Segment - Technical Manual

December 15, 2017

Software platform v2.1 R6070

MEDVISO AB
http://www.medviso.com
Griffelvägen 3
SE-224 67 Lund
Sweden
Tel: +46-76-183 6442

Contents

1 Regulatory status

Segment may be used for either investigational off label use or commercial purposes. Please see license terms which license form that apply to you. Users are also required to investigate the regulatory requirements pertinent to their country or location prior to using Segment. It is in the users responsibility to obey these statues, rules and regulations.

1.1 Commercial usage of Segment

FDA approved versions of Segment are identified with a labelling upon start up displaying licence details and the FDA 510(k) number K090833. If your version does not display this information your version is not FDA approved and you need to contact Medviso AB to receive a such license.

Please note that there are features that are not included in the FDA approval. These functions are marked in this User Manual that they are only for investigational use.

1.2 Indications for use

Segment is a software that analyzes DICOM-compliant cardiovascular images acquired from magnetic resonance (MR) scanners. Segment specifically analyzes the function of the heart and its major vessels using multi-slice, multi-frame and velocity encoded MR images. It provides clinically relevant and reproducible data for supporting the evaluation of the function of the chambers of the heart such as left and right ventricular volumes, ejection fractions, stroke volumes, peak ejection and filling rates, myocardial mass, regional wall thickness, fractional thickening and wall motion. It also provides quantitative data on blood flow and velocity in the arterial vessels and at the heart valves. Segment is tested on MR images acquired from both 1.5 T and 3.0 T MR scanners. The data produced by Segment is intended to be used to support qualified cardiologist, radiologist or other licensed professional healthcare practitioners for clinical decision making. **It is a support tool that provides relevant clinical data as a resource to the clin-**

ician and is not intended to be a source of medical advice or to determine or recommend a course of action or treatment for a patient.

1.3 Investigational purposes

None of the organizations/persons named in conjunction with the software can accept any product or other liability in connection with the use of this software for investigational purposes.

2 License terms

For general license terms of the usage of the software, see the User Manual.

Though parts of Segment is be released in source code form, this does not imply that standard open source rules do apply. Segment is a commercial product and is provided free of charge to the research community as a service and without any associated rights. Medviso AB does not give you any rights to do commercial derivative works of it. It does not give you the right to compile it to a distributable standalone form. See discussion on license terms in [1].

3 Acknowledgements

Even if this project started as a one man project, it has grown and it would never been possible without the help of many many people.

Financial support has been received from the Swedish Heart-Lung foundation, Swedish Research Council, local founds from Östergötland County, and Region of Scania.

I would like to acknowledge all the people that have put in feed back on usability and desired functionality, algorithm etc. Among others: Andreas Otto, Andreas Sigfridsson, Erik Bergvall, Erik Hedström, Henrik Haraldsson, Henrik Engblom, Håkan Arheden, Jan Engvall, Lars Wigström, Lisa Hård af Segerstad, Karin Markenroth Bloch, Marcus Carlsson, Martin Ugander, Mikael Kanski. Finally thank to you all Segment users in the research community that has inspired and contributed to the development.

Special thanks to code providers Erik Bergvall (core routines of strain analysis), Helen Soneson (strain analysis module, SPECT module, Image fusion module), Shruti Agarwal (refactory of strain analysis module), Jonatan Wulcan (Sectra Plugin module and general improvements), Johannes Töger (3D flow and volume tracking), Mårten Larsson (3D flow and kinetic energy).

Commercial development has been done by Jane Sjögren (improvements to general object segmentation, implementation of prototype based segmentation, CT functionality, and graphical seriesselector). General debugging and implementation of the new interpolated contours has been done by Johan Ugander and Erik Södervall. Report Module and general debugging have been performed by Nils Lundahl.

4 Conventions and Abbreviations

This chapter describes the typographic conventions and used abbreviations in this manual and in the program.

4.1 Typographic conventions

`A`	Key `A` at the keyboard.
`Ctrl-A`	Control key. Hold down `Ctrl` key and `A` simultaneously.
	Icon in toolbar.
`*.mat`	Filename extension.
`C:/Program`	Folder.
File	Menu, e.g. File menu.
File→Save As	Sub menu, e.g. under the File menu the item Save As is found.
Close	Push/Toggle button in the graphical user interface.
◎ Endocardium	Radiobutton in the graphical user interface.
☒ Single frame	Checkbox in the graphical user interface.

4.2 Trademarks

Below are some of the trademarks used in this manual.

- Segment is a trademark of Medviso AB.
- Segment DICOM Server is a trademark of Medviso AB.
- Sectra PACS is a trademark of Sectra Imtec AB, (`http://www.sectra.se`).
- Matlab is a trademark of the Mathworks Inc, (`http://www.mathworks.com`).

4.3 Abbreviations

2CH	Two chamber view
3CH	Three chamber view
4CH	Four chamber view

3D	Three Dimensional
3D+T	Time Resolved Three Dimensional
AA	Ascending Aorta
ASW	Anterior Septal Wall Thickness
ARD	Aortic Root Diameter
BPM	Beats per minute
BSA	Body Surface Area
CMR	Cardiac Magnetic Resonance
CO	Cardiac Output
CT	Computed Tomography
DA	Descending Aorta
DE-MRI	Delayed Enhancement MRI
ED	End diastole
EDD	End Diastolic Dimension
EDL	End Diastolic Length
EDV	End Diastolic Volume
EF	Ejection Fraction
ES	End systole
ESD	End Systolic Dimension
ESL	End Systolic Length
ESV	End Systolic Volume
FWHM	Full Width Half Maximum
GUI	Graphical User Interface
HR	Heart Rate
LGE	Late Gadolinium Enhancement
LV	Left Ventricle
LVM	Left Ventricle Mass
MaR	Myocardium at Risk
MO	Microvascular obstruction
MB	Mega Byte
MIP	Maximum Intensity Projection
MPR	Multiplanar Reconstruction
MR	Magnetic Resonance
MRI	Magnetic Resonance Imaging
PET	Photon Emission Tomography
PER	Peak Ejection Rate
PDW	Proton Density Weighted
PFR	Peak Filling Rate

PLW	Posterior Lateral Wall Thickness
PWV	Pulse Wave Velocity
ROI	Region Of Interest
RV	Right Ventricle
RVmaj	Right Ventricle Major Axis
RVmin	Right Ventricle Minor Axis
SPECT	Single Photon Emission Computed Tomography
SSFP	Steady State Free Precision
SV	Stroke volume
TOF	Time of Flight
VENC	Velocity Encoding limit

5 Document purpose

The purpose of this document is three-fold;

1. Work as a technical documentation of Segment for Medviso AB
2. Work as a technical documentation for researcher who wish to use the plug-in functionality of Segment as described in [1].
3. Work as a implementation documentation of Segment.

Therefore, all details described in this technical manual may not be applicable to all developers since parts of the source code is only accessable by Medviso AB.

6 Coding Conventions

This chapter describes the coding conventions that has been used when coding Segment. I have really tried to use a consistent naming rules to ensure that variables are given a logical name. However, there are occasions where historical reasons have lead to exceptions to the rules given below.

In general Segment originally used the global variable `NO` indicating the current image stack quite extensively. The use of `NO` is however not encouraged, and it is highly recommendable to in the function call indicate the current image stack. Work is ongoing to completly remove the global variable NO, but it takes time, please see wiki and relevant tickets on this matter.

6.1 Naming functions

Functions are always named in lower case letters only. The underscore letter is not used with the exception of:

- Callbacks are named as `functionname_Callback`.
- Functions for mouse motion callbacks are given names ending with `_Motion`.
- Some functions have helper functions that are only called from other functions with the same name. These functions are named `_helper`.
- Functions for mouse button up/down callbacks are given names ending with `_Buttonup` or `_ButtonDown`.

Note that functions should not be ended with an `end` (Matlabsupports both syntaxes). This is with the exception with objected oriented programming, see below.

6.2 Naming variables

The following rules apply:

- Global variables have all capital letters, i.e `DATA`.

- Field names of structures (and structure arrays) are given names starting with a capital letter. When two or more words are concatenated then the first letter of the second word also have a capital letter. For instance `DATA.CurrentTool`. Abbrevations are also given capital letters. For instance `SET(NO).TSize` where `T` stands for time.
- Field names with all capital letters corresponds to matrices. For instance `SET(NO).IM`, or `DATA.BALLOON`.
- The use of plural s is very limited and only used where there are limited numbers of values the field and it may be necessary to point out that there are more values than just a scalar. For instance `DATA.ViewPanels` is just instead of `DATA.ViewPanel` to point out there are (or may be) more than one view panel.

6.3 Name graphical handles

The graphical handles are named with all lower case letters and no underscore letters. Generally the names are rather long and exploratory. The following general naming conventions apply:

- Text edit boxes ends with `edit`
- Static text ends with `text`
- Axes objects ends with `axes`
- Pushbuttons ends with `pushbutton`
- Radiobuttons ends with `radiobutton`
- Checkboxes ends with `checkbox`
- Togglebuttons ends with `checkbox`

6.4 Object oriented programming

For new objected oriented functions the new syntax introduced in Matlab R2007b should be used. The old usage where the class definition is stored in a separate folder is obsoloted and existing code will be rewritten to new standard. Note that this new coding methods using `classdef` requires that an `end` statement is inserted in the end of the method code.

6.5 Packaging

To improve the file structure, Matlab's packaging feature is used. Files that are part of the same module are placed in a directory named with an initial '+' sign. Calling a function file `fcn_name.m` from a package `+package_name` is done by entering `package_name.fcn_name` (without the '+'). When adding packaged files to the compilation process, the full directory name must be included in the string. In the example above, this becomes `['+package_name' filesep 'fcn_name.m']`.

6.6 General coding conventions

The use of boolean variables in function calls should be avoided since this significantly decreases code readability. This is especially true in API-close functions. Instead it is suggested to write two functions with different names that performes the different tasks. This boolean coding convention is historically not followed, but should be followed for new code, and old code should sucessively be rewritten.

Temporarily variables should be avoided, and if they are used they should not have a scope that exceeds three lines of code. Also here this is something that historically is not always followed. This rule should be followed for new code and old code should sucessively be rewritten.

6.7 Units

The units of variables should be in SI-units, and the exact unit should also be documented in a comment on the same line as where the variable is initialized or calculated. For instance:

```
speed = length/time; %[m/s]
```

6.8 Graphical user interfaces

All new graphical user interfaces should be based on the class `mygui`. When doing so Segment automatically will remember the position of the user interface, and coding is considerably simplified so that one does not need to use

persistent variables to get application data. Please consult the documentation of `mygui` for further details. Graphical user interfaces that do not use `mygui` should be rewritten to use mygui.

6.9 Indentation and spacing

Matlab standard indentation system is to be used, but with using the tab size to two spaces instead of the default four. Spacing and use of parenthesis is to be used to enhance readability. For instance

- `A = 12;` instead of `A=12;`.
- `if (a>b) || (d<e)` instead of `if a>b||d<e`

MLINT syntax guidelines should be followed and ideally when syntax hints are overrided, they should be motivated. For instance when MLINT reports; Warning data seems to grow inside a loop and that you should consider to preallocate for speed. Then before inserting a pragma to remove the warning you should make a note on why you did not preallocate (which usually is that the routine is not time critical, but is should generally be documented). Another example is when MLINT warns that the variable is unassigned, but it is assigned by loading a `.mat` file, then this should be commented.

The use of logical short cutting operands `|| and &&` is strongly recommended.

6.10 Documenting code

The source code should be well documented so that new programmers easily can understand the code. Note, especially that the part of the code that was trickiest to write also deserves the most comments. Functions should be written as:

```
%-----------------------
%function out = foo(bar)
%-----------------------
%Explanatory help text of the function.
%Help text may span multiple lines.
```

```
Your code begins here
```

It is very important to follow this coding standard since part of this documentation is automatically generated from the source code.

7 Coordinate Systems

This chapter specifies the coordinate systems used in Segment.

7.1 Introduction

DICOM specifies coordinates in the RL (right/left), AP (Anterior/Posterior) and FH (Feet/Head) coordinate system. Internally Segment uses a coordinate system (x,y,z) that is relative to the respective image stack. In other words for each image stack the relations between (RL,AP,FH) and (x,y,z) is different.

Throughout the program, the x dimension refers to the first dimension along the arrays (i.e the **rows** in Matlab). The upper left pixel in the images displayed on the screen have the coordinate (1,1). The unit used is in pixels. In the normal image coordinate system this means what is usually meant as the y coordinate if one thinks of coordinate systems learned in elementary school. For instance plotting commands therefore generally takes the form:

```
plot(SET(NO).EndoY(:,3,1),SET(NO).EndoX(:,3,1));
```

This will plot the contour of the endocardium of the third time frame and the first slice of the current image stack. The upper left most slice in the montage view is slice one. Furthermore it is assumed that the slices should be ordered so that going from the first slice to the last would be going from the base to the apex of the heart.

The relations between the (x,y,z) and (RL,AP,FH) coordinate systems is given by the fields `ImagePosition` and `ImageOrientation`. Below are two examples given.

The (RL,AP,FH) coordinates (center) of the voxel(s) `SET(1).IM(1,1,:,1)` are given by `SET(1).ImagePosition`.

The (RL,AP,FH) coordinates (center) of the voxel `SET(1).IM(2,3,4)` are

given by

```
zdir = cross(SET(1).ImageOrientation(1:3),SET(1).ImageOrientation(4:6))';
pos = SET(1).ImagePosition(:)'+...
        (2-1)*SET(1).ResolutionX*SET(1).ImageOrientation(4:6)'+...
        (3-1)*SET(1).ResolutionY*SET(1).ImageOrientation(1:3)'-...
        (4-1)*(SET(1).SliceThickness+SET(1).SliceGap)*zdir;
```

Coordinated conversions can be performed using the functions `xyz2rlapfh`, and `rlapfh2xyz` in `calcfunctions.m`.

8 Running Segment from Matlab

You need to have Matlab R2011a or later to run Segment. Running Segment from Matlab prompt is just as easy as running it as a stand-alone application. Note that necessary mex files have been compiled for Linux (64 bit), and Windows (both 64 and 32 bit). When Matlab has started, simply type:

```
>> segment
```

To get acess to the internal variable direct from the Matlab prompt, simply type:

```
>> global DATA SET NO
```

To get the x-coordinates of the endocardial segmentation in time frame 3 and slice 4, simply type:

```
>> x = SET(NO).EndoX(:,3,4)
```

To plot the segmentation in the current slice in another window, type:

```
>> figure(22);plot(SET(NO).EndoY(:,SET(NO).CurrentTimeFrame,SET(NO).CurrentSlice),SET
>> axis image off;
```

9 Overview of Segment

Segment is written in Matlab, and is a fairly large software project. Currently it contains of around 130 000 lines of `m-code` distributed more than 200 files, and 2815 subfunctions, and 69 gui's. There are 72 files `c-code` with about 12 000 lines. There are 22 supporting `.mat` files, and 32 auxillary files. In addition there are 12 000 lines in around 40 files and 300 subfunctions of internally support lines of code keeping track on distribution, compilation process, packaging, documentation generation etc.

This Technical Manual aims to describe how to write own plug-ins that interface with the software and give some implementation details for the interested reader. A good technical overview of the Segment project is also given in [1].

Before reading this user manual the reader should be well acquainted with Segment and programming in Matlab. This manual is **not** a manual how to use Segment, it's intention is only to give technical details on how things are implemented in Segment. Some implementational details are also given in the user manual.

10 Architectural Design

The entire Segment block can be divided into ten function superunits. Some of these function superunits are coded in a single `m-file`, whereas some can be further divided into function units (and separate `m-files`). An overview on how they communicate and relate to each other is shown in Figure 1. Overviews of the `Report` and `File` superunits are shown in Figure 4 and Figure 5, respectively.

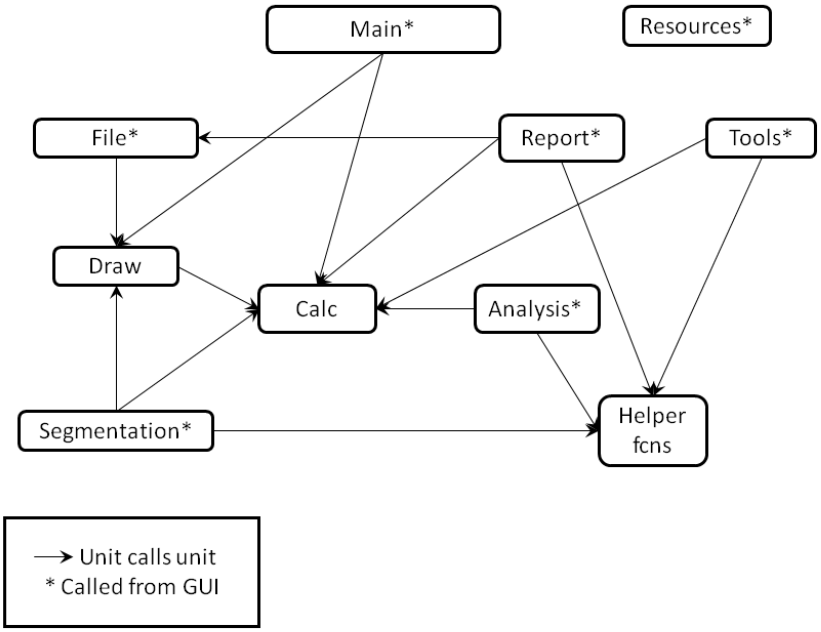

Figure 1: Overview of the function superunits, which constitute the main building blocks of Segment, and transaction analysis.

10.1 List of superunits

This list provides references to sections of chapter 15 where the superunits are documented and further divided into primitive units. It also references ID's of requirements that are fulfilled by the unit.

