for each cfg_application_*.json a corresponding cCompute*.m file must exist. It is the class defninition file containing mandatory and mode specific properties and methods.
  - you may copy cfg_application_*.json from the examples to the cControl.m directory for testing.
- After selection you need to choose a data folder suitable to the selected mode.

simple working example: Dummy

- see section Sample Application – Dummy
- to make changes open the cComputeDummy.m file in Matlab

User Interface

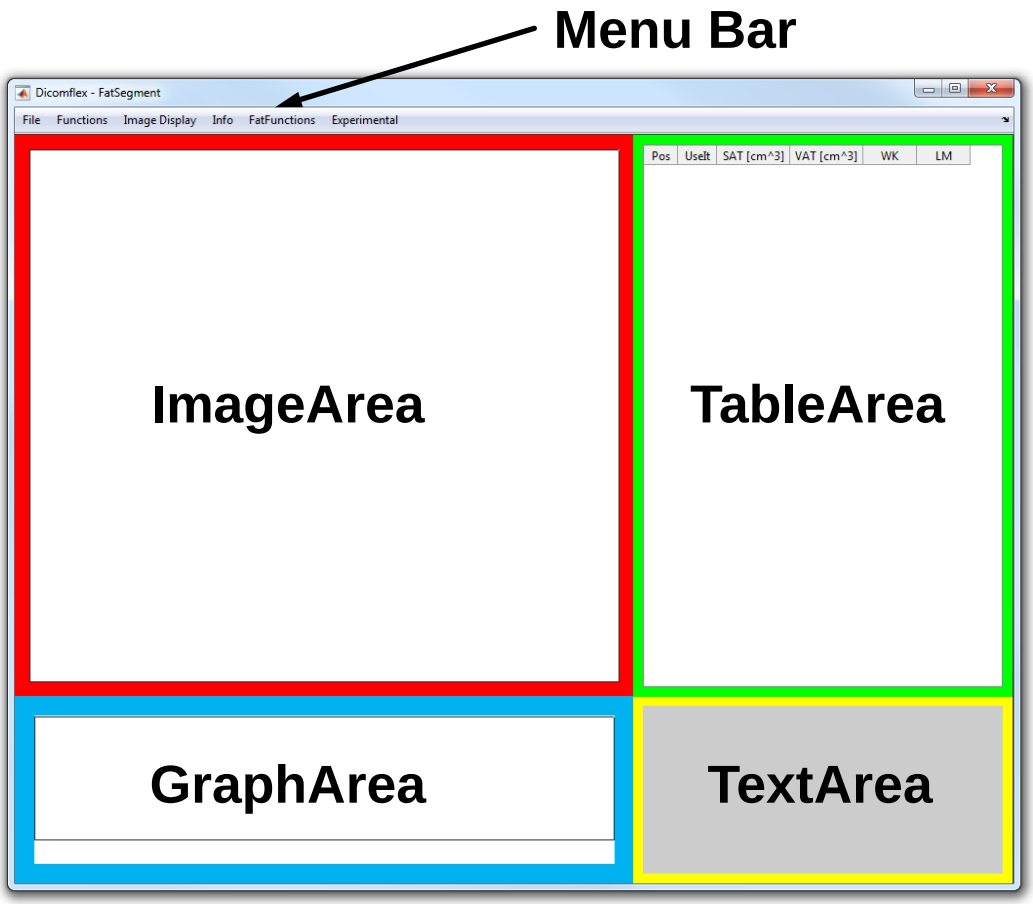


Table Area:

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

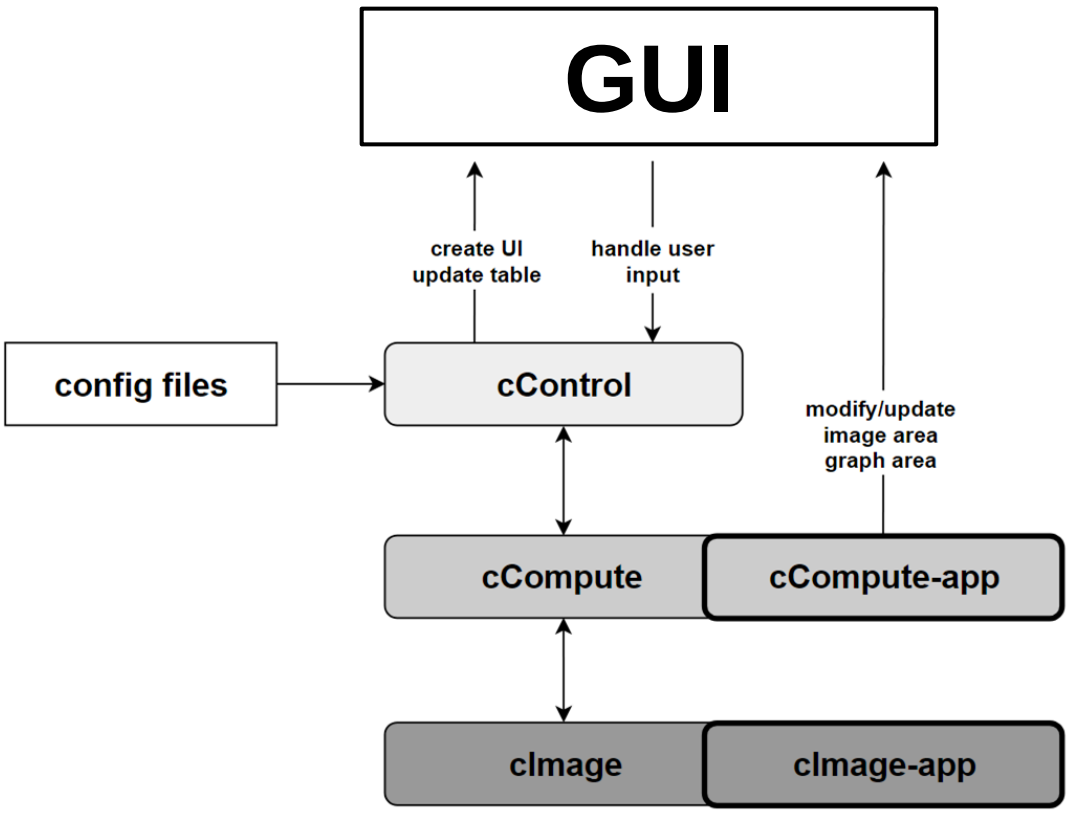
Graph Area:

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.