- Main superunit 15.1: RA14, RA15.
- File superunit 15.9: RB1, RB2, RB3, RC1, RC2, RC3, RE1, RE2, RG1, RG2, RG3, RI1.
- Draw superunit 15.2: RA4, RA5.
- Calc superunit 15.3.
- Resources superunit 15.7.
- Segmentation superunit 15.6: RA6, RA7, RA8, RA9, RA10, RA13, RE3, RA17, RA19, RA20.
- Tools superunit 15.8.
- Analysis superunit 15.10: RA12, RA21.
- Report superunit 15.5: RA16, RA17, RB4.
- Helper functions superunit 15.4.

10.2 OTS software and SOUP elements

10.2.1 OTS software

The Off-the-shelf software (OTS) software used by Segment is Matlab Compiler Runtime (MCR), which has the following specifications:

- **Manufacturer:** Mathworks
- **Version:** R2014a
- **Hardware specifications:** MCR takes about 450 MB harddisk space.
- **Software specifications:** Opearting system Windows 2000 or later (both 32 bit and 64 bit are supported).
- **Intended use:** MCR is the runtime environment for the Matlab development language. Segment is precompiled by using Matlab Compiler. The compiler converts the code to encrypted pre-parsed code that is interpreted by MCR which is installed together with each copy of Segment.

- **Features:** Execute Matlab files without an installed version of Matlab on the computer.
- **Justification for using the OTS software:** MCR enables the execution of Matlab files on computers without an installed version of Matlab.
- **Documentation to end user:** Installation instructions for MCR are included in the installation instructions for Segment. Technical documentation (this section).
- **Method to ensure appropriate use:** MCR will be installed by the administrator trained by Medviso AB. The administrator is responsible for the update of MCR when indicated by Medviso AB.
- **Method to ensure the OTS software works:** The verification of the MCR are included in the Installation Process Verification.
- **Control the OTS software:** The MCR is only upgraded when the development enviroment Matlab is upgraded. Matlab is only upgraded when it is considered adventageous for the software development, and typically it is 2-4 years between upgrades. After an upgrade, the MCR is reverified. The MCR installer is provided at the same server as the software installation files are.

10.2.2 SOUP elements

The software of unknown provenance (SOUP) element used by Segment is the DCMTK library, the Non-Sucking Service Manager (NSSM) and the Bar-Code Generator GS1-128.

DCMTK library

- **Manufacturer:** OFFIS - Institute for Information Technology
- **Version:** 3.5.4
- **Hardware specifications:** Not specified
- **Software specifications:** Opearting system Windows 2000 or later (both 32 bit and 64 bit are supported).
- **Intended use:** Handling DICOM files for the hospital PACS system.
- **Features:**

  - Tools for recieving DICOM files from the hospital PACS system
  - Tools to send DICOM files to the hospital PACS system
  - Tools to search in the hospital PACS system
  - Tools to convert between different DICOM formats

- **Justification for using the SOUP element:** A program was needed for sending and recieving DICOM files to the hospital PACS connection. OFFIS DCMTK toolkit is vendor-independent and is already established and used by hospitals.
- **Documentation to end user:** Technical documentation (this section).
- **Method to ensure appropriate use:** The DCMTK library are included in the general software package and do not need any processing by the end user.
- **Method to ensure the SOUP element works:** The verification of the DCMTK are included in the software testing.
- **Control the SOUP element:** The software comes in complete source code and is available as open source software. There is an open discussion forum about the software but no support agreement for the software. The DCMTK library is only upgraded to access bug fixes or when new features useful in Segment are released.

NSSM service

- **Manufacturer:** Iain Patterson
- **Version:** 2.24
- **Hardware specifications:** Not specified
- **Software specifications:** Opearting system Windows.
- **Intended use:** NSSM monitors the running service and will restart it if it dies.
- **Features:**

  - Monitoring programs
  - Re-start program if they crash
  - Service helper which doesn't suck

- **Justification for using the SOUP element:** A program was needed for monitor and re-start our Segment Server to ensure it is restarted if is is closed or crashed. NSSM is an open source software fulfilling our needs.
- **Documentation to end user:** Technical documentation (this section).
- **Method to ensure appropriate use:** The NSSM program is included in the general software package and do not need any processing by the end user.
- **Method to ensure the SOUP element works:** The verification of NSSM are included in the software testing.
- **Control the SOUP element:** The software comes in complete source code and is available as open source software. There is a bug list online but no support agreement for the software. The NSSM version included in our software packages is only upgraded to access bug fixes or when new features useful in Segment are released.

BarCode Generator GS1-128

- **Manufacturer:** Pedro Villena
- **Version:** 1.3
- **Hardware specifications:** Not specified
- **Software specifications:** Matlab 2012b
- **Intended use:** Generate a barcode image (BMP file) using GS1-128 symbology specifications.
- **Features:**
  - Generate an image of a barcode
  - Required input is the code in digits
  - Using GS1-128 symbology specifications for the generation
- **Justification for using the SOUP element:** A program was needed for creating a bar-code according to FDA requirement that the device need to have the device identification number as a bar-code in the labelling. BarCode Generator GS1-128 is an open source software found at Matlabs community fulfilling our needs.

- **Documentation to end user:** Technical documentation (this section).
- **Method to ensure appropriate use:** The BarCode Generator GS1-128 program is included in the general software package and do not need any processing by the end user.
- **Method to ensure the SOUP element works:** The verification of the genrated bar-code is done by an independent barcode reader. The result is stored together with the BarCode Generator and the result verifies the same code as sent into the BarCode Generator.
- **Control the SOUP element:** The software comes in complete source code and is available as open source software. There is a update list online but no support agreement for the software. The BarCode Generator GS1-128 version included in our software packages is only upgraded to access bug fixes or when new features useful in Segment are released.

10.3 Protected code sections

Some of the building blocks contains propriety algorithms and will only be made available as `p-code`, i.e platform independent precompiled Matlab code. The blocks that are be hidden are: `lv.m`, `rv.m`, `viability.m`, `flow.m`, `pacs.m`, `patientdatabase`. These functions will not be described in great detail in this document, for documentation of these files, please see internal documents at Medviso AB.

10.4 Global variables as data communication

Segment uses two global variables to store the state of image data, graphical handles. These two variable are `SET`, and `NO`. The structure of the variable `SET` is documented in Chapter 13. The variable `DATA` can be seen as a global variable, but is a global object. This allows for overloading the entire user interface of Segment. The `DATA` object contains global state of the program and all main graphical handles, and is documented in Chapter 12.

The global variable `SET` contains all image data, segmentation, ROI's, measurements, annotation points, etc. The variable `NO` is simply a scalar and points to the current image stack. It is highly recommended not to use the

global NO, but rather include this in the call.

The reason for having global variables instead of a completely functional programming approach is that:

- Simplicity for writing plug-ins.
- Execution speed. Using this approach there are way less variables/pointers that needs to be passed to the functions.

The drawback with using global variable is that the variable should at all times only reflect valid global states of the data. Therefore it is very important that modifications of the global variables are done with care.

11 Class Hierarchy Overview

This chapter documents the class hierarchy of Segment. The implementation of Segment is only partially performed using object oriented techniques, and the main reason for this is that object oriented Matlab was not possible to compile at the time when the oldest part of the code was written.

11.1 General class hierarchy

The general class hierarchy that partially constitutes Segment consists of a main GUI superclass `maingui` and its software specific subclass, such as the cardiac research edition Segment subclass `segmentgui`, the cardiac MR clinical edition Segment CMR subclass `segmentmrgui` and the cardiac CT clinical edition Segment CT subclass `segmentctgui`. An overview of this hierarchy is provided in Figure 2.

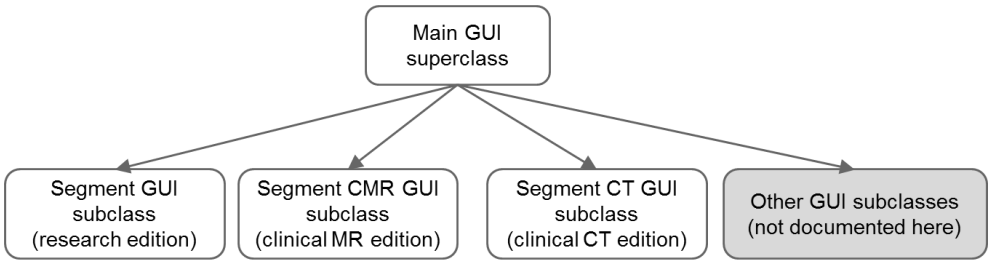

Figure 2: Overview of the general GUI class hierarchy.

11.2 Compilation class hierarchy

The class hierarchy for compilation consists of a superclass `makeitclass` and its subclasses. For compilations of Segment-like software, the subclass `segmentmakeitclass` is used. This subclass is used for the compilation of the standalone research edition of Segment, and is further divided into subclasses such as `segmentcmrmakeitclass` for compilation of cardiac MR clinical edition Segment CMR, `segmentctmakeitclass` for compilation of cardiac CT clinical edition Segment CT, and `segmentmakematlabclass` for packaging

the source code cardiac research edition. An overview is provided in Figure 3.

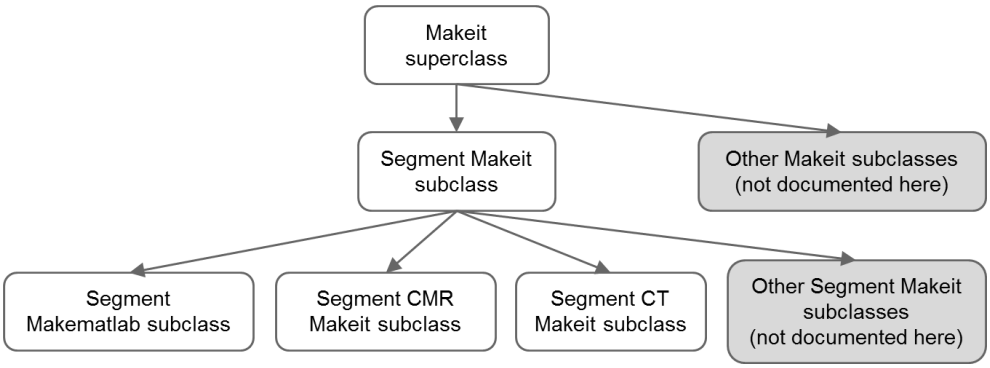

Figure 3: Overview of the class hierarchy of the compilation process.

12 DATA Object

This global object is an instance to the class `segmentgui`. It contains methods and properties that are used to store properties and access the graphical handles, preferences, etc. It is an instance of one of the subclasses of the `maingui` class and contains the following properties:

`LogFile`	Location of the log file keeping track of the current session.
`ProgramVersion`	A string containing the version number. This string is initiated at startup.
`ProgramName`	Name of the program in use, e.g. `Segment`.
`fig`	Handle to the main GUI figure.
`BlockingFigs`	A list of open figures that contains GUI's that are sensitive to changing the current image stack `NO`.
`imagefig`	Handle to the figure where the images are plotted. Currently this is same as `fig`, but are reserved for future use.
`DataLoaded`	True if image data is present.
`Silent`	True if no graphical output should occur. This is usefull when writing plugins that perform batch processes.
`Handles`	Struct with all graphical handles in the main GUI.
`InteractionLock`	When set to true, new calculation initiated from callbacks are prohibited. This may be obsoleted in the future and replaced with Matlab's queueing properties.

`Undo`	Struct containing undo information.
`UndoN`	Scalar that contains a pointer in the undo history list.
`LastSaved`	Time stamp when the image stack(s) last were saved. This is used for the auto-save functionality.
`Volrend`	Reserved for future use with the future volume rendering module.
`ViewPanels`	A vector pointing to image stacks. If 2 x 2 panels are shown on the screen, then the length of `ViewPanels` is 4. For panels that does not contain any image stack the corresponding pointer is set to zero.
`ViewPanelsType`	A cell array where each element contains the view mode for each image panel. Allowed view modes are: • mode `one` shows the image stack as one slice. • mode `montage` shows all slices of the image stack arranged in a matrix style. • mode `mmodetemporal` shows the temporal part of a mmode image display. • mode `mmodespatial` shows the spatial part of a mmode image display. The best way of changing viewing mode is to call the function `viewimage_Callback`.
`ViewPanelsMatrix`	A cell array where each element i a two element vector containing the number of rows and columns for a montage view in the corresponding panel.

`ViewIM`	A cell array with the length same as `ViewPanels`. Each element contains a `uint8` array of size N x M x T where T is the number of time frames, N and M are arbitrary numbers that depends on `ViewPanelsType`. This is used for graphical output and contains mapped intensities.
`ViewMatrix`	A two element vector with the size of the image panels (n x m).
`LastView`	Save view settings for the last panel view. Used upon double-clicking to go from the current panel view to the last panel view.
`CurrentPanel`	Points to the current image panel. This number needs to be in the range $[1..N]$, where N is the number of valid panels.

`CurrentTool` String containing the current tool. Valid tools are:

- `point`
- `measure`
- `select`
- `dragendo`
- `dragepi`
- `drawendo`
- `drawepi`
- `drawrvendo`
- `drawrvepi`
- `drawroi`
- `drawscar`
- `drawrubber`
- `drawrubberpen`
- `endopin`
- `epipin`
- `contrast`
- `crop`

`CurrentTheme` Indicates which set of segmentation tools is currently on display to the user.

`Tools` Structure of handles to tools available in the toolbar.

`SegmentFolder` Location of the folder from which Segmentis loaded.

`imagedescriptionfile` Text document for setting image description parameters.

`manualfile`	Location of the Reference manual.
`GUI`	Structure of `mygui` objects for windows that use this class.
`GUISettings`	GUI settings
`GUIPositions`	Struct array to store the positions of GUI's that have been in use.
`AllowInt16`	Option whether or not to allow `int16` data.
`Pref`	Structure with preferences. For further details, please see Section 15.7.2.
`PrefHandles`	Handles for Preferences window.
`PrefHandlesAdvanced`	Handles for Advanced Preferences window.
`PrefHandlesPacs`	Handles for PACS Preferences window.
`Preview`	Struct storing preview data. This struct is intensely used by the file open GUI. Generally the data is store in this struct and thereafter copied to the SET variable upon completion of the loading process. Therefore this struct also contains information on the last loaded/viewed image stack.
`StartTime`	Start time for video playback.
`Run`	True if a movie is playing.
`Record`	True if recording is active in the movie recorder GUI.
`LastPointer`	Type of the pointer displayed.
`LastPointerShapeCData`	Cdata of shape of the pointer displayed.

`VisibleThumbnails` Array of indices of thumbnails visible in main GUI.

`Icons` Structure of images used as icons in toolbars and icon bars.

`ImageTypes` A cell array contains the different precode image types in Segment. They are:

- `General`
- `Perfusion Rest`
- `Perfusion Stress`
- `Strain FFE`
- `Strain TFE`
- `Late enhancement`
- `Cine`
- `Scout`
- `Qflow`
- `T2Stir`
- `T1BB`

The image types are used among others to find what image stack is what for automated batch processing of files. This list is subject to future changes.

`ImageViewPlanes` A cell array contains the different precode image view planes in Segment. They are:

- `2CH`
- `3CH`
- `4CH`
- `Sagittal`
- `Coronal`
- `Frontal`
- `Transversal`
- `Short-axis`
- `RVOT`
- `Aorta`
- `Pulmonalis`
- `Vena cava inferior`
- `Unspecified`

The image types are used among others to find what image stack is what for automated batch processing of files. This list is subject to future changes.

`ImagingTechniques` Short names of the parameter files (`*.par`). This variable is created a startup. Therefore, Segment needs to be restarted before new parameter files can be used from the menu. However, the files themselves are not cached so the content in the files can be changed without restarting Segment.

`ImagingTechniquesFullNames` Full names / titles of the parameter files. See above for details.

ThisFrameOnly	True if changes are applied only to the current timeframe. Example of functionality is segmentation, clearing segmentation etc. See details in the Reference Manual.
StartFrame	First frame played in a movie. Initiated by SET(NO).CurrentTimeFrame, but needed for calculation what frame to display when playing a movie. See also DATA.StartTime.
SegIntersection	Seg intersection
IntersectionIM	Intersection image
ShowIntersectionMask	Show intersection mask
BalloonLevel	Used to determine if the field DATA.BALLOON needs to be recalculated. Stores the parameter that was used last time to calculate DATA.BALLOON.
BALLOON	A matrix of the same size as SET(NO).IM containing the radial balloon force. For further details, see [2, 3].
EndoBalloonForce	Used in the automatic LV segmentation for MR images.
EpiBalloonForce	Used in the automatic LV segmentation for MR images.
LevelSet	Helper structure to SET(NO).LevelSet. DATA.LevelSet stores parameters that are pertinent to the graphical user interface that does not need to be stored in SET(NO).LevelSet.
DATASETPREVIEW	A 64 x 64*N array containing the preview thumbnails, where N is the number of image stacks.

`EndoEDGE0`	Edge image of the endocardium in the first direction. For further details, see [2, 3].
`ENDOEDGE1`	See above.
`ENDOEDGE2`	See above.
`ENDOEDGE3`	See above.
`EpiEDGE0`	Edge image of the endocardium in the first direction. For further details, see [2, 3].
`EpiEDGE1`	See above.
`EpiEDGE2`	See above.
`EpiEDGE3`	See above.
`EndoEdgeDetected`	True if the edge images of the endocardium have been calculated.
`EpiEdgeDetected`	True if the edge images of the epicardium have been calculated.
`MovieFrame`	Temporary variable used to store one frame of a movie when recording a movie using the movie recorder in Segment.
`NumPoints`	Number of points along the endocardium, epicardium and ROI's. This default set to 80. This variable should be possible to change to get more points along a contour, but this have not been verified or tested. Note that changing this variable will most probably cause version incompabilities with `.seg` and `.mat` files.

BpInt	Intensity in the blood pool of the image stack SET(NO). This is used to avoid unnecessary recalculations of DATA.BALLOON.
MInt	Intensity of the myocardium of the current image stack. See above.
Pin	Keeps track of pins.
Measure...	Contains measurement data such as offset and coordinates of points.
Cursor...	Contains cursor data such as location and offsets.
NeedToSave	True if there are changes that need to be saved.
Testing	Used by maketest to tell a program when it is being tested.
RecordMacro	Used by macro_helper to tell when a macro is being recorded.
Macro	Used by macro_helper to keep track of the macros in a list.
MacroN	Used by macro_helper to keep track of the position in the Macro list.
Buffer	Contains pending actions used for macros and testing.
CineTimer	Timer used in cinewindow.
contourp	Used in the automatic LV segmentation for MR images.
contourbimage	Used in the automatic LV segmentation for MR images.

`DynamicPACS`	Stores PACS details specified through API by an external program.

13 SET Variable

This global variable is probably the most important variable/structure since it contains all image data and all measurements.

The variable is a struct array. As an example, `SET(2)` refers to the second image stack. `SET(2).CurrentTimeFrame` refers to the field that contains the current time frame of the second image stack. The function `loadfieldhelper` is used to control backwards compability in the `SET` struct. When adding new fields it is essential to also update this function.

Below a list of all fields are given.

`AccessionNumber`	DICOM tag Accession Number.
`AcquistionTime`	DICOM tag AcquisitionTime
`AutoLongaxis`	True if the amount of long-axis compensation should be automatically calculated. See Reference Manual for details.
`BeatTime`	Controls how fast one heart beat is played when playing images as a movie. Originally set to 60/HeartRate. May be adjusted with the slower/faster icons.
`CenterX`	Position of the center mark (+).
`CenterY`	Position of the center mark (+).
`Children`	List of numbers of image stacks that are children of (derived from) this stack.
`Colormap`	This field contains a colormap (256 x 3) vector. When the image is grayscale it is an empty field.

`CurrentSlice`	This field contains the current slice of the image volume.
`CurrentTimeFrame`	This field contains the current time frame.
`Cyclic`	True for image stacks that are cyclic, i.e covers a complete heart cycle. This flag is used by the segmentation engine when delineating the left ventricle.
`DICOMImageType`	This field contains the ImageType from the DICOM files.
`EDT`	Left ventricle end diastolic time.
`EDV`	Left ventricle end diastolic volume.
`EF`	Left ventricle ejection fraction.
`EPV`	Volume inside the epicardial volume of the left ventricle.
`EST`	Left ventricle end systolic time.
`ESV`	Left ventricle end systolic volume.
`EchoTime`	Echo time in MRI pulse sequence. Read from DICOM tags if presented otherwise set to zero. Given in milliseconds.
`EndAnalysis`	End time of analysis, used in flow quantification. See also `StartAnalysis`. For further details, see Reference Manual for details. Default value is same as `TSize`.

`Endslice`	This field contains the last slice in the set of selected slices. The selected slices is then given by: `ind = SET(NO).StartSlice:SET(NO).EndSlice;`. The field may be an empty array when no slices are selected.
`EndoCenter`	True if center of endocardium is used as center of the left ventricle when doing regional wall motion analysis. If false then the center of the epicardial volume is used. See Reference Manual for further details.
`EndoDraged`	A logical array containing a 1 where the left ventricle endocardial contour have been dragged.
`EndoInterpX`	Interpolation points for the endocardium.
`EndoInterpXView`	Interpolation points for the endocardium.
`EndoPinX`	X-coordinates of endocardial pins. Stored as a cell-array of vectors of doubles with the size T x Z, where T is the number of time frames, and Z is the number of slices. For more details about pins, see the Reference Manual.
`EndoPinY`	Y-coordinates of endocardial pins.
`EndoPinXView`	Compact representation for endocardial pins. Stored as a cell vector with the T elements containing vectors.
`EndoPinYView`	See above.

`EndoX`	X-coordinates of the left ventricle endocardium. Applied values are either `[]`, or an array with size N x T x Z, where N is the number of points along the contour (80, but technically `DATA.NumPoints`, T is the number of timeframes, and Z is the number of slices.
`EndoY`	Y-coordinates of the endocardium.
`EndoXView`	A compact representation used for drawing the endocardial contour in montage view. Stored as a double array with the size $((N+1)*Z)$ x T, where N is the number of points along the contour (`DATA.NumPoints`), T is the number of timeframes, and Z is the number of slices. It is the same as `EndoX` but each slice offseted and packed separated with a `NaN` so that drawing of the endocardial contours can be done with a single command.
`EndoYView`	The correspondence for the Y-coordinates to `EndoXView`.
`EpiDraged`	Same as above, but for the epicardial contour.
`EpiInterpX`	Epicardial interpolation points.
`EpiInterpXView`	Epicardial interpolation points.
`EpiPinX`	Corresponding to `EndoPinX`.
`EpiPinXView`	Corresponding to `EndoPinXView`.
`EpiPinY`	Corresponding to `EndoPinY`.
`EpiPinYView`	Corresponding to `EndoPinYView`.
`EpiX`	Epicardial contour, otherwise same as `EndoX`.

`EpiXView`	Corresponding to `EndoXView`.
`EpiY`	See above.
`EpiYView`	Corresponding to `EpiXView`.
`FileName`	Filename of the file storing this image stack.
`FlipAngle`	Flip Angle in MRI pulse sequence. Read from DICOM tags if presented otherwise set to zero. Given in degrees.

`Flow`

This struct contains information regarding how different image stacks are related to store flow information. Generally all information that are common for all the coupled image stacks are only stored in the magnitude image stack to avoid data redundancy. One example of this is storage of ROI's that are only performed in the magnitude image stack. The `Flow` struct contains the following fields:

- `Angio` contains a reference to which image stack that contains the angio imag. This angio image is essentially the absolute value of the velocity times the image magnitude and could be useful for vessel identification. Normally this field is set to an empty vector as the angio image is seldom constructed.
- `MagnitudeNo` contains reference to which image stack contains the magnitude information. For the image stack containing the magnitude information this points to itself.
- `PhaseNo` contains reference to the image stack containing the through plane flow information. Naming is somewhat inconsequent, by kept for legacy reasons. A better name would be `PhaseZ`.
- `PhaseX` contains reference to the image stack containing the flow in the X direction (Segment coordinate system).
- `PhaseY` contains reference to the image stack containing the flow in the Y direction (Segment coordinate system).

Then there are also a few fields that are optional and only available when eddy current compensation has been performed. These fields are:

- `PhaseCorr` contains the phase offset for the current direction (i.e if the current image stack is velocity/phase in the X direction), then `PhaseCorr` contains phase offset in the X direction, and so fourth. This field is always present, and if not applicable only an empty matrix is stored.
- `PhaseCorrPercentiles`, level of which percentiles to include in the detection of the static tissue. See documentation on the eddy current compensation for further details.
- `PhaseCorrMethod` contains the selected method for the eddy current compensation.
- `PhaseCorrTimeResolved` true if the eddy current compensation is time resolved. Default value is false. If time resolved, then `PhaseCorr` has the same image dimensions as `IM` otherwise the third image dimension is one.
- `PhaseCorrStaticTissueRois` true if regions where static tissue should be taken from drawn ROIs instead of estimated from the image. Default value is false.
- `VelMag` contains a reference to the image stack that contains a velocity magnitude stack (absolute value of the velocity). Normally this field is set to an empty vector as the velocity magnitude is seldom constructed.

`Fusion` Struct reserved for Fusion Module.

`GEVENCSCALE`	Special tag read from DICOM for GE scanners.
`HeartRate`	Hear rate of the image stack. Note that different image stacks may have different heart rates. Unit is beats per minute. `HeartRate` is used to calculate cardiac output.
`IM`	This field contains the image data stored as a 4D *single* array. The order of the dimensions are X x Y x T x Z. For more details on coordinate system conventions, see Section 7. The image data should lie between 0..1. For more details on image scaling see the fields `IntensityScaling` and `IntensityOffset`.
`ImageOrientation`	Same as DICOM tag `ImageOrientation`. If not presented then set to `1 0 0 0 1 0`. The three first numbers contains the normalized vector of Segments Y-direction given in the patient/table coordinate system. The last three numbers contains the direction of the X-direction. For further details on coordinate systems see Section 7.
`ImagePosition`	Same as DICOM tag `ImagePosition`. If not presented then set to `0 0 0`. Contains position in mm in 3D of the upper left pixel in the top slice in the image stack. For further details on coordinate systems see Section 7.

`ImageType`	Type of images that the image stack depicts. The field is used when identifying what image stacks to use for different analysis. If not set at loading then Segmenttries to figure these details out by looking at what operations have been performed on what image stacks. Currently applied values are: • `'General'` (Generic Image type if not specified/identified). • `'Perfusion Rest'` • `'Perfusion Stress'` • `'Strain FFE'` (Strain Fast Field Echo). • `'Strain TFE'` (Strain Turbo Field Echo). • `'Late enhancement'` (Viability). • `'Cine'` • `'Scout'` • `'Qflow'` • `'T2Stir'` • `'T1BB'`

`ImageViewPlane` View plane of images that the image stack depicts. The field is used when identifying what image stacks to use for different analysis. If not set at loading then Segmenttries to figure these details out by looking at what operations have been performed on what image stacks. Currently applied values are:

- `'Unspecified'` (Generic Image view plane if not specified/identified).
- `'2CH'` (Long axis 2 chamber view).
- `'3CH'` (Long axis 3 chamber view).
- `'4CH'` (Long axis 4 chamber view).
- `'Sagittal'`
- `'Coronal'`
- `'Frontal'`
- `'Transversal'`
- `'Short-axis'`
- `'RVOT'`
- `'Aorta'`
- `'Pulmonalis'`
- `'Vena cava inferior'`

`ImagingTechnique`

String of capital letters identifying type acquisition used when acquiring the image stack. Possible values depends are given by the `.par` files. Each such files corresponds to one image type and contains information of how image should be mapped before segmentation. The two first letters should correspond to imaging modality (i.e `MR`, `CT`, `PT`, `US`, `CR`...). Shipped `.par` files are:

- `CTheart.par` (Segmentation of LV).
- `MRBB.par` (MR black-blood sequence).
- `MRDE.par` (MR delayed contrast enhancement).
- `MRGE.par` (MR gradient echo images).
- `MRPDW.par` (MR proton density weighted images).
- `MRSSFP.par` (MR steady state free precision, or fiesta).
- `MRSTIR.par` (MR STIR pulse sequence, edema sequence).
- `MRTOF.par` (MR Time of flight sequence for vessels).
- `NMBPSPECT.par` (Blood pool SPECT images).
- `OT.par` (Myocardial probability, obsoleted).
- `PT.par` (Generic for PET, not for segmentation use).
- `US.par` (Generic for ultrasound. Not for segmentation use).

`IntensityMapping` A struct containing information how the voxel values of the image stack is shown on the screen. The struct has the following fields:

- `Brightness`
- `Contrast`
- `Compression` (Reserved for future use)

The fields `Brightness` and `Contrast` translates into intensity according to the equation:

$$I = cx_i + b - 0.5 \tag{1}$$

where c is contrast, b is brightness, x_i is the input intensity, and I is the output remapped intensity. The intensity I is then clipped to the range [0..1]. The field `Compression` is reserved for future use and will be used to implement image mappings that are sigmoid shaped.

`IntensityScaling` The fields `IntensityScaling` and `IntensityOffset` are used to convert the image data `IM` to the true data as presented in the DICOM images. The true data is calculated as:

$$z = I\alpha + \beta \tag{2}$$

where z is the true image data, I is the data stored in the field `IM`, α is `IntensityScaling`, and `IntensityOffset`.

`IntensityOffset` See above.

`IntensityScaling` See above.

`InversionTime` Inversion time in MRI pulse sequence. Read from DICOM tags if presented otherwise set to zero. Given in milliseconds.

`LVM`	The left ventricle myocardial volume is calculated as `EPV-LVV-PV`. Unit is ml.
`LVV`	The volume within the left ventricular endocardium. Unit is ml.
`LevelSet`	Struct that contains general object segmentation module.
`Linked`	List of numbers of image stacks that are linked to this stack in `Parent/Children` relations.
`Longaxis`	Long-axis motion of the left ventricle, see Reference Manual for details on how the volume is compensated for long-axis motion. To convert to mm subtract with 1.
`MaR`	Struct that contains automatic and/or manual segmentation of the ischemic myocardium at risk from either MRI or SPECT.
`Measure`	Struct that contains data of measurements. Please see Section 15.6.1 for details.
`Mmodex`	X-coordinate for the center of the mmode line.
`Mmodey`	X-coordinate for the center of the mmode line.
`Mmodelx`	Direction coefficient of the mmode line.
`Mmodely`	Direction coefficient of the mmode line.
`Mmodem1`	Distance from the mmode line center to the first measurement point. Usually a positive value.
`Mmodem2`	Distance from the mmode line center to the second measurement point. Usually a negative value.

`Modality`	Same as the corresponding DICOM tag.
`MontageRowZoomState`	Four element vector describing the current zoom state for the image stack in row montage view mode. This representation is potentially subject to future changes.
`MontageZoomState`	Four element vector describing the current zoom state for the image stack in montage view mode. This representation is potentially subject to future changes.
`NormalZoomState`	Four element vector describing the current zoom state for the image stack in normal mode. This representation is potentially subject to future changes.
`NumberOfAverages`	Number of averages acquired in MRI pulse sequence. Read from DICOM tags if presented otherwise set to zero.
`OrgXSize`	This refers to the original X-size of the images in the DICOM files. This is used to load `.seg` files to uncropped image data. The fields `OrgYSize`, `OrgTSize`, and `OrgZSize` have the same function in the other coordinate directions.
`OrigFileName`	Original filename. Note that this information is removed when anonymizing an image stack.
`Overlay`	Number of an image stack that is currently used as a color overlay of this stack.
`PER`	Left ventricular peak ejection rate. Taken as the largest negative derivative of `LVV`.
`PERT`	Time of peak ejection. Given in time frames.

`PFR`	Left ventricle peak filling rate. Calculated using circular convolution with a central difference (three elements) and taken as the largest derivative of `LVV`.
`PFRT`	Time of peak filling. Given in time frames.
`PV`	Volume of the papillary muscles. See Reference Manual for further details how that is calculated.
`PapillaryIM`	Visualisation of papillaries to overlay on image.
`Parent`	Number of the image stack from which this stack was derived.
`PathName`	Path to where the image is saved.

`PatientInfo`

- `Name` is the name of the patient.
- `ID` is the `PatientID`.
- `BirthDate` in the date format `'YYYYMMDD'`.
- `Sex`. Applied values are `''`, `'M'`, `'F'`, `'-'`.
- `Age`. Applied values are `[]`, `''`, numeric, `'78Y'`.
- `HeartRate`. Same as `SET(NO).HeartRate`, retained for backwards compability.
- `AcquisitionDate`. Applied values, `''`, `[]`, and `'YYYYMMDD'`.
- `BSA`. Applied values 0 or numeric value. Manually entered or automatically calculated from Weight and Length. See Reference manual for details and equations used.
- `Weight`. Measured in kilograms. Applied values are 0 or numeric value.
- `Length`. Measured in centimeters. Applied values are 0 or numeric value.

`Perfusion` This is a struct containing information for perfusion analysis.

`Point` Struct that contains data of annotation points. Please see Section 15.6.2 for details.

`ProgramVersion` Describes the version of which the set struct was created. Used for backwards compability issues, and also to detect potential forward compability issues.

`RVEDV` Right ventricle end diastolic volume, see also `EDV`.

`RVEF`	Right ventricle ejection fraction, see also `EF`.
`RVEPV`	Right ventricle epicardial volume, see also `EPV`.
`RVESV`	Right ventricle end systolic volume, see also `ESV`.
`RVEndoInterpX`	RV interpolation points.
`RVEndoInterpXView`	RV interpolation points.
`RVEndoInterpY`	RV interpolation points.
`RVEndoInterpYView`	RV interpolation points.
`RVEndoX`	Endocardial contour of the right ventricle. See also `EndoX`.
`RVEndoXView`	Same as `EndoXView`, but for the right ventricle endocardium.
`RVEndoY`	See above.
`RVEndoYView`	See above.
`RVEpiInterpX`	RV epicardial interpolation points.
`RVEpiInterpXView`	RV epicardial interpolation points.
`RVEpiInterpY`	RV epicardial interpolation points.
`RVEpiInterpYView`	RV epicardial interpolation points.
`RVEpiPinX`	RV epicardial interpolation points.
`RVEpiPinXView`	RV epicardial interpolation points.
`RVEpiPinY`	RV epicardial interpolation points.