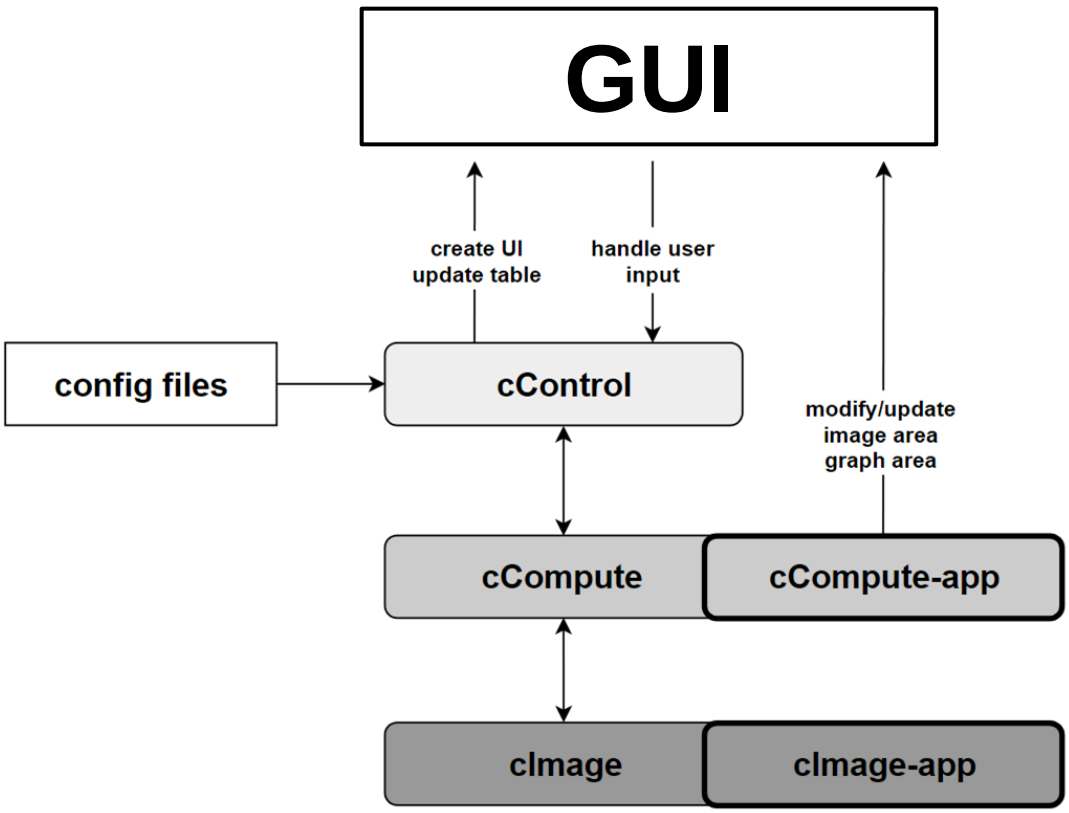
Software Structure



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.