RVEpiPinYView	RV epicardial interpolation points.
RVEpiX	Epicardial contour of the right ventricle. See also EpiX.
RVEpiXView	Same as EpiXView, but for the right ventricle endocardium.
RVEpiY	See above.
RVEpiYView	See above.
RVM	Right ventricle mass, see also LVM.
RVSV	Right ventricle strove volume, see also SV.
RVV	Right ventricle volume. See LVV.
RepetitionTime	Repetition time in MRI pulse sequence. Read from DICOM tags if presented otherwise set to zero. Given in milliseconds.
Report	Field used to contain information for the patient report sheet generator functionality. This field is only set for the first image stack (i.e SET(1)). The exact content of this struct is subject to change.
ResolutionX	Contains pixelspace in mm in X-direction. For discussion on coordinate systems, see Section 7. ResolutionY gives pixelsize in mm in Y-direction.
Roi	a struct that contains information of store region of interest (ROI). Please see Section 15.6.8 for details.
RoiCurrent	number of current Roi
RoiN	number of Rois in total in the image

`Rotated`	True for image stacks that are rotated around a common rotation axis. This is currently implemented as an add-on and only affects the volume calculation. Currently 3D visualization does not take this flag into account. This will be addressed in future versions.
`RotationCenter`	Position of the rotation center in the image stack (y-coordinate).
`SV`	Left ventricle stroke volume.
`Scanner`	Identified scanner identified by parsing the DICOM tag `Manufacturer`. Presented values are: • ADAC • Bruker • GE • Philips • Siemens • Suinsa • Toshiba
`Scar`	This is a struct containing information for viability analysis. For details about the algorithms and internal representation, please see 15.6.6.
`SectorRotation`	Rotation of the Sector compartmentization of the left ventricle used for regional myocardial analysis. Unit is degrees. This is can be manually adjusted from the bulls-eye GUI.
`SequenceName`	Name of the sequence, taken from DICOM tags.

`SeriesDescription`	Series description of the image serie, taken from DICOM tags.
`SeriesNumber`	Series Number taken from DICOM tags.
`SliceGap`	Gap between slices in millimeters (if present). Read from DICOM file by looking at the tag `SpacingBetweenSlices` if present. If not present, then the DICOM tags `ImageOrientation` and `ImagePosition` plus `SliceThickness` from above are used.
`SliceThickness`	Thickness of slices in millimeters. Read from DICOM file by looking at the tag `SliceThickness`. If not present, then the tags `ImageOrientation` and `ImagePosition` are used.
`SpectSpecialTag`	Read from DICOM file for determination of the image tag `ImageType` in SPECT images. Valid inputs are 'Rest', 'Rest Prone', 'Stress' or 'Stress Prone'.
`StartAnalysis`	Time frame for start of analysis, used in flow quantification. See Reference Manual for details. Default value is 1.
`StartSlice`	This field contains the first slice in the set of selected slices. For more details, see below.
`Strain`	This is a struct containing information for strain analysis from velocity encoded MR images. For details about the algorithm and internal representation, please see 15.10.3.
`StrainTagging`	This is a struct containing information for strain analysis from tagged MR images.
`Stress`	Reserved for future usage.

`StudyID`	StudyID taken from DICOM tags.
`StudyUID`	StudyUID taken from DICOM tags.
`TDelay`	Trigger delay (i.e starting of image acquisition after trig pulse). Read from DICOM tags if presented otherwise set to zero. Given in seconds. Currently unused.
`TIncr`	Time increment in seconds between time frames. Must be non zero when image stack is time resolved. If increment between time frames is not uniform, the value contained here is the mean time increment.
`TimeVector`	Vector containing time position of each timeframe.
`TSize`	Size of image stack in temporal direction. The following two expressions are equivalent `SET(NO).TSize` and `size(SET(NO).IM,3)`.
`VENC`	Velocity encoding range in MRI pulse sequence. Read from DICOM tags if presented otherwise set to zero. Given in centimeters per second.
`View`	Struct that stores information of the current view. Used to retain the same view after saving an image stack. Only `SET(1).View` is filled.
`XMin`	When loading images it is possible to crop the images. `XMin` contains the number of pixels that the contours are translated due to cropping compared to the original image size. When no cropping is performed `XMin` is 1.

`XSize`	Size of image stack in X-direction. The following two expressions are equivalent `SET(NO).XSize` and `size(SET(NO).IM,1)`.
`YMin`	Same as `XMin` except relates to the Y-direction.
`YSize`	Size of image stack in Y-direction. The following two expressions are equivalent `SET(NO).YSize` and `size(SET(NO).IM,2)`.
`ZSize`	Size of image stack in Z-direction (number of slices). The following two expressions are equivalent `SET(NO).ZSize` and `size(SET(NO).IM,4)`.

14 Implementation Details

This chapter contains the implementation details given in the Segment User Manual.

14.1 Version handling

A proper version handling is employed when developing Segment. A detailed version history of Segment is found in the revision log of Segment SVN.

14.2 Numeric representations

All numbers are stored and used internally as double precision floating points with the following exceptions:

- Images are stored as single floats (normalized) or as integers (uint8), and then as they are stored in the DICOM files. Most functions in Segment will automatically convert the data to floats.
- Edge detection results are stored as integers (16 bits, 'normalized')
- Character strings are stored in 8bit ASCII format
- Infarct maps are stored as int8 (manual interaction), and uint8 (result).
- General segmentation tool store objects as levelset function with an uint8 representation where the zero levelset resides at 128.

Internally the image stack is normalized upon loading by a global maximum intensity such that all values are [0..1]. Offset and scaling is also calculated so that the image stack can be reconverted back to original signal intensities.

14.3 Loading data and interpretation of DICOM tags

This section describes how Segment interprets DICOM information to calculate important parameters suchs as geometric properties of the images.

- Number of slices. This is calculated from the presence of different slices based on the DICOM tags `ImagePosition` and `ImageOrientation`.

- Number of timeframes. This is based on dividing the total number of images with the number of slices.
- Time increment in ms between each timeframe. If uniform, this is based on the difference between the number of timeframes divided by largest and the smallest value of the DICOM tag `TriggerTime`. If the DICOM tag `TriggerTime` is not present then the DICOM tag `TR` is used as time increment. Note that this might depent on your k-space acquisition scheme so for scanners that do not report `TriggerTime` you really need to double check the estimated value of time increment. For perfusion and other image stacks with non-uniform time increment, this is calculated using differences in `AcquisitionTime`.
- Heart rate. The heart rate is taken from the DICOM tag `HeartRate` if present. Note that many vendors (including Siemens) does not specify this. As a fall back Segment tries to calculate the heart rate assuming full R-R intervall coverage by using of trigger time (i.e it does not working for prospective imaging series). For long image acquisitions where one image is taken approximately for each heart beat then the heart rate is taken as the time between start of image acquisition and end of image acquisition adjusted for the number of frames. Note that in many cases this heart rate calculation will fail. Heart rate can be adjusted under patient details. Note also that heart rate may vary between image stacks therefore do not press Apply for all when manually changing heart rate. Heart rate is not used in any calculaion, instead time increment between image frames is used in all calculations.
- Slice thickness in mm. The slice thickness is taken from the DICOM tag `SliceThickness`. If this tag is not present then the information is taken from same DICOM tags as number of slices, and assuming slice gap to be 0.
- Gap between slices in mm. This is taken from the DICOM tag `Spacing BetweenSlices`.
- Pixelspacing in X-direction in mm (vertical direction in Segment). This is taken from the DICOM tag `PixelSpacing`.
- Pixelspacing in X-direction in mm (horisontal direction in Segment). This is taken from the DICOM tag `PixelSpacing`.
- Velocity encoding (VENC) in cm/s. For non velocity encoded images

this should be 0. How this is interpretated involvs proprietry information of different scanner vender information.

- Rotated image stack. This should by default be false. If your image stack is rotated, then change this to true. Currently this parameter is not taken from information in the DICOM tags and the user needs to manually change this when loading rotated image stacks.
- Cyclic image. If the image stack is cyclic, i.e covers the whole heart cycle this should be true (default). For prospectively gated image series this should be false. This affects mainly the automated segmentatin algorithm. Currently this information is not read from the DICOM information.

14.4 Volume calculations

The volume calculations are done by a summing the area in each slice. The main reason for not using a more advanced volume integration method is that no one else is using that and therefore it might be difficult to compare the results. Segmentation (i.e. delineation of endocardium and epicardium) is stored on a sub-pixel accuracy and subsequent calculations are on a sub-pixel basis. For viability the classification into viable or scar is done on a pixel-wise basis and there the volume calculations are done by summing the number of pixels.

For the rotated image stacks the volume is given by a integration method. The volume contribution of each outline is given by :

$$\delta V = \frac{\pi}{2 * Z} \int y(s)^2 \texttt{sign}(y(s)) \frac{dy}{ds} ds \tag{3}$$

where the curve is given on a parametric representation $(x(s), y(s))$, Z is the number of slices in the rotated image stack. No long-axis compensation is performed for the rotated image stacks.

14.5 Mass calculations

When converting volume to mass the density is assumed to be 1.05 g/ml. Note that this number differs in the literature between 1.04 to 1.05. Furthermore, note that these numbers are valid for healthy myocardium ex-vivo,

what happens in for instance infarcted regions is not shown in the literature. Therefore usually it is better to report volume instead of mass.

14.6 Calculation of BSA

The formula used is based on Mosteller.

$$BSA = \sqrt{\frac{w * h}{3600}} \tag{4}$$

where w is the body weight in kg, and h is height in cm.

14.7 Peak ejection/filling rate

When calculating peak ejection and peak filling rate the volume curve is differentiated using forward difference approximation. For cyclic datasets cyclic convolution is used for the calculation.

14.8 Wall thickness

Currently wall thickness is defined as the thickness along a radial spike from the endocardial or the epicardial center (depending on setting in the preferences. In the future I plan to also include the modified center line method. Note that the centers are calculated for each timeframe separately.

Wall thickening is defined as the wall thickness in end-systole minus the wall thickness in end-diastole. Note that it is possible to manually or automatically select what timeframes that are diastole or systole respectively.

Fractional wall thickening is defined as:

$$WT_f = \frac{WT - WT_{ED}}{WT_{ED}} \tag{5}$$

Where WT_f is fractional wall thickness and WT is wall thickness and WT_{ED} is wall thickness in end-diastole. In the bulls eye plot then fractional wall thickening is showed in end-systole.

14.9 Calculation of regurgitant volumes and shunts

The regurgitant fraction for the aortic valve and the pulmonary values are calculated as:

$$r = 100\frac{v_{back}}{v_{forward}} \tag{6}$$

where r is regurgitant fraction, v_{back} is backward volume, and v_{forwd} is forward volumes. Backward volumes is taken as timeframes where the net flow is negative and integrated over the entire cardiac cycle.

The regurgitant fraction for the tricuspid and mitral valve are calculated as:

$$r = 100\frac{SV - v_{forwd}}{SV} \tag{7}$$

where r is regurgitant fraction, and SV is stroke volume for left or right ventricle, respectively. v_{forwd} is forward volume.

The Q_p/Q_s ratio is defined as

$$Q_pQ_s = \frac{Q_p}{Q_p} \tag{8}$$

where Q_p is the stroke volume of the pulmonary artery and Q_s is the stroke volume of the aortic artery.

14.10 Infarct size, extent and transmurality

Calculations of infarct sizes etc are based on 'counting' pixels, i.e. each pixel has a binary classification. There are two methods for regional analysis available, one are based where the percentage of the pixels that are inside the sector. The other method is based on radial spikes from the center (endo- or epicardial depending on setting in the preferences). The line between endocardium and epicardium is resampled in 50 steps and the percentage of infarcted pixels are counted.

Infarct extent is defined as the projected infarcted area on the endocardial surface [4].

$$I_{ext} = \sum_i \frac{T_i R_i}{R_i} \tag{9}$$

where I_{ext} is the infarct extent, T_i is the transmurality of sector i and R_i is the mean endocardial radius of sector i.

14.11 Number of SD from remote for Scar

The number of SD from remote for an existing scar segmentation is calculated by the function found in the main menu in Segment under MR menu Viability menu and then the menu option Get SD from Remote. The presented value is calculated by first calculate the mean and sd in the remote area ($Mean_{remote}$ and SD_{remote}). If there exist ROIs named `Remote ROI`, these regions define the remote area. Otherwise the whole myocardium except for the scar region defines the remote area. The presented SD from remote value is then calculated by

$$SDfromRemote = \frac{T_{optim} - Mean_{remote}}{SD_{remote}} \tag{10}$$

The optimal threshold value (T_{optim}) represent the optimal threshold for seperating the remote and the scar regions based on the existing scar segmentation. This value is defined by an exhaustive search where the threshold is set to all intensities represented in the image stack. For each threshold, the number of missclassified pixels are counted (total of both missclassified remote pixels and missclassified scar pixels). The optimal threshold value is then defined as the threshold corresponding to the minimal number of missclassified pixels.

14.12 MR relaxometry calculations

The MR relaxometry calculation for T1/T2 mapping is given in the paper [5]. Implementation of the ADAPTS T2* mapping is given in the paper [6].

14.13 Pulse wave velocity

The implementation of the pulse wave velocity unit is described in the paper [7].

14.14 Torsion

In short axis cardiac images the heart muscle wall of the left chamber is well approximated by a circle. The method finds the axis of rotation, AoR, for the left chamber as the center of a circle fit to the tracking points generated by the segment strain module. For the circle fitting a least squares method is used.

14.14.1 Least squares circle fit

The circle is fitted by minimizing the global squared radial difference between all tracking points for all timeframes, (x_i, y_i), $i = 1, ..., N$ and a circle with radius $r = \sqrt{a}$ for each slice. For nicer calculations we make the tracking point cloud zero mean and define a new coordinate system

$$u = x - \frac{1}{N}\sum_{i}^{N} x_i, \quad v = y - \frac{1}{N}\sum_{i}^{N} y_i \tag{11}$$

The properties of the circle determining the fit is the radius r and center (u_c, v_c) The circle equation we are going to work with is

$$f(u, v) = (u - u_c)^2 + (v - v_c)^2 - a = 0 \tag{12}$$

which yields the least squares expression we want to minimize.

$$M(a, u_c, v_c) = \sum_{i}^{N} f^2(u_i, v_i) = \sum_{i}^{N}((u_i - u_c)^2 + (v_i - v_c)^2 - a)^2 = 0 \tag{13}$$

The minima is found by solving,

$$\frac{dM}{da} = 0 \tag{14}$$

$$\frac{dM}{du_c} = 0 \tag{15}$$

$$\frac{dM}{dv_c} = 0 \tag{16}$$

for all parameters of M. From (14) we get that

$$\frac{dM}{da} = 2\sum_{i}^{N} f(u_i, v_i)\frac{df(u_i, v_i)}{da} = -2\sum_{i}^{N} f(u_i, v_i) = 0. \tag{17}$$

Resulting in

$$\frac{dM}{da} = 0 \iff \sum_{i}^{N} f(u_i, v_i) = 0. \tag{18}$$

Then consider (15). As (15) (16) only differ in notation, any result for (15) is applicable to 16.

$$\frac{dM}{du_c} = 2\sum_{i}^{N} f(u_i, v_i)\frac{df(u_i, v_i)}{du_c} = 4\sum_{i}^{N}(u_i - u_c)f(u_i, v_i) \tag{19}$$

Since 18,

$$\frac{dM}{du_c} = 0 \iff \sum_{i}^{N} u_i f(u_i, v_i) = 0. \tag{20}$$

and the same goes for 16.

$$\frac{dM}{dv_c} = 0 \iff \sum_{i}^{N} v_i f(u_i, v_i) = 0. \tag{21}$$

expanding equation (20) yields

$$\frac{dM}{du_c} = \sum_{i}^{N} u_i(u_i^2 - 2u_i u_c + u_c^2 + v_i^2 - 2v_i v_c + v_c^2 a) = 0 \tag{22}$$

Define $S_u = \sum_i^N u_i$ and $S_v = \sum_i^N v_i$ then

$$\frac{dM}{du_c} = S_{u^3} - 2u_c S_{u^2} + u_c^2 S_u + S_{uv^2} - 2v_c S_{uv} + v_c^2 S_u - aS_u = 0 \tag{23}$$

In making the coordinates zero mean $S_u = 0$ we get the equation

$$u_c S_{u^2} + vcS_{uv} = \frac{1}{2}(S_{u^3} + S_{uv^2}) \tag{24}$$

After doing the same for (21) we obtain the system

$$\begin{cases} u_c S_{u^2} + vc S_{uv} = \frac{1}{2}(S_{u^3} + S_{uv^2}) \\ u_c S_{uv} + v_c S_{v^2} = \frac{1}{2}(S_{v^3} + S_{vu^2}) \end{cases} \tag{25}$$

which can be converted into a matrix equation

$$\begin{bmatrix} S_{u^2} & S_{uv} \\ S_{uv} & S_{v^2} \end{bmatrix} \begin{bmatrix} u_c \\ v_c \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(S_{u^3} + S_{uv^2}) \\ \frac{1}{2}(S_{v^3} + S_{vu^2}) \end{bmatrix} \tag{26}$$

This gives us an easy way to get the least squares fitted circle center. For the center in the original (x, y) domain translate with the previously subtracted mean. Finally for the radius, expanding equation (18) and simplifying yields

$$a = u_c^2 + v_c^2 + \frac{S_{u^2} + S_{v^2}}{N}, \tag{27}$$

where

$$r = \sqrt{a}. \tag{28}$$

14.14.2 Angular discontinuity detection

After fitting a circle to each time frame with tracking points we can translate the points in each time frame so that the fitted circle center i.e the AoR is in origo. With this in place a polar coordinate change results in an approximate line like formation of the points, lets call it a worm. Who's movement along the θ axis is the rotation of the heart muscle. Here a problem arises. Since $\theta \in [-\pi,\ \pi]$, $\theta_t + \Delta\theta > |pi|$ results in a sign change and the point appears at the lower limit if it passed the upper and vice versa. This needs to be mended if we are to measure angular distance from a starting point. This is done by examining

$$\Delta\theta_t = \theta_t - \theta_{t+1} \tag{29}$$

for each tracking point, adjusting the point with $\pm\pi$ (sign depends on border transition) if $|\Delta\theta_t| > 5$.

Torsion is then found as the difference between the rotation in a apical and a basal slice normalized with the distance along the long axis of the heart between the slices and the mean radius.

14.15 Longaxis volumes

Volumes can be calculated using segmentation from longaxis images. The algorithm begins with automatically locating images labeled 2CH, 3CH and 4CH that contain segmentation. If the same kind of segmentation is found in two such images, the volume is calculated by rotating each segmentation area one full revolution around the axis of intersection and taking the mean of these volumes. If there are three images that contain the same segmentation, the volumes are calculated as described above for each pair of images, and the mean of these three values is used.

15 Unit Implementation

This chapter contains the implementation of different units of Segment. Some of the units are modules, which means that they are a part of the program that represents a separate module and requires a special license file. A unit is the smallest divided unit of the code documented in the technical manual.

15.1 Main Segment superunit

The purpose of the main Segment block is to work as a backbone for which all other units can be incorporated. It hosts almost all central callbacks and motion functions.

The unit consists of the file `segment_main.m`, the GUI superclass `maingui.m` and its software specific subclass. This class hierarchy is documented in detail in Chapter 11.

Interactions

Superunits called from the main Segment unit are

- Draw - Graphical updates upon commands to change view or modify an image.
- Calc - Calculations for images or quantifications that are used in the GUI.

Datastructure

The data structure is the previously described `SET` structure and the `DATA` object.

Functions in `segment_main.m`

addnotopanel(no)

addtopanels(no,mode)

```
Finds an open space, otherwise increases number of panels.
```

addviewicon_helper(callback,tooltip,cdata,tag,separator)
```
Helper fcn to add an icon.
```

allhidden

[x,y,name] = askcontour(queststri)
```
Show menu so that user can indicate what contour to use.
Used by levelset to import contours, and by export function
to export contours as ascii file.

returns contour in x, and y, and a name of the contour.
```

autocontrast(no,silent)
```
Helper functionk to autocontrast_Callback_.
```

autocontrast_Callback
```
Automatically calculates contrast settings.
```

autocontrastall_Callback
```
Automatically calculates contrast settings.
```

buttondowntoggler(caller,panel)
```
when clicking in a image toggles to the correct handle buttondown.
```

[varargout] = cell2clipboard(outdata,writetofile)
```
Converts a cell to a string that is output to clipboard.
If more than 8000 cells are written then an .xls file is
written instead. Note that this used active-X on Windows and requires
Excel to be installed on the computer.
```

center_Buttondown(panel)
```
Called when center '+' is pressed down, sets motion and buttonup fcns.
```

center_Buttonup
```
This function is called when buttonup occurs after draging center point.
```

center_Motion
```
Motion function of the center point.
```

centeronslice(slice,no,zsz)
```
Put slice in center of montagefit view.
```

changewheel_Callback(h,e)
```
scrollwheel with modifer.
Tab are not included but you can if you like.
```

checkconsistency(timeframes,slice)
```
Check consistency, to prevent earlier
manual segmentations that have problems with
direction lef/right.
```

[xout,yout] = checkconsistencyhelper(xin,yin)
```
Make sure that the contour is counter clock-wise and that it starts at
three o clock. Also ensures that the points are evenly distributed.
```

corrupted = checkcorrupteddataforautomaticsave(setstruct)
```
This function checks if the data (SET) is corrupted due to corrupted
loading when loading files which has been saved with older saveversion
 see ticket 502 in wush for more details on the bug.
```

checkforduplicatehide(name,state)

cinetool_Callback
```
Starts the cinetool that allows simultanues segmentation at the
same time as it plays.
```

clickedpin_Callback(type)
```
Called when user clicks on a pin.
```

contrast_Callback(arg,panel)
```
Activated by contrast tool, different from resetlight_Callback (above).
```

mask = createmask(outsize,y,x)
```
Function to generate a mask from a polygon represented with the vectors x
and y.
```

ctrlc
```
function to handle ctrl-c keypress, which is disabled.
```

do = doatrialscar(no)
```
Helper function to check if user input is for atrial scar or lv scar
(default).
```

doputpin_Callback(type)
Put pins. Called when clicked, puts an pin and refines.

dragepi_Buttondown(type,panel)
Button down function for scaling of objects / contours.

dragepi_Buttonup(type)
Button up function for scaling.

dragepi_Motion(reset)
Motion function for scaling.

ok = enablecalculation
Returns ok, if slices are selected. Sideeffect of
this function is that it turns on interaction lock, stops movie. Calling
this function will not allow user to click on a new function before
endoffcalculation is called. Note that this mechanism needs to be pretty
safe with try/catch clauses otherwise Segment may hang. Viewrefresh calls
endoffcalculation.

endo_Buttondown(panel)
Button down function for manual draw of endocardium.

endoffcalculation
Turns off interaction lock, called after a calculation. See above.

epi_Buttondown(panel)
Button down function for manual draw of endocardium.

esed_Buttondown(type)
Buttondown function when dragging ES or ED markers in volume graph.

esed_Buttonup(type)
Buttonup function when dragging ES or ED markers in volume graph.

esed_Motion(type)
Motion function when dragging ES or ED markers in volume graph.

esedtimebar_Buttondown(type)
Buttondown function when dragging ES or ED markers in time bar graph.

esedtimebar_Buttonup(type)
Buttonup function when dragging ES or ED markers.

esedtimebar_Motion(type)
Motion function when dragging ES or ED markers in time bar graph.

evalcommand_Callback
Function to evaluate matlab commands from compiled version.

fasterframerate_Callback
Makes movie play faster.

figure_DeleteFcn
Shut down Segment in a controlled manner, and remove
global DATA SET NO. Note that filequit callback takes
care of asking user yesno.

flowaxes_Buttondown
Called when user has clicked flow graph, sets current
timeframe to clicked point.

framemode_Callback(type)
If type 1 we have pressed the single frame mode button type 2 the multi frame mode button

preview = genemptypreview(datapath)
Generate an empty preview struct.

[x,y,slice] = getclickedcoords
Find coordinates where the user last clicked. x and y are given in internal
coordinate system, i.e the functions determines slice in montage view.

no = getclickedpreview(˜,y)
function which returns the clicked preview image.

[intersections, maxintersect] = getendointersection(no)
Returns the intersection of the endocardial segmentation and the current
slice and current time frame of SET(no).

r = getfieldifcommon(SET, fname)
Helper function to filesavedicom_Callback.

t = getframenumber
Calculates what frame to show when playing a movie if storing a movie then show next frame otherwise user timer info.

slicestoinclude = getmontagesegmentedslices(no)
Get slices to include in montage segmented view.

numericversion = getnumericversion
function called to get versionnumber after R. For example in '1.9 R4040' this func

helpimagepos(im,sector,ofs,konst)
helper function to display image.

helpimageposneg(im,sector,konst)
helper function to display image.

highlighttool(h)
Helper function to change color of a tool to represent that the tool is selected.

state = iconson(name)
if given name of button return state if run with no input returns states of all buttons. if given cell with multiple button return icons in order of request.

fig = initializesegment(programversion)
Initialization of Segment GUI.

initmenu
Initalize the main menu, for instance adds extra utilities, and plugins.

iconhandles = initviewtoolbar
Initalize view toolbar.

interpdeletepoint
Delete interp point.

interpdeletepointall
Delete interp points for all slices timeframes.

interpdeletepointthisslice
Delete interp points this slice.

interpdeletepointthisslicephase
```
Delete interp points this slice and phase.
```

interpdrawGuessPoints_Callback(type,panel)
```
Guess points in current selection then moves to corrosponding tool.
```

interpdraw_Buttondown(panel,type,forcedraw)
```
Button down function to draw interp points.
```

[interpx,interpy] = interpdrawhelper(interpx,interpy,contourx,contoury,x,y)
```
Helper fcn to interpdrawbuttonup
Side effects is update of DATA.Pin.
```

[xout,yout] = interphelper(pinx,piny)
```
Interpolates points to create a contour without loops.
```

interppointButtonup(type)
```
Button up function for interp points.
```

interppointMotion(type)
```
Motion function for interp points.
```

tool = interptoolfromcoords(x,y,slice)
```
get tool from point
valid for 'interpendo' 'interpepi' 'interprvendo' 'interprvepi'.
```

mainresize_Callback
```
This fcn is called when user resizes GUI.
```

makeviewim(panel,no)
```
Rearrange data to show all slices.
```

manualdraw_Buttondown(panel,type,new)
```
Button down function for manual drawings.
```

manualdraw_Buttonup(type,new,obj)
```
Button up function for manual drawing.
```

manualdraw_Motion
```
Motion function for manual drawings.
```

mccheckconsistency
Check consistency, to prevent earlier
manual segmentations that have problems with
direction left/right.

measure_Buttondown(panel)
Button down function for placing measurements.

measure_Motion(ind)
Motion function for measurements.

measure_Motion_translate(reset)
Motion function for measurements.

measureexport_Callback
Export measurements.

measuremove_Callback(dx,dy)
Helper function to move measurements.

measurepoint_Buttondown(panel)
Buttondown function when clicking on a measurement point/marker.

measureput_Buttonup
Called when a measurement point (except the endpoint) is placed.

measureshapeexport_Callback
Export measurement shape.

mmode1_Buttondown(panel)
Called when mmode1 is pressed down, sets motion and buttonup
function.

mmode1_Motion
Motion function of the first mmode point.

mmode1line_Buttondown(panel)
Called when mmode1line is pressed down, sets motion and buttonup
function.

mmode1line_Motion
Motion function of the first mmode line.

mmode2_Buttondown(panel)
Called when mmode1 is pressed down, sets motion and buttonup function.

mmode2_Motion
Motion function of the second mmode point.

mmode2line_Buttondown(panel)
Called when mmode1line is pressed down, sets motion and buttonup function.

mmode2line_Motion
Motion function of the second mmode line.

mmode_Buttonup
This function is called when buttonup occurs after draging one of the mmode objects.

mmodecenter_Buttondown(panel)
Called when mmode center is pressed down, sets motion and buttonup function.

mmodecenter_Motion
Motion function of the mmode center point.

mmodempoint1_Motion
Motion function for mmodepoint one.

mmodempoint2_Motion
Motion function for mmodepoint two.

mmodepoint1_Buttondown(panel)
Called when mmodepoint1 is pressed down, sets motion and buttonup function.

mmodepoint2_Buttondown(panel)
Called when mmodepoint1 is pressed down, sets motion and buttonup function.

mmodetimebar1_Buttondown(panel)
Called when mmodepoint1 is pressed down, sets motion and buttonup function.

mmodetimebar1_Motion
Motion function for mmodepoint timebar.

mmodetimebar2_Buttondown(panel)
Called when mmodepoint1 is pressed down, sets motion and buttonup function.

mmodetimebar2_Motion
Motion function for mmodepoint timebar.

montage_Buttondown(panel)
Button down function for selecting slices in montage view.

montage_Buttonup(panel)
Button up fucntion for selecting slices in montage view.

montage_Motion
Motion function when selection slices in montage view.

move_Buttondown(type,panel)
Button down function for moving / translating objects / contours.

move_Buttonup(type)
Button up function for moving functions.

move_Motion(reset)
Motion function for moving / translating objects / contours.

moveall_Buttondown(type,panel)
Button down function for moving / translating all objects / contours.

moveall_Buttonup(type)
Button up function for translating all objects / contours.

moveall_Motion(reset)
Motion function for moving / translating all objects / contours.

movealltowardsapex_Callback
Changes CurrentSlice (and that of parallel image stacks) towards apex.

movealltowardsbase_Callback
```
Changes CurrentSlice (and that of parallel image stacks)
 towards base.
```

movetowardsapex_Callback
```
Change current slice towards apex.
```

movetowardsbase_Callback
```
Change current slice towards base.
```

nextallframe_Callback
```
Displays next timeframe in currrent image panel and adjust all visiable
image stacks to the corresponding part of the cardiac cycle.
```

nextframe_Callback
```
Displays next timeframe of current image panel. Sideeffect is that the
movie display is stopped if running.
```

normal_Buttondown(panel)
```
Called when button down in normal view.
```

sameview = orientationcomparison(setindex1,setindex2)
```
Compares the SET.ImageOrientation between two SETs.
 Help function to updateparallelsets

 Return values:
 sameview : true if the orientations are parallel.
```

pan_Buttondown
```
Button down function for panning of current image panel.
```

pan_Buttonup
```
Button up function for panning.
```

pan_Motion(init)
```
Motion function of pan.
```

pin_Buttonup
```
Button up function for pins.
```

pin_Motion(type)
```
Motion function for pins.
```

[PinX,PinY,pinwarn] = pinresize(PinX,PinY,tf,slice,mean,f)
Helper function to move pins when resizing contours.

placetimeresolvedpoints_Callback
When this option is enabled then points are placed timeresolved and the name is re-used.

playall_Callback(keypress)

playall_Helper
Starts movie display of all visible image stacks.

playmovie_Callback(keypress)
Starts playing current image stack as a movie.

plotmodelrot_Callback(daz,del)
Changes view of 3D model of the segmentation.

[x,y] = pointsfromcontour(X,Y,nbr_points)
select points from X,Y with probability higher when derivative magnitude in respect to distance to mass centre is large.

previousallframe_Callback
Displays previous time frame in current image panel. For all other visible image stacks they are adjusted to show correspondig part of the cardiac cycle.

previousframe_Callback
Displays previous time frame of current panel.

putpin_Buttondown(panel,type)
Button down function for pins.

putpin_Callback(panel,type)
Callback to put pins.

recursekeypressfcn(h,fcn)
Helper function to create callbacks to keypressed function.

z = remap(im,cmap,c,b)
Remap data according to cmap.

renderstacksfromdicom(no)
Render image stacks in main gui. This function is typically called upon loading.

resetguipositions
Resets GUI positions of Segment.

resetlight_Callback
Activated by toolbar icon, different from contrast_Callback (below).

resetpreview
Reset preview structure in DATA.Preview.

z = reshape2layout(im,no,panel,outsideelement)
Convert a 3D array to an layout:ed image with cols, and rows.

rotatetemp(alpha)
This function should late be replaced with qa tool that rotates objects, just as scale and move does.

saveguipositiontodisk
Save guipositions to disk.

scale_Buttondown(type,panel)
Button down function for scaling of objects / contours.

scale_Buttonup(type)
Button up function for scaling.

scale_Motion(reset)
Motion function for scaling.

selectallslices_Callback
Selects all slices in current image stack.

setcurrenttimeframe(frame)
function to set current time frame, input argument is frame to be set to current time frame.

slowerframerate_Callback
Makes movie play slower.

smoothendowall_Callback(no)
Adjust the LV wall so that the thickness is more even. Adjustment is done on the endocardial side.

stopmovie_Callback
End movie display.

switchtolongaxisslice(slice,laxfield,silent)
Called when slice has changed in HLA or VLA view.

killbuttondown = switchtopanel(panel,updateimagestack)
Make panel the currentpanel.

switchtoslice(slice)
Called when current slice has changed.
If slice has changed, make sure montage/one are in sync
Does nothing if slice = CurrentSlice.

thumbnail_Buttondown
Buttondown fcn for thumbnails.

thumbnail_Buttonup
Buttonup fcn for thumbnails.

thumbnail_Motion
Motion fcn for thumbnails.

out = thumbnailno(in)
Helper fcn to remember what image stack were clicked.

thumbnailslider_Callback(varargin)
Slider callback for thumbnail slider.

timebar_Buttondown
Button down function for the time bar in the time bar graph.

timebar_Motion
Update current timeframe when user clicked in time bar graph.

timebaraxes_Buttondown
```
Called when user has clicked time bar graph, sets current
timeframe to clicked point.
```

timebarflow_Buttondown
```
Button down function for the time bar in the volume graph.
```

timebarflow_Motion
```
Update current timeframe when user clicked in volume graph.
```

timebarlv_Buttondown
```
Button down function for the time bar in the volume graph.
```

timebarlv_Motion
```
Update current timeframe when user clicked in volume graph.
```

unlighttool(h)
```
Helper function to change color of a tool to represent unselected.
```

unselectallslices_Callback
```
Unselects slices in current image stack.
```

update_thumbnail(nos)
```
This fcn updates thumbnail no.
```

updateflow
```
calculate flow and graphically update.
```

result = updateintersections_Callback(slices,no,type)
```
Calculates the intersection of a endocardial segmentation
 and SET(no) for
 slices == 'all'
 or
 slices == 'current'

 Allowed segmenation 'type' is 'LVEndo' or 'RVEndo'

 Calculates intersection for SET(NO) if nargin==1.
 Returns false if calculations fails.
```

updatemmode(arg,nbrofcycles)
```
Calculate and show mmode image.
```

updatemmodeline
Updates the mmode line in mmode display.

updatemmodevisibility
Updates mmode visibility.

updatemodeldisplay(~)
Do nothing, introduced to disable excessive calls to updatemodeldisplay.

updateoneim(no)
Updates viewim for 'one view' or 'mmodespatial'
Called when changed currentslice.

updateparallelsets
Chages CurrentSlice in SETs that are parallel to SET(NO) to the slice
closest to SET(NO).CurrentSlice

Help function to movealltowardsbase/_Callback and
movealltowardsapex_Callback.

updateselectedslices
Graphically updates which slices are selected.

updateslider(whattodo)
Update when user changes in thumbnail slider.

updatevolume(lvsegchanged)
Calc volume of segmentation and graphically update.

viewaddtoolbar_Callback
Helper function to add a toolbar.

viewallimagestacks_Callback
Displays all image stacks.

viewhideall_Callback(varargin)
all hide buttons.

viewhidecolorbar_Callback
Toggle visibility of colorbar.

viewhideimagestack_Callback(no)
```
Remove clicked image stack from view panels.
```

viewhideinterp_Callback(varargin)
```
Toggle visibility of contours from other image stacks.
```

viewhideintersections_Callback(varargin)
```
Toggle visibility of plane intersections.
```

viewhidelv_Callback(varargin)
```
Toggle visibility of lv segmentation.
```

viewhidemanualinteraction_Callback
```
toogle visibility of manual interaction of Scar/MaR.
```

viewhidemar_Callback(varargin)
```
Toggle visibility of MaR contours
global DATA
drawfunctions('drawall',DATA.ViewMatrix);.
```

viewhidemeasures_Callback
```
Toggle visibility of measurements.
```

viewhideothercontour_Callback(varargin)
```
Toggle visibility of contours from other image stacks.
```

viewhidepanel_Callback
```
Hides current panel.
```

viewhidepap_Callback
```
Toggle visibility of papillary overlay.
```

viewhidepins_Callback(varargin)
```
Toggle visibility of pins.
```

viewhideplus_Callback(varargin)
```
Toggle visibility of center +.
```

viewhidepoints_Callback
```
Toggle visibility of annotation points.
```

viewhideroi_Callback(varargin)
Toggle visibility of roi's.

viewhiderv_Callback(varargin)
Toggle visibility of rv segmentation.

viewhidescar_Callback(varargin)
Toggle visibility of scar contours
global DATA.

viewhidetext_Callback(varargin)
Toggle visibility of text.

viewimage_Callback(type)
Select type of image to view on screen.

viewimagestack_Callback(no)
Make clicked image stack visible.

viewimagestackas_Callback(mode)
Select how to view current image stack.

viewinterp_Callback(val)

viewmanualinteraction_Callback
Displays a GUI indicating in which slices and timeframes manual interaction have been made for LV segmentation.

viewpandir_Callback(direction)
Pans current view panel.

viewrefresh_Callback
Main graphical refresh.

viewrefreshall_Callback
Main refresh of GUI.

viewspecial_Callback(mode)
Called to switch to predefined layouts of the GUI. Mode is current mode to use.

viewzoomin_Callback
Zooms in current view panel.

viewzoomout_Callback
Zooms out in current view panel.

volumeaxes_Buttondown
Called when user has clicked volume graph, sets current timeframe to clicked point.

zoomhelper(ax,f,no,panel)
Helper function to zoom image stacks. Zooming is done by changing xlim and ylim.