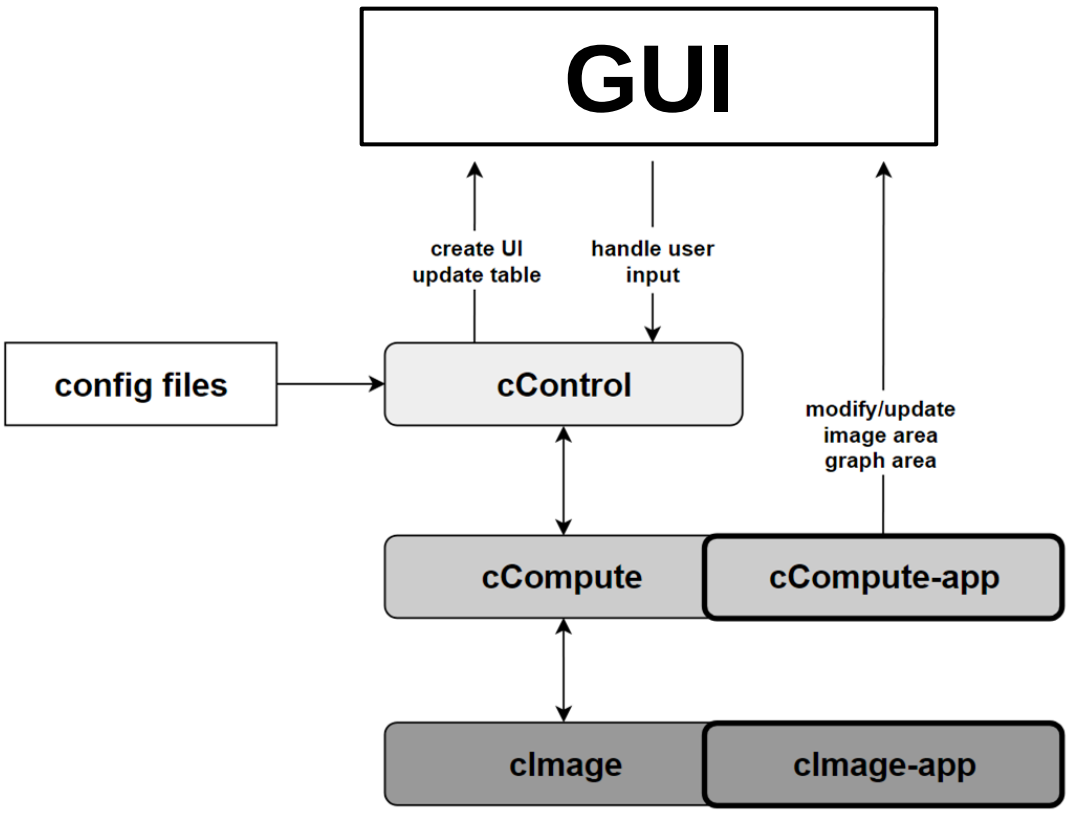
Software Structure



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

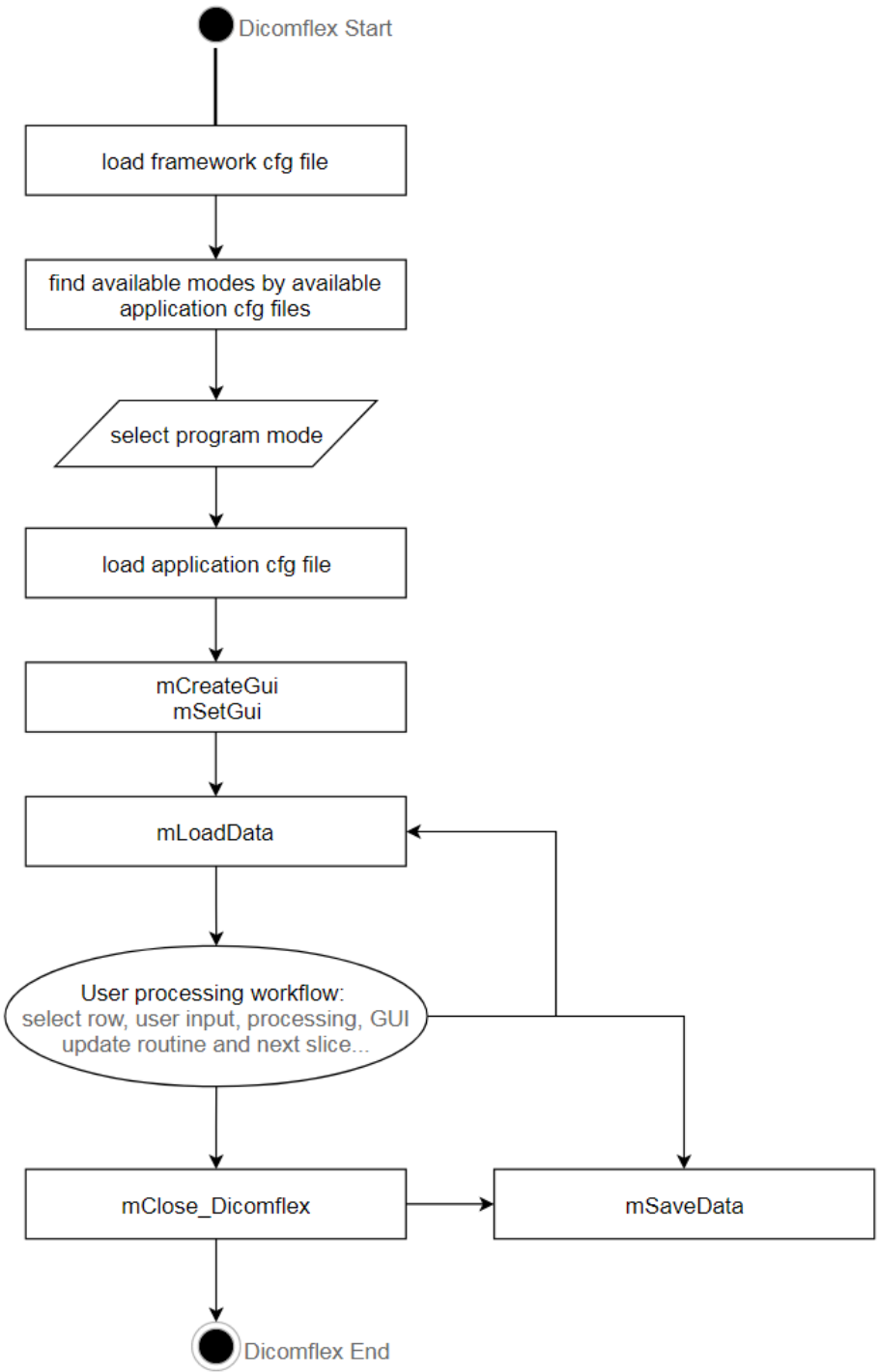


Config files:

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

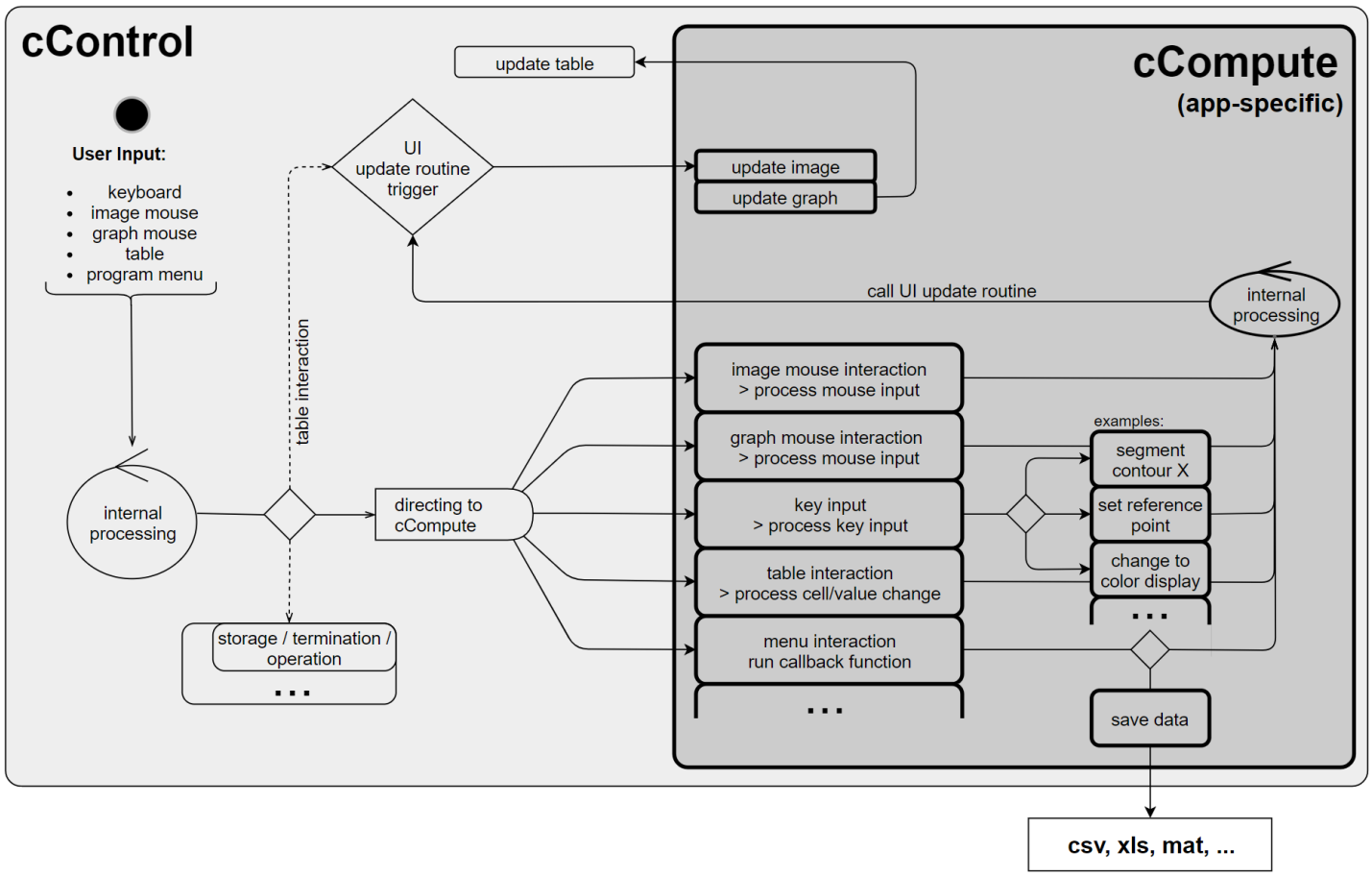
Dicomflex FlowChart



To start the Dicomflex from the Matlab console type cControl to create a cControl instance.