Functions in `maingui.m`

addmainicon_helper(g,callback,tooltip,cdata,tag,separator)
Helper function to add an icon.

checkversion(g)
Check if new version is available. Define separately for each GUI.

cleardatalevelset(g,onlyprototype)
Clear the struct g.LevelSet properly called when switching image stacks.

contextmenu(g)
Context menu button down / callback
Overloaded in CVQgui
Note: Can act on one slice (CurrentSlice) or range of slices (StartSlice:EndSlice).

copyrvendoforward
Overloaded in SegmentGUI.

defaultpref(g)
Sets Pref to default values. Called by segpref('default/_Callback')
Overloaded in CVQgui, RVQgui, Segment CMRgui.

dispwelcometext(g)
Displays welcome text. Overloaded in CVQgui, SegmentCMR GUI.

drawroiinpanel(g,panel)
Draw ROI's in one slice mode.

drawsectorgrid(g,no)
Overloaded in RVQ GUI.

drawvolume(g)
Draw volume curve. Overloaded in CVQgui.

enableopenfile(g)
Called by openfile('enablesegmentgui'). Overloaded in CVQgui and RVQgui.

filecloseall_Callback(g,silent)
Deletes all image stacks and resets Segment to original state.
Overloaded in CVQgui and RVQgui.

fileopen_Callback(g)
Loads a set of files as a 4D volume, and also updates preferences.
Calls the fcn openfile which displays the fileloader GUI.
Overloaded in CVQgui.

fail = filesaveallas_Callback(g,pathname,filename)
Overloaded in some other GUI's.

filename = generatesavefilename
Called by segment('filesaveallas_Callback'). Overloaded in CVQgui.

pathname = getpreferencespath(g)
Get path to preferences folder.

value = getthisframeonly(g)
Function to check thisframe onlye mode or not.

heartratezero(g, heartrateest)
React upon a heart rate read as zero from DICOM in openfile('initset').

iconcallbackhelper(handle,callback)
Helper function to updateicons.

handle = iconcdatahelper(handle,icondata,tiptext)
Helper function to updateicons.

[type,viewplane,technique] = imagedescription
Define the image types, image view planes and imaging techniques
written by Helen Soneson 2009-05-25
Moved by Nils Lundahl to maingui.m 2012-07-20
Overloaded in RVQ GUI.

init(g)
Create GUI.

initLogFile(g)
Initiate log file.

initbullseye
Autoselects slices with myocardium in them before GUI is opened
Overloaded in Segment CMR GUI and Segment Spect GUI.

initmaingui(g)
Initiates main GUI. Overloaded in most other GUI's.

initmaintoolbar(g)
This method is overloaded in any GUI with a toolbar.

varargout = initopenfile(g)
Initializes the openfile GUI. Optional output is the fig
Called by openfile (main). Overloaded in CVQgui
#doprotect.

initreportflow(handles)
Overloaded in CVQgui.

inittoolbar(g)
Initiates toolbars. Overloaded in CVQgui.

loadguipositions(g)
Load prestored positions of GUI's.

g = maingui(programversion)
Constructor
Initialization of Segment GUI.

ok = manualdraw_Buttonup_roi(no,xr,yr,slice)
Called by segment/_main('manualdraw/_Buttonup')
Overloaded in CVQgui and RVQgui.

measure_Buttonup(g)
Button up function for measurements.
Overloaded in CVQgui.

[stri,lstr] = measureasklabel(g)
Asks for a label of a measurement.
Overloaded in CVQgui and RVQgui.

measureclearall_Callback(g)
Clear all measurements.

measureclearthis_Callback(g,arg)
Clear current measurement.
Overloaded in RVQgui.

measurefontsize(g,panel,index)
Sets measure font size. Used in CVQgui.

measurerenamethis_Callback(g,arg)
Rename current measurement.
Overloaded in CVQgui.

mywaitbarclose(h)
Makes it possible to overload mywaitbar behaviour.

h = mywaitbarstart(iter,stri,ignorelimit,fighandle)
Makes it possible to overload the mywaitbarstart behaviour.

h = mywaitbarupdate(h)
Makes it possible to overload mywaitbar behaviour.

normalizephaseupdate(g)
Called by segpref.m methods
Overloaded in CVQgui (different handle placement).

plotrois(g,panel,no)
Plot roi's if existing. Overloaded in CVQgui.

point_Buttondown(g,panel)
Button down function when point tool is active.
Overloaded in CVQgui.

pointshowthisframeonly_Callback
```
Callback to make the current point visible in this time frame only.
Overloaded in CVQgui.
```

printthumbnailnumber(g,thumbsize)
```
Overloaded in RVQ GUI.
```

really = quit(g)
```
Quit segment, also ask user if he/she is sure.
Overloaded in CVQgui.
```

renderstacksfrommat(g)
```
This function displays stacks from a mat files in main gui.
Typically called upon loading;.
```

name = roilabelmenu(g,roitoname,roinamein)
```
Prompt name of ROI from a menu selection
Overloaded in CVQ and RVQ.
```

roiputroi_helper(g,no,m)
```
Called by roi('roiputroi/_Buttondown')
Overloaded in CVQgui and RVQgui.
```

segmentclearall_Callback(g,force)
```
Clear all segmentation, both endo and epi, lv and rv
Overloaded in CVQgui and RVQgui.
```

segmentclearallbutsystolediastole_Callback(g)
```
Clears all segmentation in all timeframes but systole and disastole.
```

segmentclearalllv_Callback(g,force)
```
Clear all LV segmentation, both endo and epi
Overloaded in CVQgui.
```

segmentclearalllvbutsystolediastole_Callback(g)
```
Clears all LV segmentation except in systole and diastole.
Overloaded in CVQgui.
```

segmentclearallrv_Callback(g,force)
```
Clear all RV segmentation, both endo and epi
Overloaded in CVQgui.
```

segmentclearallrvbutsystolediastole_Callback(g)
Clears all RV segmentation except in systole and diastole.
Overloaded in CVQgui.

setprefhandles(g)
Called by segpref (main). Overloaded in CVQgui since uses another fig.

setthisframeonly(g,value)
Callback to set this frame only mode.

startlog(log)
Start Segment log.

stoplog
Stop Segment logging.

switchtoimagestack(g, no,force)
This function makes no current image stack, and updates graphics.

switchtoimagestack_part2(g, no)
switchtoimagestack continued. Overloaded in SegmentGUI and CVQgui.

thisframeonly_Callback(g,thisframeonly,silent)
Sets segment in 'this frame only' mode. Changes made to segmentation,
copying etc are performed only for the current timeframe.

togglescar(g)
Called by segment('updateviewicons'). Overloaded in CVQgui.

updateaxestables(g, arg, varargin)
Method to update AxesTables, in GUIs where present.

updateedes
Overloaded in CVQgui.

updateicons(g,mode)
Updates icons when new mode is selected.
Overloaded in CVQgui, RVQgui and Segment CMR GUI
If making major changes, make sure to
update csegment('updateicons')!!! /JU.

updateprefhandles(g)
Updates PrefHandles. Called by segpref('update')
Overloaded in CVQgui and RVQgui.
Default folder locations.

updateprefhandlesadvanced(g)
Called by segpref('updateadvanced'). Overloaded in SegmentGUI and RVQGUI.

updatetimethings(g,no,mode)
This fcn is called from switchimagestack and disables/enables features that are not available depending of the image stack is timeresolved or not.
Overloaded in CVQgui.

updatetitle(g)
Updates titleline of main GUI. Overloaded in CVQgui.

updatevolumeaxes(g)
Called by segment_main('updatevolume').

versionhello
Versionhello, overloaded in CVQgui, RVQ, SegmentTransfer...

Functions in segmentgui.m

checkversion(g)
Check if new version is available.

copyrvendoforward
Overloads empty method in maingui.

failedaborted(g)
Message when a function is aborted.

initmaintoolbar(g)
Intialization of main toolbar
If new handles are added, add those to list in killhandles.m as well!.

licensemsg(m)
Display license message.

g = segmentgui(programversion)
Constructor.

tip = singleframetip
Return tip for single frame mode.

switchtoimagestack_part2(g, no)
switchtoimagestack continued. Overloads method in maingui.

updateicons(g,mode)
Update icons when new mode is selected
Overloads main GUI method.

updateprefhandlesadvanced(g)
Make update of handles in advanced preferences GUI.

out = viabilityallowweighted
Return whether to allow weighted viability mode.

Functions in maingui.m

defaultpref(g)
Sets Pref to default values. Overloads method in main GUI.

dispwelcometext(g)
Displays welcome text. Overloaded in CVQgui, SegmentCMR GUI.

fileopen_Callback(g)
Loads a set of files as a 4D volume, and also updates preferences.
Calls the fcn openfile which displays the fileloader GUI.
Overloaded in CVQgui.

filesaveallas_Callback(g,pathname,filename,~)
```
savetodatabase=true;
  //%here should be a switch-case clause on set preferences for save to
  /%local disk or patientdatabase
  if savetodatabase.
```

initLogFile(g)
Initiate log file. Overloads main GUI method.

initbullseye
```
Autoselects slices with myocardium in them before GUI is opened.
```

initmaingui(g)
```
Initiates main GUI. Overload SegmentGUI method to use CMR gui instead.
```

initmaintoolbar(g)
```
Intialization of main toolbar
If new handles are added, add these to list in killhandles.m as well!.
```

inittoolbar(g)
```
Initiates toolbars.
Create menubar.
```

licensemsg(~)
```
Overloads SegmentGUI method.
```

g = segmentmrgui(programversion)
```
Constructor.
```

tip = singleframetip
```
Return tip for single frame mode.
```

updateicons(g,mode)
```
Updates icons when new mode is selected.
Overloads method in main GUI
 If making major changes, make sure to
 update csegment('updateicons')!!! /JU.
```

out = viabilityallowweighted
```
Return whether to allow weighted viability mode.
```

15.2 Draw superunit

The purpose of the draw unit is to perform all graphical update of image panels in Segment.

Interactions

This unit calls the Calculation superunits to help calculate some of the details that are to be drawn.

Datastructure

The functions uses the `SET` structure, and the fields `.rows` and `.cols` are especially important since they contain the number of panels.

Other important data structure are in the `DATA` object.

1. `ViewIM` contains a cell array of preprocessed images ready to be displayed. One for each available image panel. The images are either RGB images or uint8 images that goes to grayscale mapping.
2. `ViewMatrix` contains the number of image panels available on the screen in the current view mode.
3. `ViewPanels` a vector that linkes to the image stack number for each panel. For non active panels the number is zero.
4. `ViewPanelsType` a cell array with the type of each image panel.

Functions

s = cellref(a,varargin)

```
Returns vector of content of e(varargin), for all elements e of cell a.
```

drawall(n,m)

```
Draws all panels that should be visible or up to the number specified as n.
  If called with n=[], then lookup from imagepanels
  When n,m specified draw n*m panels.
```

drawallslices

```
This fcn updates graphics in all visible image panels.
```

drawannotationpoints(no,panel)

```
Draw annotation points, if available.
```

drawcolorbar(panel)

```
Draw color bar in an image panel.
```

drawcontours(no,panel)

```
Initiate handles and draw endo- and epicardial contours of the LV and RV.
```

drawcontrastimage(no)
draws contrast image.

drawimagehlavla(panel,viashow,marshow,olshow)
Main workhorse for creating view of one image slice.
- viashow is whether to show viability
- marshow is whether to show MaR
- olshow is whether to show overlays or not.

drawimagemmode(no,panel)
Draw temporal mmode view.

drawimagemontage(panel,viashow,marshow)
Main workhorse for creating montage view.
- viashow is if viability data should be updated.
- marshow is if MaR data should be updated.

drawimageno(no)
Call drawimagepanel for the panels that contain no including flow panels. If no is not specified then the current image stack NO is used. This function is typically called when new objects have been created or modified.

drawimageone(panel,viashow,marshow,olshow)
Main workhorse for creating view of one image slice.
- viashow is whether to show viability
- marshow is whether to show MaR
- olshow is whether to show overlays or not.

drawimagepanel(panel)
Draws selected panels, used upon loading or when when major changes have occured such as change of slice, added measurement points etc. This fcn is the true workhorse in graphics etc.

drawimagetypetext(no,pno,panel)

drawimageview(nos,sz,panelstype)
Draws a full view specified by the arguments.

drawinterp(no,panel,olshow)
Initiate handles and draw interpolation points.
Do not work well with overlays.

drawintersectionpoints(no,panel)
Initiate handles and draw intersections with other contours.

drawintersections
Draw intersections between image visible stacks. Uses calcplaneintersection to calculate intersection lines. This function respects view settings.

drawmarhelper(no,panel)
Function to draw MaR contour on screeen used from drawimageslice, drawimagemontage.

drawmeasures(no,panel)
Draw measurements, if available.

drawmontagecontours(no,panel)
Initiate handles and draw contours for montage view.

drawmontageimagetypetext(no,pno,panel)
Initiate handles and draw image type text.

drawmontageinterp(no,panel)
Initiate handles and draw interpolation points for montage view.

drawmontagemeasures(no,panel)
Initiate handles and draw measurements for montage view.

drawmontagepins(no,panel)
Initiate handles and draw pins for montage view.

drawmontagepoints(no,panel)
Initiate handles and draw annotation points for montage view.

drawpins(no,panel)
Initiate handles and draw pins.

drawsliceno(no)
Call updatenopanels.

drawthumbnailframes
Draw frames around thumbnail images.

drawthumbnails(calculatepreview,sliderupdated)
Draw all thumbnails. Calculatepreview is a boolean
indicating if thumbnails needs to be redrawn.

drawviabilityhelper(no,panel)
Function to draw viability contour on screeen used from drawimageslice,
drawimagemontage.

showedits(no)
Show edits (overlays) of scar and mar if current mode.

showmaredits(no)
Show MaR edits on screen as a temporary overlay.

showviabilityedits(no)
Show viability edits on screen as a temporary overlay.

updateannotationpoints(no,panel)
Update coordinates of annotation point handles.

updatecontours(no,panel)
Update coordinates of contour handles of one view panels.

updateglazoomstate(no,ysz)

updateinterp(no,panel)
Update coordinates of interpolation point handles.

updateintersectionpoints(panel)
Draw intersection with segmentation in other image stacks.

updatelongaxiscontours(arg,no,panel)
Update coordinates of contour handles for HLA or VLA image view.

updatemeasures(no,panel)
Update coordinates of measurement handles.

updatemodeldisplay(no,panel)
Update data used to correctly display segmentation in montage view. This fcn needs to be called when the segmentation has changed.

updatemontagecontours(no,panel)
Update coordinates of contour handles for montage image view.

updatemontageinterp(no,panel)
Update coordinates of interpolation point handles for montage view.

updatemontagemeasures(no,panel)
Update coordinates of measurement handles, for montage view.

updatemontagepins(no,panel)
Update coordinates of pin handles for montage view.

updatemontagepoints(no,panel)
Update coordinates of annotation point handles for montage view.

updatemontagerois(no,panel)
Update coordnates of ROI handles for montage view.

updatenopanels(no,stateandicon)
Update panels containing image stack no.

updatepins(no,panel)
Update coordinates of pin handles.

updaterois(no,panel)
Update coordinates of ROI handles.

updatesax3contours(no,panel)
Update coordinates of contour handles for SAX3 image view.

updatevisibility
Make sure visibility of handles correspond to status of hide/show icons.

viewupdateannotext(panel)
Updates the visibility of point/measurement text.
Respects hideX settings.