Software Workflow

User Input Processing

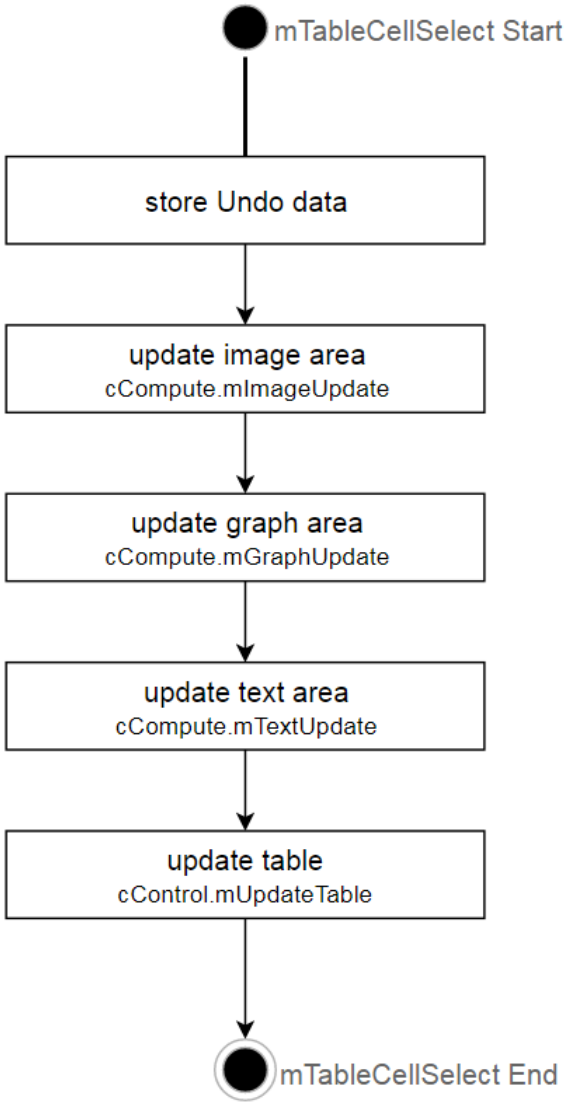


All user input is caught by the cControl class. If not directly processed there (e.g. when selecting another slice in the TableArea), it is redirected to the appropriate method of the cCompute class. The called cCompute method might do some general things and might forward to the cCompute-app class.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

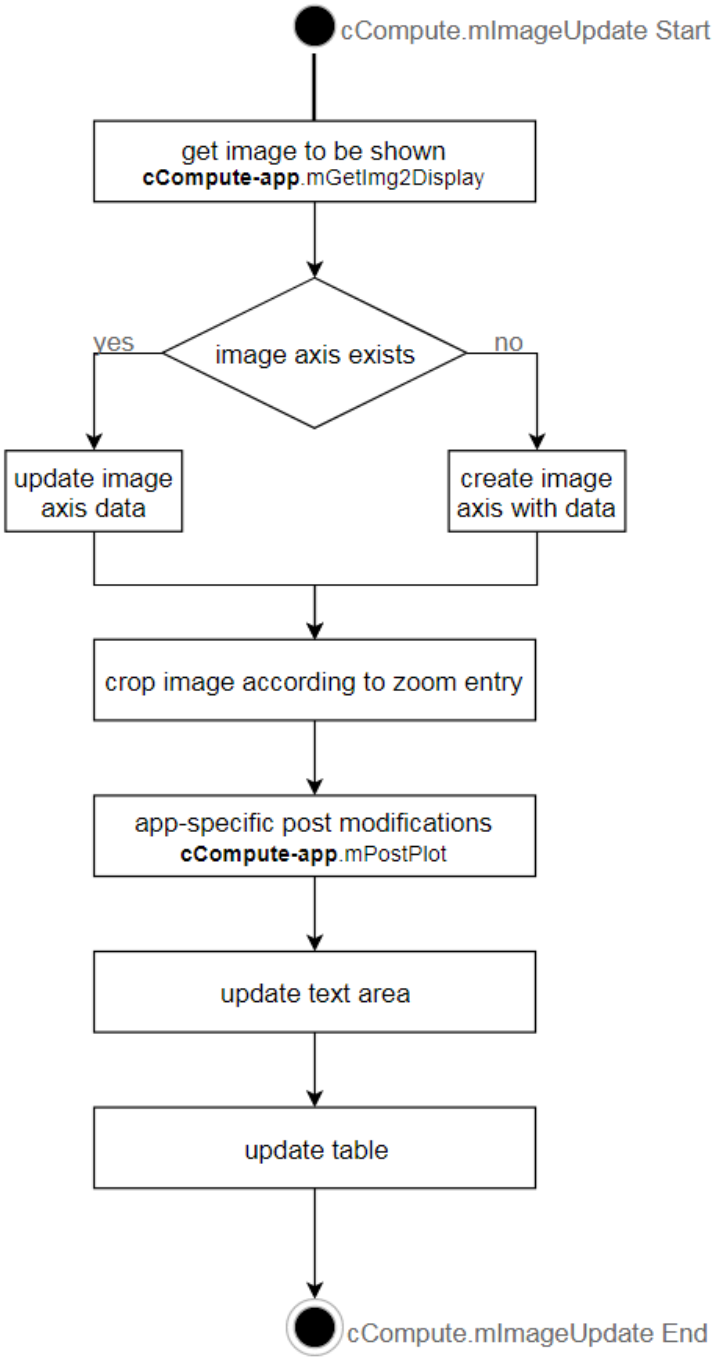
GUI update routine



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

Software Workflow

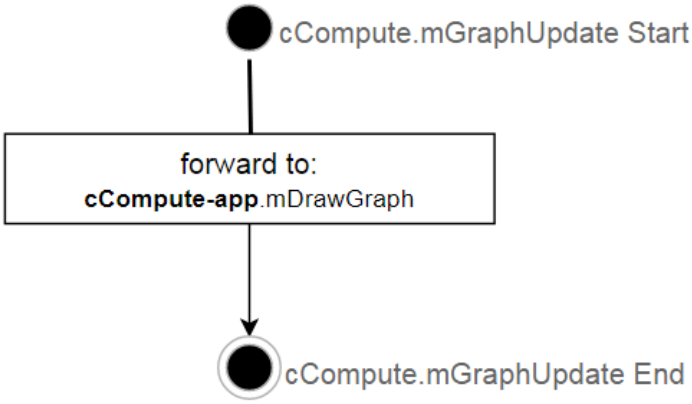
ImageUpdate



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

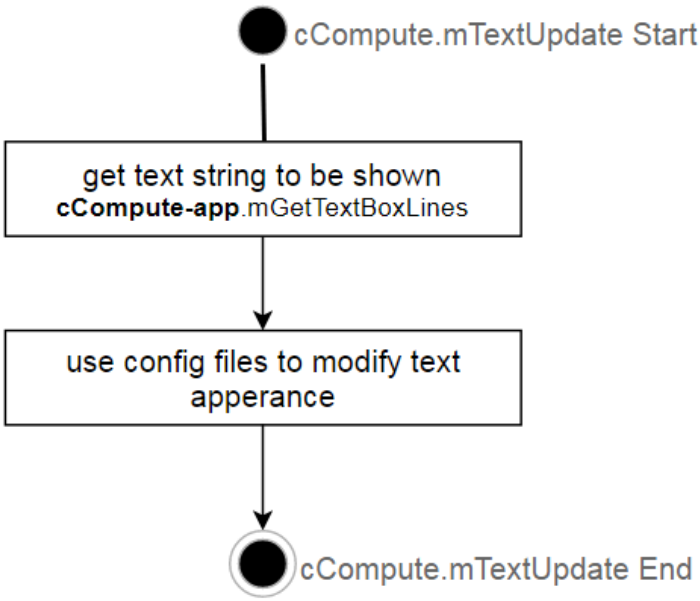
GraphUpdate



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

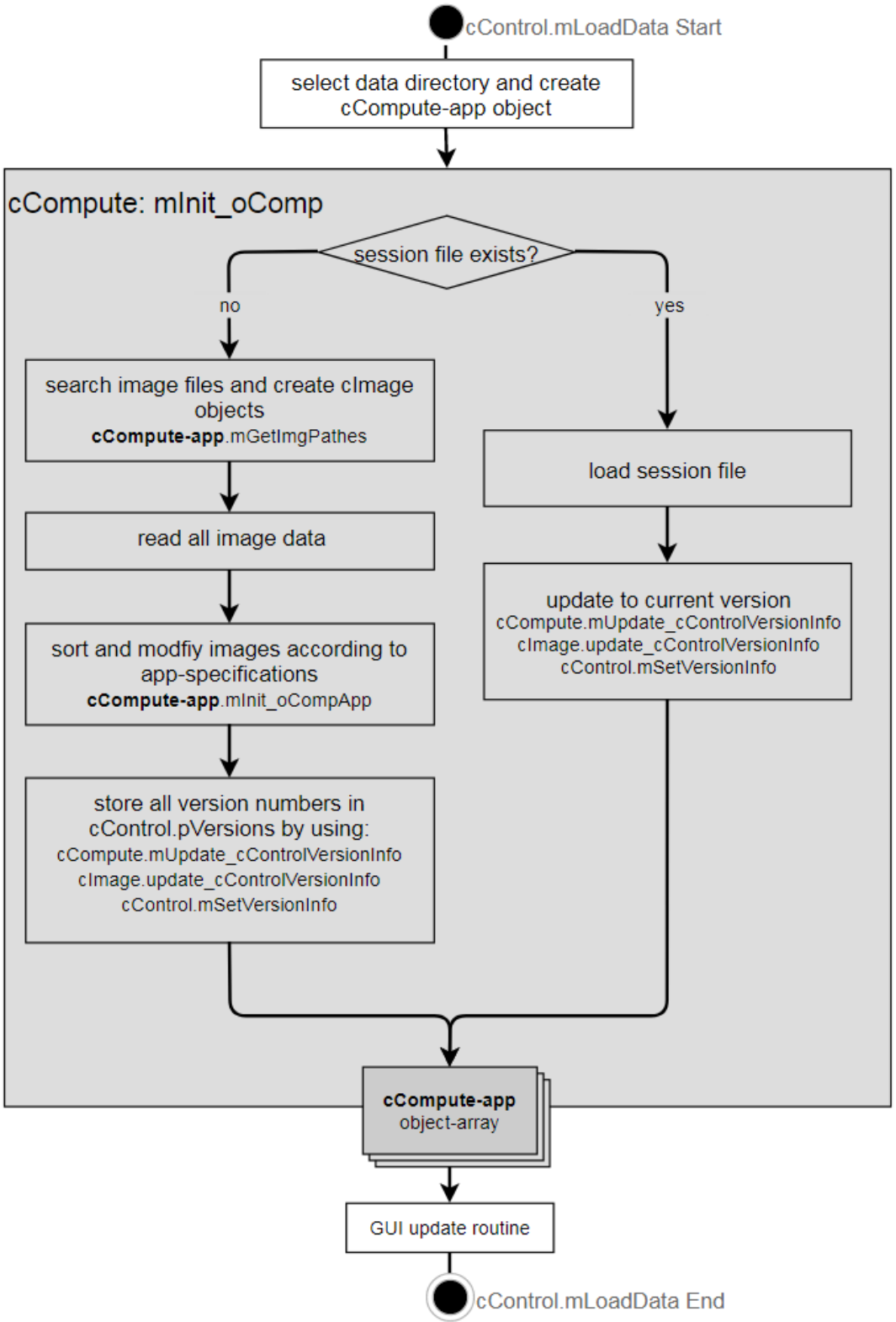
TextUpdate



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

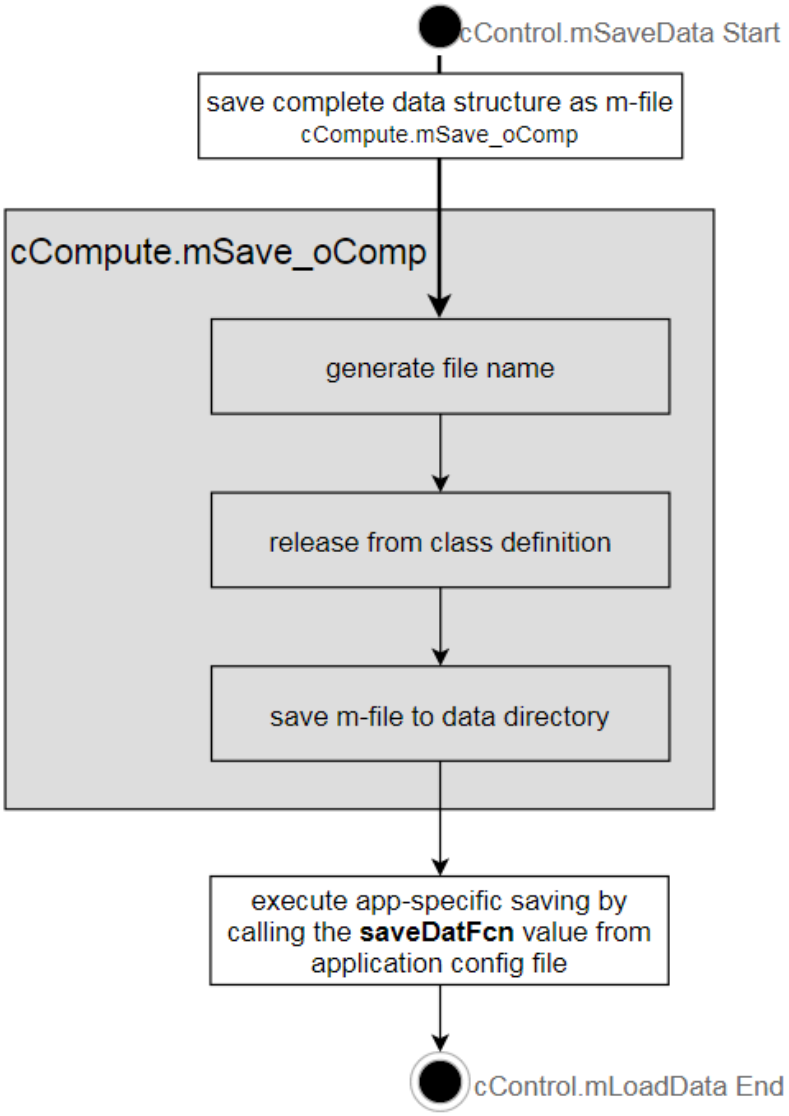
Load Data



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



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

Version handling/SessionFile Update

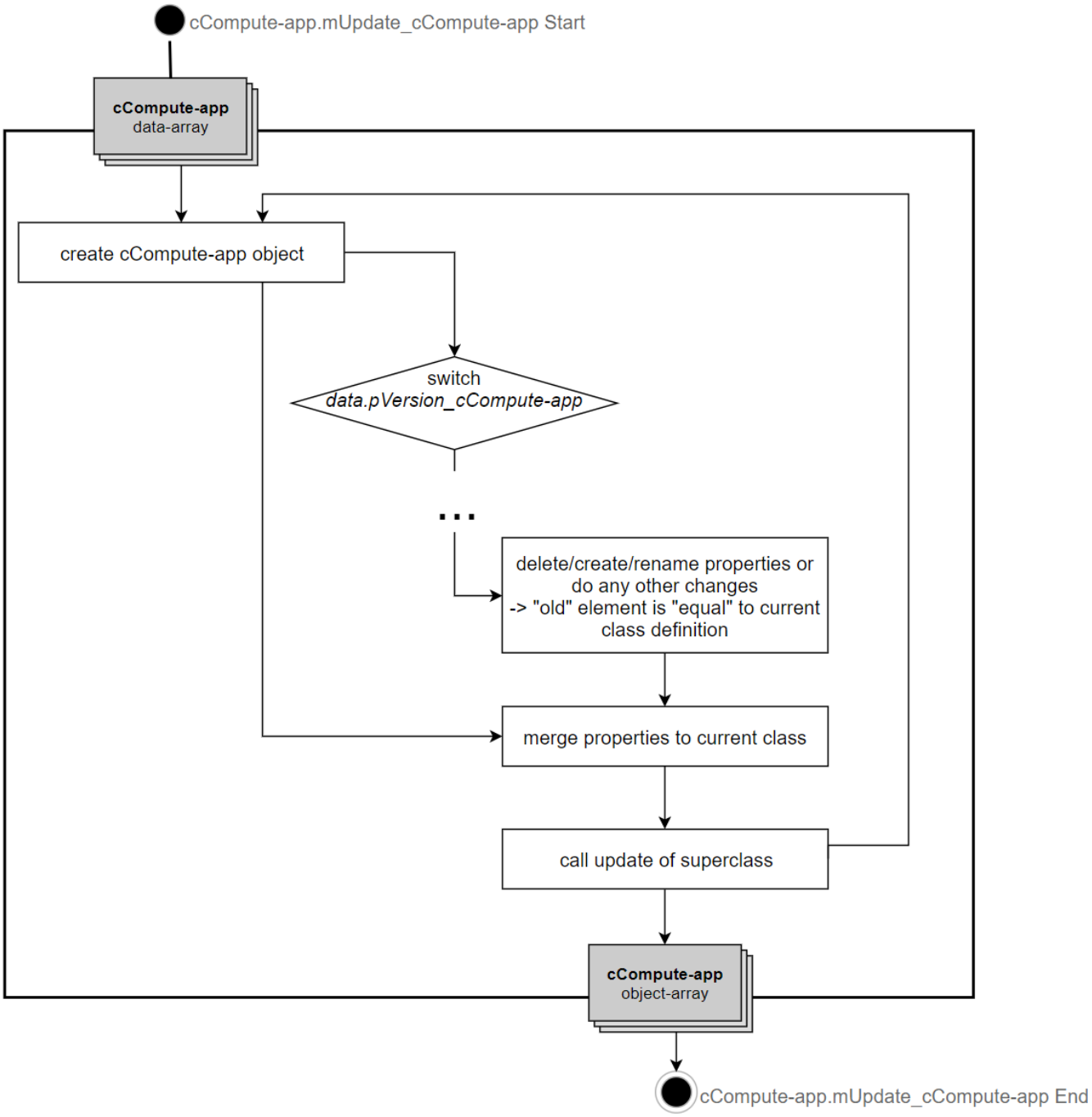
Background info to Session files:
A session file consists of the following data:

- *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



Properties, Methods

cControl

Properties

pApplication	- name of the application (Dicomflex)
pVersion_cControl	
pHandles	- to store all relavant handles (Gui, …)
pAcfg	- application config entries
pFcfg	- framework config entries
pActiveKeys	- currently pressed keyboard keys
pLineCoord	- stored coordinates of a mouse drag
pActiveMouse	- currently pressed mouse keys
pSaveDate	- timestamp of the last session save file
pVersions	- versions of other program parts
pVarious	

Methods (static)