If point is inbound, pointtext is visible.
If measurementtext is inbound, measurementtext is visible.
If whole ROI is inbound, place XXXX
If only some of ROI is inbound, place at any inbound point
If whole ROI is outside, make ROItext not visible.

viewupdatetextposition(panel)

```
Updates the location of the corner text, to always stay still.
 From drawx(), it is called panelwise, since the linked panels aren't
 ready when the first panel is drawn. From other situations, it is called
 without arguements, and handles the linkage by itself. /JU.
```

15.3 Calculation superunit

The purpose of the calculation unit is to provide general functionality in the Segment platform. This also prevents from code duplication of core functionality such as calculation of volumes.

Interactions

Calls to other superunits are negligible.

Datastructure

The calculation tools uses the `SET` struct and sometimes also use the global variable `NO` for indicating current image stack.

Functions

bsa = calcbsa(weight,height)

```
Calculates BSA. Formula based on Mosteller
weight in kilo and height in cm.
```

calcdatasetpreview

```
Calculate thumbnails. They are stored in the variable
DATA.DATASETPREVIEW. Size of the thumbnails is given by
DATA.GUISettings.ThumbnailSize. It is stored as a RGB image.
```

rad = calcendoradius(no)

```
Calculates endocardial radius. Respects setting in preferences
if to use endocardial or epicardial center as reference. Loops
over timeframes and slices.
```

rad = calcepiradius(no)

```
Calculates epicardial radius. Respects setting if to use
endocardial or epicardial center in calculation. Loops over
timeframes and slices.
```

calcflow(no)

```
calculates flow from ROIs.
```

[meanint,defectextent,varargout] = calcintensityanddefect(im,...
tf,numpoints,numsectors,nprofiles,pos, ...
no,endox,endoy,epix,epiy,sz,resolution,defect,numwidth,timeresolved)

```
Calculates the mean intensity within sectors as a preperation to
generate a bullseye plot.

-im                                    image
-tf                                     timeframe
-numpoint                          numpoints to evaluate in
- numsectors:                     number of sectors
- nprofiles:                           number of profiles
- pos:                                 slices to use
- endox,endoy,epix,epiy: myocardial borders
sz:                                        size of image stack
resolution:                           resolution of image stack
defect:                                  (EH: I do not know what it is, works if emp
numwidth:                          specify if it should calculate several sectors ac
                                           wallthickness. If a scalar it
                                           specifices the number of
                                           layers in the myocardium. If a
                                           two element vector assumes
                                           endo and epi percentages cut.
timeresolved:                     true if timeresolvde.
```

varargout = calclvvolume(no,docomp)

```
Calculate LV volume. Docomp if to use longaxis motion, see below.
Uses area*(thickness+slicedist).
```

calclvvolumepolar(no)

```
Calculate volume of lv when rotated image stacks.
no is the imsage stack. The LV volume calculation includes
longaxis compensation taken from the setting in the GUI and
stored in the SET structure.
```

mantelarea = calcmantelarea(no)

```
Calculate the mantel area in cm2̂ of the LV endocardium in each timeframe.
```

[spacedist,timedist] = calcmmodedists(no)

```
Calculate distances in space and time between mmode lines.
```

[outim,slicestoinclude] = calcmontageviewim(no,matrix,segmentedonly,cmap,c,b,im,tfs,oneextra
```
Calculate view image for montage view.
```

[linex,liney] = calcmontageviewline(no,matrix,inputx,inputy,tf,segmentedonly,tfs,oneextraslice)
```
Calculate view lines for montage view.
```

res = calcmyocardvolume(numsectors,no,tf)
```
Calculate myocardvolume. numsectors is the number
of sectors to calculate in, and no is the image stack.
uses findinmeansectorslice to do some of the calculation.
```

[xofs,yofs] = calcoffset(z,type,no,panel)
```
Calculate offset required to plot coordinates in viewing
mode specified by type.
```

[cellx,celly] = calcoffsetcells(no,panel,type)
```
Same as calculateoffset, but returns cells, used in
updatemodeldisplay.
```

[impos,ormat] = calcormat(no)
```
Calculate orientation matrix for stack number no
Can be used for coordinate transformations:
RLAPFH = impos + ormat * (XYZ - 1)
XYZ = ormat / (RLAPFH - impos) + 1.
```

[x,y] = calcplaneintersections(NO,no,TYPE,type,slice,hslice)
```
Calculate intersections between image planes.
  NO is the viewed plane
  no is the (potentially) intersecting plane
  TYPE is the viewtype of the viewed plane
  type is the viewtype of the intersecting plane
  slice is the slice of the intersecting plane
  hslice is the horizontal slice of the viewed plane (if derived longaxis).
```

radvel = calcradialvelocity(no)
```
Calculate radial velocity of the endocardium.
Forward difference is used.
```

[meanarea,area] = calcroiarea(no,roino,thisframeonly)
```
Calculates roi area (helper fcn).
```

[m,sd,rmin,rmax] = calcroiintensity(no,roino,normalize,thisframeonly)
```
Calculates intensity within a ROI (helper fcn).
```

[rows,cols] = calcrowscols(no,z)
```
Calculate a good montage setup, so maxiumum number of slices
can be displayed on the screen.
```

[varargout] = calcrvvolume(no)
```
Calculate RV volume. no is the image stack.
The RV volume calculation does not involve any longaxis
compensation.
```

calcrvvolumepolar(no)
```
Calculate RV volume when rotated image stacks.
```

[xout,yout] = calcsegmentationintersections(no,type,viewtype)
```
Calculates intersections between segmentation in image stacks.
no is image stack and type is endo or epi, can also be rvendo etc.
the type is a string that dynamically calls a field of the SET struct.
```

ticks = calcticks(num,res)
```
Helper fcn to calculate length of ticks in volumegraph.
```

z = calctruedata(im,no)
```
Calculate true image intensities (as before Segment internal
normalization). Uses IntensityScaling and IntensityOffset stored
in SET structure. im is input image, and no is image stack,
where to take the scaling from.
```

calcvolume(no)
```
Calculate volume of segmentation and updates. Updates both
lv and rv segmentation. Calls subfunctions to do the work.
```

wallthickness = calcwallthickness(sectors,no)
```
Calculate wallthickness. Uses calcendoradius and calcepiradius
to do the work.
```

clearallflow(no)
```
clear all ROI flow from no.
```

clearflow(no,roinbr)
```
clear ROI flow from ROI with roinbr.
```

[window,level] = con2win(contrast,brightness,no)
Convert from contrast to window and level.

[x,y,z] = cyl2cart(xin,yin,slice,numslices,rotationcenter)
Convert from cylindrical to cartesian coordinates.

z = econvn(im,f)
Function to filter image, uses fastest available convolver.

[res,varargout] = findmeaninsector(type,inp,pos,numsectors,no,tf)
Find indicies of points along the contour that corresponds to which sector. Then also calculated mean values of the input inp.

[varargout] = findmeaninsectorslice(type,numpoints,tf,slice,... numsectors,no,endox,endoy,epix,epiy,rotation)
Find indicies of points along the contour that corresponds to which sector. Used when analysing the myocardium of short axis slices. Operates on given slice.

[pos,xdir,ydir,zdir] = getimageposandorientation(no,slice,type)
Return image orientation for current slice or second input argument. If two outputvariables, then zdir is also outputed.

[x,y,z] = gla2sax(xi,yi,no)
Convert image coordinates from GLA to short-axis view.

z = remapuint8(varargin)
Same as remapuint8viewim, but squeezes result to make sure it can be viewed other than in the main window.

z = remapuint8viewim(im,no,tempmap,c,b)
If SET(no).Colormap exists:
Remap from image data to true color using color lookup.
If not:
Remap according to indexed grayscape
Unless:
An external colormap has been supplied, then use that.
This is used to force truecolor grayscale, by passing
true to RETURNMAPPING.
c,b are contrast/brightness settings, used by contrast///_Callback when doing realtime update during mouse motion.

See RETURNMAPPING above.

[xnew,ynew] = resampleclosedcurve(x,y,n,method,distributed)
```
General helper fcn to resample a closed curve.
```

[xnew,ynew] = resamplemodel(x,y,n,method,distributed)
```
Resample the a stack of contours (typically segmentation)
to different number of points (n).
```

map = returnmapping(no,forcetruecolor)
```
If SET(no).Colormap exists:
  Returns truecolor colormap for this.
If not:
  Returns grayscale 1-D indexcolor map.
Unless:
  forcetruecolor is true, then 3-D truecolor grayscale is returned,
  regardless of SET(no).Colormap
  See REMAPUINT8 below.
```

[pos] = rlapfh2xyz(no,rl,ap,fh)
```
Convert from RL,AP,FH coordinates to Segment internal coordinate system.
```

[x,y] = sax2gla(xi,yi,zi,no)
```
Convert image coordinates from short-axis to GLA view.
```

im = truedata2im(z,no)
```
Inverse of calctruedata.
```

volume_helper(no)
```
Helper to find peak ejection, and empty volumes etc.
Lots of checks to prevent NaN or Inf to be presented
when some of the data are missing.
```

[contrast,brightness] = win2con(window,level,no)
```
Inverse of con2win.
```

[pos] = xyz2rlapfh(no,x,y,z)
```
Converts from segment coordinate system to RL,AP,FH coordinate system.
```

15.4 Helper functions superunit

This superunit contains functions that are frequently used to provide other functions with basic information about the state of Segment.

15.4.1 Find unit

The purpose of the Find unit is to contain functions that assist other functions in finding requested image stacks or slices.

Interactions

Calls to other superunits are negligible.

Datastructure

The Find unit tools use the `SET` struct to look for information that is relevant to determining stack/slice content.

Functions

cineshortaxisno = findcineshortaxisno(multiple)

```
Find one [LV] or two [LV and RV] cine short axis stack.
```

shortaxisno = findctshortaxisno(multiple)

```
Find one [LV] or two [LV and RV] CT short axis stack.
```

[flowno,flowroi] = findflowaxisno

```
Find only one flow image stack.
```

tfs = findframeswithsegmentation(type,no)

```
Find timeframes in no containing segmentation of type.
```

[x,y] = findlvcenter(no,slices)

```
Finds the center of the LV, uses the autocrop function as a helper
function.
```

marno = findmarshortaxisno

```
Find only one mar shortaxis stack.
```

[cineno,scarno,flowno,strainno,marno,varargout] = findno

```
Find matching image stacks output
 normalno (normal short axis stack or closest equivalent)
 scarno (viability short axis stacks)
 flowno (flow image stack(s))
 marno (stacks with MaR)
 [stress] stress images
```

```
  Note that all scarno for instance is not garanteed to have non empty
  SET.Scar.
```

[ind] = findoutflowtractslices(no,tfs)

```
  Find slices with outflow tract (a wall thickness that is less than 2mm)
  tfs are the timeframes to search in.
```

scarno = findscarshortaxisno

```
  Find only one scar shortaxis stack.
```

ind = findslicewithendo(no,tfs)

```
  Find slices with endocard in any timeframe.
```

ind = findslicewithendoall(no)

```
  Find slices with endocard in all timeframes.
```

ind = findslicewithepi(no,tfs)

```
  Find slices with endocard in any timeframe.
```

ind = findslicewithepiall(no)

```
  Find slices with endocard in all timeframes.
```

ind = findslicewithmarall(no)

```
  Find and return slices with mar.
```

ind = findslicewithrvendo(no,tfs)

```
  Find slices with RV endocard in any timeframe.
```

ind = findslicewithrvepi(no)

```
  Find slices with RV epicard in any timeframe.
```

ind = findslicewithscarall(no)

```
  Find and return slices with scar.
```

shortaxisno = findspectshortaxisno(multiple)

```
  Find one [LV] or two [LV and RV] CT short axis stack.
```

no = findstack(label)

```
  Finds the stacks with a specific ImageType or ImageViewPlane
  label: string with the ImageType or ImageViewPlane name sought
  no=the stacks with ImageType or ImageViewPlane to the corresponding label.
```

setstack_Callback(type)

```
  open interface for user to manually define flow image stack.
```

15.4.2 Exist unit

The purpose of the Exist unit is to contain functions that assist other functions in determining whether some segmentation content exists in a given slice or stack.

Interactions

Calls to other superunits are negligible. This unit is commonly used in the test script in order to check if operations have been successful.

Datastructure

The Exist unit tools search the `SET` struct for existence of requested information.

Functions

z = anyall(a)
```
Equivalent to z = any(a(:));.
```

y = existendo(no)
```
True if endocardium exist in some slices.
```

y = existendoinselected(no,t,s)
```
True if endocardium exists in selected slices.
If argument is specified, only look in timeframe t.
```

y = existendoinslices(no,t)
```
True for the slices where endocardium exists.
If argument is specified, only look in timeframe t, oterwise look if there
is endocardium in any time frame.
```

y = existendoonlyinedes(no)
```
True if endocardium exists in ED and ES slices.
```

y = existepi(no)
```
True if epicardium exist in some slices.
```

y = existepiinselected(no,t)
```
True if epicardium exist in selected slices.
If argument is specified, only look in timeframe t.
```

y = existepiinslices(no,t)
```
True for the slices where endocardium exists.
If argument is specified, only look in timeframe t, oterwise look if there
is endocardium in any time frame.
```

y = existepionlyinedes(no)
```
True if epicardium exists in ED and ES slices.
```

y = existrvendoinselected(no)
```
True if RV exist in selected slices.
```

y = existrvendoonlyinedes(no)
```
True if RV endocardium exists in ED and ES slices.
```

y = existrvepionlyinedes(no)
```
True if RV epicardium exists in ED and ES slices.
```

15.4.3 Input/Output unit

This unit consits of functions that are internal helper functions in Segment.

Interactions

Interactions are minimal.

Datastructure

Not applicable.

Functions

15.4.4 myadjust.m

```
MYADJUST objecthandle and fighandle/mygui object.
Adjust so that a message box gets to the correct screen.
fighandle is a figure handle
```

15.4.5 mybrowser.m

```
MYBROWSER(URL) Opens an URL platform independent.
```

15.4.6 mycat.m

```
--------------------------------------------------------------------------
```

15.4.7 myclientserver.m

```
MYCLIENTSERVER Class to create interface object to communicate with a
computational server that can perform tasks in background.

B = MYCLIENTSERVER(NAME) Sets up a computational client with the name NAME.

METHODS

DISPLAY
  Overloads display.

SEND(B,FCN,CMD,ARG). Sends a non blocking message.
  FCN is function handle / function that is executed upon return of
  message. The function handle is called with:

  1) Name of slave
  2) Received status, 0 = sucess
  3) Received command
  4) Received arguments, often empty, depends on command

  Note that there is NO buffering of commands, so you can not stack
  multiple commands. If you do not give the slave time to read the message
  before you send another message.

SET(B,'PROPERTY',VALUE)
   Set property values

OK = PING(B,TIMEOUT).
  Pings client.

OK = ALIVE(B).
  Test if server/slave is alive

DELETE(B).
```

```
  Destructor for class. Sends kill command to server/slave.

B = MYCLIENTSERVER instanciate object.

[STATUS,RES,<ARGS>] = SENDBLOCK(B,CMD,ARG).
  Sends a blocking message.

VALUE = GET(B,'PROPERTY')
```

15.4.8 mycopyfile.m

```
MYCOPYFILE Copies a file from source to dest

MYCOPYFILE(SOURCE,DEST)

Should be platform independent and faster than making a system
call. If really large files are used then do a system call instead.
Essentially a wrapper to COPYFILE, kept for naming consistency.

See also MYDEL, MYMKDIR.
```

15.4.9 mycountunique.m

```
MYCOUNTUNIQUE counts the number unique elements in the input vector.

N = MYCOUNTUNIQUE(V,<tol>)

N is the number of unique elements
V is numeric input vector
tol is numeric tolerance, if omitted then set 1e-6.

See also matlab builtin unique.
```

15.4.10 mydel.m

```
MYDEL Delete file from disk

[OK] = MYDEL(filename)

Should be platform independent.
```

```
 See also MYCOPY, RECYCLE.
```

15.4.11 mydir.m

```
 MYDIR Extracts directory information.
 .  and  .. are always first on the list
  thumbs.cache,folders.cache,dicomdir excluded.
  folders comes first
```

15.4.12 mydisp.m

```
 Internal display function
```

15.4.13 mydispexception.m

```
 MYDISPEXCEPTION(ME) Pretty prints a caught
   exception for debugging. ME is matlab error struct
   caught in a try catch clause.

 See documentation on try and catch syntax.
```

15.4.14 myecgreader.m

15.4.15 myexecutableext.m

```
 MYEXECUTABLEEXT Returns proper file extension for exectutables.

 See also, MYREQUIREPC, MYMKDIR, MYMOVEFILE, MYCOPYFILE, MYDEL, MYDIR.
```

15.4.16 myfailed.m

```
 MYFAILED(STRI,FIGHANDLE)
   Displays an error message STRI. FIGHANDLE is
   an optional alignment indicating alignment.
   Difference to errordlg is the alignment and
   possibility to hit return to okey the message.
   Alignment string may be a fig handle or an MYGUI object.

 See also MYWARNING, MYMSGBOX, MYADJUST, MYWAITBARSTART.
```

15.4.17 mygetcurrentpoint.m

```
Returns CurrentPoint of an axes h. In case of macro or testing, returns
current point from buffer.
```

15.4.18 mygetedit.m

```
Get string from an edit box. Use this instead of get(h,'String') to make
macro evaluation possible.
```

15.4.19 mygetframe.m

```
Same as getframe, but with error checking for multiple screens
Also makes sure that the frame is visible on screen
Einar Heiberg
```

15.4.20 mygetlistbox.m

```
Get value from a listbox handle. Used instead of normal get to also be able
to use macros in Segment.
```