mDrawContour (imgAxis, contCoord, contColor, varargin)

Methods

mSetVersionInfo(oCont, name, ver, updateFcn)	- modifies or adds entries to pVersion
mGetSaveFilePrefix(oCont)	
mGenerateHistData(oCont)	- gives back a data struct for undo storing
mMakeUndo(oCont, varargin)	- when crtl+z was pressed…
mGraphAxisButtonDown(oCont, a, hit, varargin)	- mouse press on graph
mImgAxisButtonDown(oCont, a, hit, varargin)	- mouse press on image
mMouseWheel(oCont, a, b, varargin)	- scrawling through slices
mKeyPress(oCont, a, key)	
mKeyRelease(oCont, a, key)	
mMenuCallback(oCont, fcn, varargin)	- catches most menu entry presses
mImageDisplayMode(oCont, b, varargin)	- menu bar callback
mSaveData(oCont, varargin)	- menu bar callback
mLoadData(oCont, varargin)	- menu bar callback
mCloneSoftware(oCont, varargin)	- menu bar callback
mTableCellEdit(oCont, hTab, select, varargin)	
mTableCellSelect(oCont, varargin)	- this method also triggers the GUI update
mUpdateTable(oCont)	- updates all table entries
mCreateZoomView	
mCloseZoomView	
mCreateMenu(oCont, s, parent)	- generates menu bar by
mSetGUI(oCont, varargin)	- resizing of GUI
mCreateGUI(oCont)	- create GUI elements
mLogging(oCont, action, varargin)	- logging of actions in pVarious
cControl(varargin)	- main calling method to create oCont object
mClose_oCont(oCont, varargin)	- quit program

Properties, Methods

cCompute

Properties

pVersion_cCompute	
oImgs	- will store all loaded raw images as cImage object
pDataName	- any kind of string describing the data (use for application specific stuff only)
pHistory	- store history --- not implemented
pStandardImgType	- an arbitary defined name for the standard image of the specific appliaction
pPatientName	
pSliceLocation	
pVarious	- used for application specific unspecific storage

Methods(static)

mGetBoundaryImageCoord(boundImg)	- get coord from bw img
mGetBoundMask(imgData, coord, varargin)	- get mask from xy coord array

Methods

mGetImgPathes(oComp, oCont)	- find all images on HDD
mInit_oComp(oComp, oCont)	- start new session or open existing
mUpdate_cControlVersionInfo	- calles oComp.mSetVersionInfo for all cCompute classes
mSliceSelected(oComp, oCont, old, new)	
mImageUpdate(oComp, oCont)	
mGraphUpdate(oComp, oCont)	
mTextUpdate(oComp, oCont)	
mShowHotkeyInfo(oComp, oCont)	- lists all hotkeys in an info screen
mMergeContours(oComp, oCont)	- merges drawn coords to existing bw mask
mCopyBound(oComp, oCont)	
mPasteBound(oComp, oCont)	
mContourTrackingONOFF(oComp, oCont)	
mGetImgOfType(oComp, type)	
mGetStandardImg(oComp)	
mSave_oComp(oComp, path, oCont)	
mShowFitInFitTool(oComp, oCont)	
mStartFitTool(oComp, oCont, cBound)	
mUpdateFitTool(oComp, oCont, cBound)	
mUseFitToolData(oComp, oCont, dFT, varargin)	
mUpdate_cCompute(oComp)	- update loaded data to new cCompute
cCompute(cCompArray)	- calling method to create oComp object

Properties, Methods

cCompute-app < cCompute

Properties(mandatory)
pVersion_cComputeApp

Methods(mandatory)
mInit_oCompApp(oComp, oImgs, oCont) - to fill properties of oComp array
mGetImg2Display(oComp, oCont)
mPostPlot(oComp, oCont)
mDrawGraph(oComp, oCont)
mGetTextBoxLines(oComp, oCont) - to be printed in text area
mTableEdit(oComp, select)
mKeyPress(oComp, oCont, key)
mKeyRelease(oComp, oCont, key)
mImgAxisButtonDown(oComp, oCont, hit)
mImgAxisButtonMotion(oComp, a, b, oCont)
mImgAxisButtonUp(oComp, a, b, oCont)
mUpdate_cComputeApp(oComp, data, saveDate) - update data to newest cComputeApp version
cComputeApp (cCompArray) - calling method to create oCompApp object

Properties, Methods

cImage

Properties

name	- name of image
date	- unused
datenum	- unused
path	- full path to the image file
nr	- image number
imgType	- string describing the image
data	- image data
pVersion_cImage	- version of cImage class

Methods

conv2RGB(img)	
dataResize(img, scale)	- change size of image
scale2(img, scaleMinMax)	- scale range of pixel values
update_cControlVersionInfo(img, d)	
cImage(path)	- calling method to create cImage object

Properties, Methods

cImageDcm < cImage

Properties

dicomInfo - header information of dicom image
pVersion_cImageDcm

Methods

readDicom(imgs) - reads an cImageDcm array from HDD
rescaleDicom(imgs) - rescale according to dicom header information
getVoxelVolume(imgs)
patientName(img) - determine best patient name
sliceLocation(img)
sliceThickness(img)
slicePosition(img)
cImageDcm(path, imgPathes, imgName) - calling method to create cImageDcm object

Properties, Methods

cBoundary (optional class)

Properties

name	- name of boundary
coord	- coordinates
various	

Methods

getBoundInd(Bounds, boundName)	- find the index of a bound with boundName
getBoundOfType(Bounds, boundName)	- give back the bound boundName
setBound(Bounds, boundIn)	- overwrite or create new element in Bounds
areaBound(bound)	- area in pixels fo bound
boundary	- calling method to create cBoundary object

Properties, Methods

cComputeApp (optional methods)

There exist several methods in already implemented applications that may be reused. Those are methods for boundary management and data fitting. See T1Mapper and FatSegment for those methods.

for fitting see cComputeT1Mapper -> % % % Fit Management % % %
for segmentations see cComputeFatSegment -> % % % Segmentaion Organisation % % %

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
     see Properties, Methods page above
   - 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 %-%-%
   At this point it may also be good to generate the applicationConfig in parallel.
8. Note:
   - cComputeApp will use the sliceLocations value of the dicom header to find images and sort them to oComp objects in mInit_oCompApp. In the case non dicom images are used, please remove that part accordingly.
   - cCompute_template.m 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

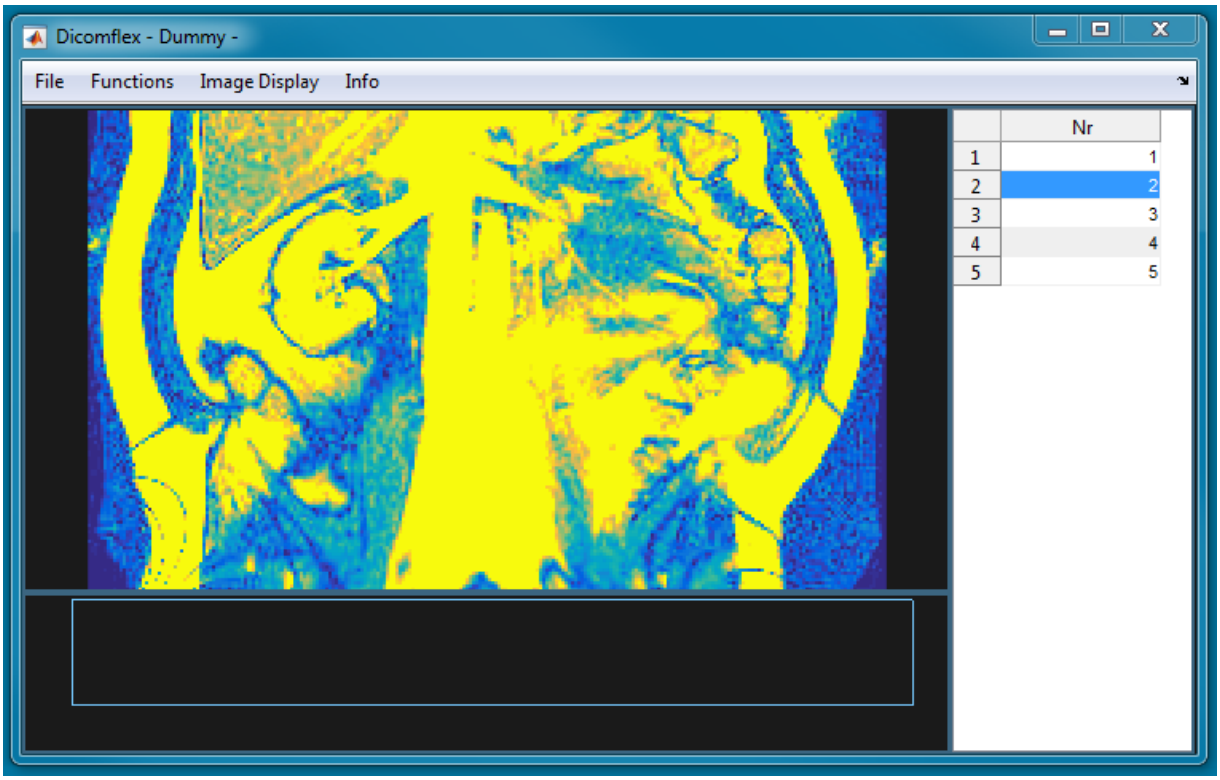
```
% % % isat data mode config % % %
%% general
savePath = uigetdir('C:\Users\StangeR\Dropbox\UniKlinik\MatlabSource\DicomFlex');   % where

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 'TlMapper'...

    case 'FatSegment'...
end

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

    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']));
```

Sample Application

Dummy

For the Dummy simple example there exits a folder Example/Dummy. It consists the code, an executable and sample images to be loaded with it. To get started to the following steps:

1. copy the cfg_application_*.json file in the same folder as the cControl.m file and make the cCompute*.m available for the interpreter.
2. type "cControl" in the Matlab command window
3. a list dialog should now appear. Choose “Dummy"!
4. a load dialog will appear. Please select the "DataFolder" folder
5. a question dialog will appear. You can freely choose to start a new session or use the existing one the program found in the folder.



You can either load the images by choosing “start new session” or load the existing dataset with the Dummy mode of the Dicomflex.

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.

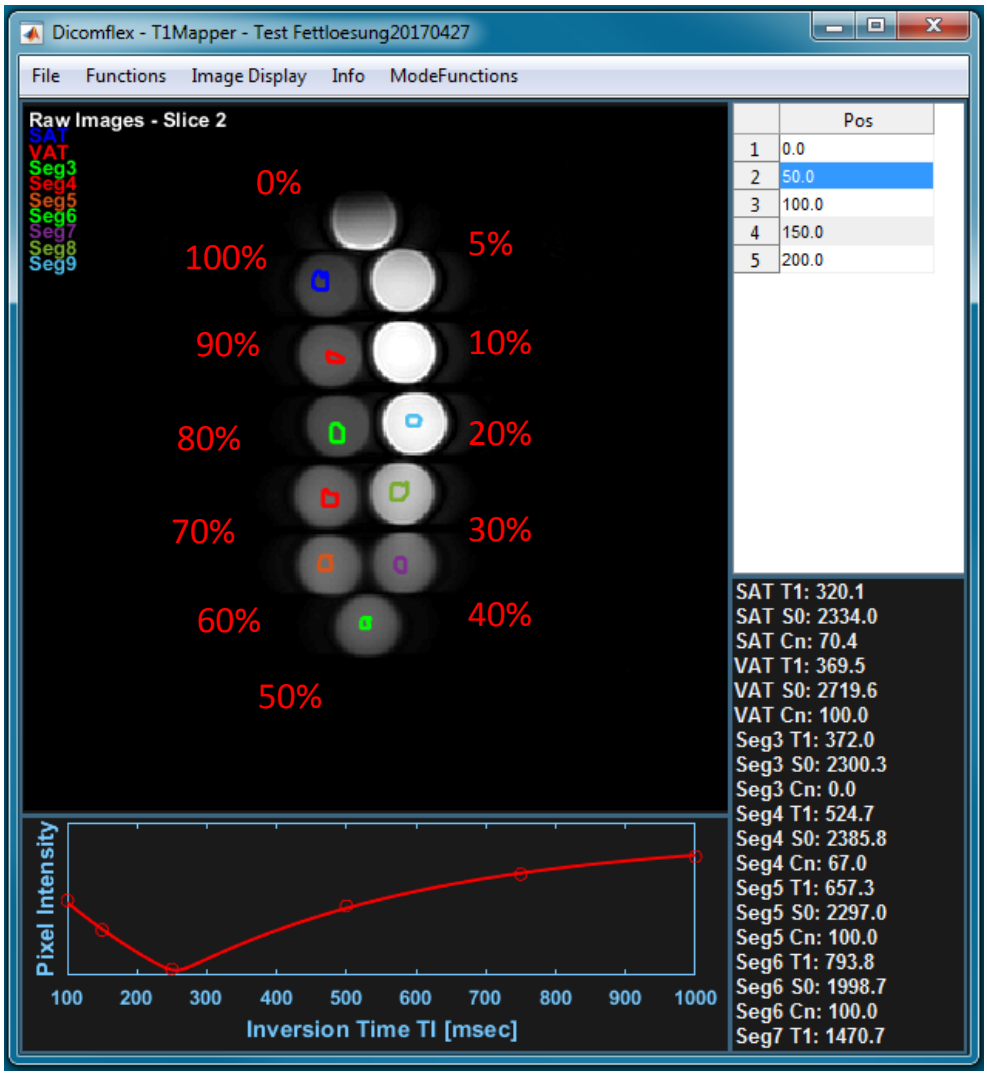
```
%% app specific values
menu = struct('path', {}, 'callback', ''); % empty menu
switch cfg.applicationName
    case 'T1Mapper'
        cfg.imageDisplayMode = 'Raw Images'; %%...%%
        %% external windows%%...%%
        %% Contour settings%%...%%
        %% Key associations%%...%%
        %% Gui 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.

Sample Application

T1Mapper

In the Git repository you will find a phantom MRI data set. It consits of tubes filled with fatty emulsions from 0% to 100 % of fat.



You can either load the images by choosing “start new session” or load the existing dataset with already computed T1 maps in the T1Mapper mode of the Dicomflex. Fitting T1 values only makes sense for fat fractions of 50% - 100% as the fitting in this mode is limited to T1 values from 180ms to 4000ms.

Unfortunately the dataset is does not give good T1 values as the phantoms where not designed for T1 fitting. The T1 values are dramatically corrupted due to MR-artifacts.
Due to patient ethics it was not possible to upload a sample dataset of a patient were fitting and mapping would work very nice!