15.4.21 mygetnumber.m

```
Input dialog function that get a scalar value.
Syntax
  MYGETNUMBER(STRI,DEFAULT,MIN,MAX)

Where:
 - PROMPTSTRI prompt for dialog.
 - TITLESTRI title of dialog box.
 - DEFAULT is default value.
 - <MIN>, optional minimal value.
 - <MAX>, optional maximal value.
```

15.4.22 mygetslider.m

```
Get value from a slider handle. Used instead of normal get to also be able
to use macros in Segment.
```

15.4.23 mygetvalue.m

```
Get value from a slider handle. Used instead of normal get to also be able
to use macros in Segment.
```

15.4.24 mygui.m

```
MYGUI Class for handeling GUI's in a efficeint manner

G = MYGUI(FIGFILENAME,<'BLOCKING'>);

The purpose of this class is to facilitate to generate
GUI's than can also work as a temporary containter data
for that so that data transfer between subfunctions becomes
simple. If blocking is specified then segment internal blocking
figure mechanism will be used.

The function also keeps track of position for the gui to
be saved later and to be able to set position of messageboxes,
called with for example mywarning, mymsgbox and yesno when
calling with last input argument as the mygui object.

Example:
 First, reserve space for it in DATA Structure:

  DATA.GUI.MyGUIExample = []; \%in segment.m

In your code to initialize:
  gui = mygui('myguiexample.fig');

In your subfunctions:
  global DATA

  gui = DATA.GUI.MyGuiExample;

Now you can use gui as an ordinary variable, and everthing
is stored in the GUI, and will be available to other
subfunctions until the GUI is deleted. Some examples:

  gui.myfieldname = somearray;
  gui.myfieldname1.myfieldname2.myfieldname3 = 134;
  temp = gui.myfieldname(2);
  temp = get(gui.handles.text2,'string');
```

```
You can also use the gui to get good alignment of the
msgboxes mywarning, myfaield, mymsgbox, mymenu, mywaitbarstart and yesno.

 myfailed('An error has occured.',gui);

When closing the gui the close function can be called with
 DATA.GUI.MyGUIExample=close(DATA.GUI.MyGUIExample);
 when closing by using this syntax the position of the gui
 will be saved in the struct DATA.GUIPositions

The position of the gui can also by saved by
 saveguiposition(gui);
 this could be used in a resize function

See also SUBSREF, SUBSASGN.
```

15.4.25 myguide.m

```
MYGUIDE  Same as GUIDE, but can also open .fig files located in package
     directories without having to change directory manually
```

15.4.26 myicon.m

```
MYICON add icon to MYICONPLACEHOLDER
 Constructor takes (axes,image,text,exestring,dentstick (Says if the button s
Klas Berggren  and  Einar Heiberg
```

15.4.27 myiconplaceholder.m

```
MYICONPLACEHOLDER Class to handle icons
```

15.4.28 mymaximize.m

```
MYMAXIMIZE(HANDLE) Maximizes a figure or a mygui object
and aligns it to correct monitor.
```

15.4.29 mymenu.m

```
m = MYMENU(header,options,defaultstring)
m = MYMENU(header,item1,item2,item3,...)
```

```
Same as menu, but with the following additions:
- more elegant! uses keyboard to select items.
- modal display
- can use default string
- last optional argument is figure handle
```

15.4.30 mymkdir.m

```
MYMKDIR(NEWDIR)
[SUCESS] = MYMKDIR(NEWDIR)

Make directory, difference to MKDIR is that
is does not issue an error message
if the folder already exists.

See also MYCOPY, MKDIR
```

15.4.31 mymovefile.m

```
[STATUS,MESSAGE,MESSAGEID] = MOVEFILE(SOURCE,DESTINATION,MODE)
moves the file in source to dest.

Essentially a wrapper for MOVEFILE, kept for naming consistency.

See also MYCOPYFILE, MOVEFILE.
```

15.4.32 mymsgbox.m

```
MYMSGBOX Helper function, works as MSGBOX

MYMSGBOX(STRI,TITLE,[ALIGNMENT])
  STRI String to display
  TITLE Title of messagebox
  FIGHANDLE Optional figure handle indicating alignment

See also MYWARNING, MYMSGBOX, MYADJUST, MYWAITBARSTART.
```

15.4.33 mynanmean.m

```
NANMEAN Mean value, ignoring NaNs.
The same function as Matlabs inbuild function nanmean (duplicated to not
need statistical toolbox)
   M = NANMEAN(X) returns the sample mean of X, treating NaNs as missing
   values.  For vector input, M is the mean value of the non-NaN elements
   in X.  For matrix input, M is a row vector containing the mean value of
   non-NaN elements in each column.  For N-D arrays, NANMEAN operates
   along the first non-singleton dimension.
   NANMEAN(X,DIM) takes the mean along dimension DIM of X.
```

15.4.34 mypoly2cw.m

```
Finds clockwise orientation of segmentation
Written by Klas.
```

15.4.35 myrequirepc.m

```
Z = MYREQUIREPC Returns true if PC platform.
  If not PC platform, then an error messsage is
  displayed. Used to block certain functions
  that is not implemented for other platforms
  than PC.
```

15.4.36 myset.m

```
MYSET Same as SET, but removes NaN's in the handles.
```

15.4.37 mystubfailed.m

```
Displays generic error message from stubs
```

15.4.38 myuigetdir.m

```
[PATHNAME,OK] = MYUIGETDIR(PATHNAME,TITLESTRI)
Corresponding to uigetdir, but also fixes macro recording and test
scripts.
```

15.4.39 myuigetfile.m

```
[FILENAME, PATHNAME, FILTERINDEX,OK] = MYUIGETFILE(FILTERSPEC, TITLE)
Corresponding to uigetfile, but also fixes macro recording and test
scripts.
```

15.4.40 myuiputfile.m

```
[FILENAME, PATHNAME, FILTERINDEX,OK] = MYUIPUTFILE(FILTERSPEC, TITLE)
Corresponding to uiputfile, but also fixes macro recording and test
scripts.
```

15.4.41 myurlread.m

```
S = MYURLREAD('URL',timeout) reads the content at a URL into a string, S.  If the
    server returns binary data, the string will contain garbage.

    S = MYURLREAD('URL',timeout,'method',PARAMS) passes information to the server as
    part of the request.  The 'method' can be 'get', or 'post' and PARAMS is a
    cell array of param/value pairs.

    [S,STATUS] = MYURLREAD(...) catches any errors and returns 1 if the file
    downloaded successfully and 0 otherwise.

Same as URLREAD, except that it adds two timeout settings
```

15.4.42 mywaitbarclose.m

```
MYWAITBARCLOSE

Closes a mywaitbar structure, and removes waitbar.

See also MYWAITBARSTART, MYWAITBARUPDATE.
```

15.4.43 mywaitbarmainclose.m

```
MYWAITBARCLOSE

Closes a mywaitbar structure, and removes waitbar.

See also MYWAITBARSTART, MYWAITBARUPDATE.
```

15.4.44 mywaitbarmainstart.m

```
H = MYWAITBARSTART(NUMITERATIONS,STRING,[IGNORELIMIT],FIGHANDLE)

Similar to waitbar, but first input argument is
the total number of iterations that will be performed.
Second input argument is string that will be displayed,
and third optional argument is a limit that if fewer
iterations that this is performed, then there will be
no graphical output. Another difference to standard waitbar
is that no more than 20 graphical steps are displayed to
minimize CPU consumption. Function returns a handle to a
MYWAITBAROBJECT. FIGHANDLE is optional and indicates alignment

See also MYWAITBARUPDATE, MYWAITBARCLOSE.
```

15.4.45 mywaitbarmainupdate.m

```
H = MYWAITBARDATE(H)

Update a mywaitbarstructure, and increment
counter one step. Graphic is updated depends on
setting when intializing the structure.

See also MYWAITBARSTART, MYWAITBARCLOSE.
```

15.4.46 mywaitbarstart.m

```
H = MYWAITBARSTART(NUMITERATIONS,STRING,[IGNORELIMIT],FIGHANDLE)

Similar to waitbar, but first input argument is
the total number of iterations that will be performed.
Second input argument is string that will be displayed,
and third optional argument is a limit that if fewer
iterations that this is performed, then there will be
no graphical output. Another difference to standard waitbar
is that no more than 20 graphical steps are displayed to
minimize CPU consumption. Function returns a handle to a
MYWAITBAROBJECT. FIGHANDLE is optional and indicates alignment
```

```
 See also MYWAITBARUPDATE, MYWAITBARCLOSE.
```

15.4.47 mywaitbarupdate.m

```
 H = MYWAITBARDATE(H)

 Update a mywaitbarstructure, and increment
 counter one step. Graphic is updated depends on
 setting when intializing the structure.

 See also MYWAITBARSTART, MYWAITBARCLOSE.
```

15.4.48 mywarning.m

```
 MYWARNING(STRI,HANDLE) Displays a warning message
   User needs to click OK to discard message.

 See also  MYWARNINGNOBLOCK
```

15.4.49 mywarningnoblock.m

```
 MYWARNING(STRI,HANDLE) Displays a warning message
   User needs to click OK to discard message.

 See also  MYWARNINGNOBLOCK
```

15.4.50 myworkoff.m

```
 MYWORKOFF Graphically show that calculation is
   finished by restoring pointer.
```

15.4.51 myworkon.m

```
 MYWORKON Graphically indicate that Segment is
   busy by showing watch pointer.
```

15.5 Report superunit

The Report superunit contains functions that report data to the user. Some of these functions are of higher order and use others to extract data to be

used as a subreport. An overview of the units and how they communicate is provided in Figure 4.

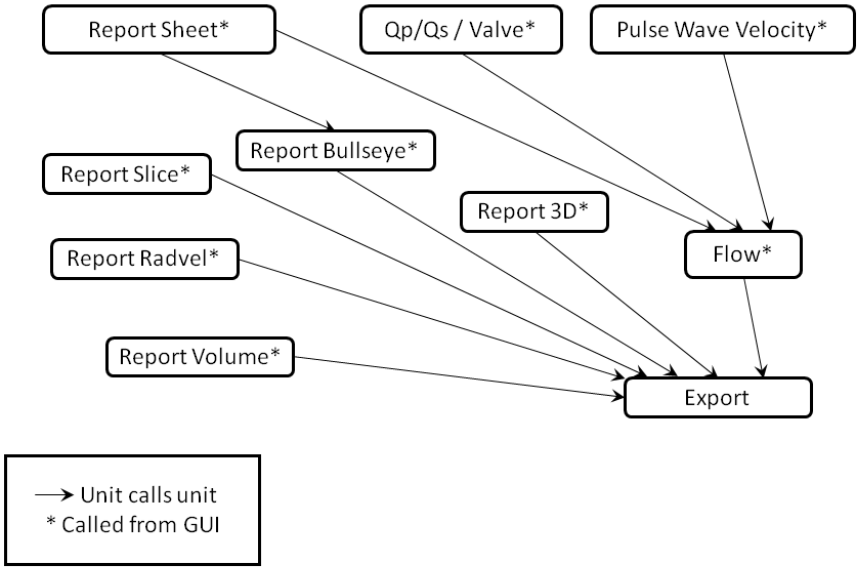

Figure 4: Overview of the internal structure of the `Report` function superunit and transaction analysis.

15.5.1 Report 3D unit

The purpose of the Report 3D unit is to do 3D model reports.

Interactions

The Export unit is called for export of data.

Datastructure

The information used to build a 3D model is contained in `SET` struct. Number of stack to report is taken from `NO` variable.

Functions

init
`initiating the 3D model interface.`

report3dmodelrotated
`Display 3D model of rotated image stacks.`

rotatedplothelper(xc,yc,linecolor)
`Helper function to display segmentation in rotated image stacks.`

15.5.2 Report Volume unit

The purpose of the Report Volume unit is to do volume reports.

Interactions

The Export unit is called for export of data.

Datastructure

The volume information is taken from the `SET` struct. Number of stack to report is sometimes taken from the `NO` variable.

Functions

curve_Callback(no)
`GUI for reporting the volume curve.`

loop_Callback
`Reporter for volume loop (not yet implemented).`

15.5.3 Report Radial Velocity unit

The purpose of the Report Radial Velocity unit is to do reports of radial velocity.

Interactions

Suitable image stacks are found by the Find unit and the calculations that generate the reported results are performed by the Calculation unit. The Export unit is called for export of data.

Datastructure

The radial velocity information is taken from the `SET` struct. Number of stack to report is taken from the `NO` variable.

Functions

helpimageposneg(im,sector,konst)
`Helper function to display image.`

15.5.4 Flow unit

The purpose of the Flow unit is to quantify and report flow data from phase contrast images.

Interactions

This unit interacts with the Calculation superunit for assistance in some calculations. The Export unit is called for export of data.

Datastructure

The data used for flow analysis is contained in the `Flow` field of the `SET` struct.

Functions

expandflowroi(no,m,d)
`Expand the roi M outwards by D mm's.`

flow3dmovie_Callback(arg,scaling,no,up,down,left,right)
`Displays 3D movie of velocity in a vessel.`

flowcreate_helper(angio)
`Helper function to create flow derived image stacks.`

flowcreateangio_Callback
`Create angio image.`

flowcreatevelmag_Callback
`Create velocity magnitude image.`

flowdelete_helper(angio)
Helper function to delete angio images.

flowdeleteangio_Callback
Delete angio image.

flowdeletevelmag_Callback
Delete velocity magnitude image.

floweddycurrent_Callback(arg)
Callback for menu selection to do eddy current compensation.

[ok,pulmno,pulmroi,aortno,aortroi] = flowfindqpqs
Find data sets with pulmonary and aortic flow, to use for Qp/Qs analysis.

flowfix(n)
temp function to flip 2d flow directions.

flowflipdirs_Callback
Flips LR and UP direction.

flowflipleftright_Callback
Flips up/down direction'.

flowflipupdown_Callback
Flips up/down direction'.

flowplotquiver_Callback
Plots a quiver plot of current image stack.

flowpropagate_Callback(m)
Propagate a flow ROI to next time frame and refine.

flowrefine_Callback(m,silent)
Refine segmentation of a flow ROI.

flowsegmentroi_Callback(name)
Do segmentation of a flow roi.

flowsetvenc_Callback
Set VENC for current image stack.

flowshrink_Callback(m,factor)
Shrink a flow ROI.

flowtrackroi_Callback(m)
Track a flow ROI through all time frames.

[varargout] = flowvesselenhancement(no)
Beta function to show vessels with time depending flow.

setflowpreferences_Callback
open a box where it is possibly to set the ROI propagation parameters: balloon force, edge force and curve force.

close_Callback
Close flow report GUI.

export
Export flow data to clipboard.

flowbar_Buttondown(type)
Button down function for flow bar in flow report GUI.

flowbar_Buttonup(type)
Button up function for flow bar in flow report GUI.

flowbar_Motion
Motion function for flowbar.

varargout = getdata(arg)
Output requested data from flow calculations.

ok = init(no)
Init flow report GUI.

recalculate(no)
Calculate flow data.

update
Make graphical update of plot.

zoom
Callback for checkbox toggling zoom on/off.

15.5.5 Report Slice unit

The purpose of the Report Slice unit is to do slice based reports of regional function.

Interactions

The Export unit is called for export of data.

Datastructure

The regional function information is taken from the `SET` struct. Number of stack to report is taken from the `NO` variable.

Functions

close_Callback
`Close GUI.`

export_Callback
`Export data to clipboard. Renamed to avoid confusion with export.m.`

frac
`Callback upon changing frac/wall radiobutton selection to frac.`

frameupdate
`Different timeframe, update less.`

init
`Initiate GUI.`

next
`Callback to go to next frame.`

parameter
`Update things upon changed selection in parameter listbox.`

play
`Callback for video playback.`

prev
`Callback to go to previous frame.`

sector
`Update plots upon changed sector rotation.`

sectors
`Update upon changing number of sectors.`

slice
`Different slice, update a lot of things.`

update
`Update GUI.`

wall
`Callback upon changing frac/wall radiobutton selection to wall.`

15.5.6 Report Bullseye unit

The purpose of the Bullseye unit is to be able to present quantifiable segmentation parameters as bullseye plots. Derived variables from LV segmentation, viability can be presented. To plot and not color the apical sector alphamapping is used. This requires a renderer that supports this alpha mapping.

The algorithm to convert to a 17 segment model is performed by replicating each slice into three separate slices with the slice thickness that is one third of the original slice thickness. Then the total number of slices are always dividable by three. The slices are divided into equal thirds, for basal, mid, and apical parts.

Interactions

Interaction with the calculation unit to perform underlying calculations. The Export unit is called for export of data.

Datastructure

The data in the unit is stored as a mygui object with the following fields:

- `volumeconsistent`, boolean and true if volume consistent checkbox is enabled.
- `numsectors`, number of sectors in the bullseye plot.

- `slice`, for which the image with spoke wheel is plotted.
- `outdata`, variable extracted with data (from `SET`) depending on user selection.
- `ahaoutdata`, variable with extracted data in aha 17 segment format.

Functions

[stri,pos] = aha17nameandpos(i)

```
Get name of 17 segment.
```

im = bullseye(m,ax,n,vc,no,tf)

```
Calculate bullseye from matrix m. Ax is optional axis where the output
image im should be displayed.
m:   values in the sectors
ax:  figure axes
n:   number of pixels in the resulting image
vc:  volume consistent (true or false)
im:  resulting bullseye image.
```

[varargout] = bullseye2(m,ax,n,flipx,vc,no)

```
Generate bullesye data. Optional output argument is im
m:     values in the sectors
ax:    figure axes
n:     number of pixels in the resulting image
flipx: flip in x-direction (true or false)
vc:    volume consistent (true or false).
```

[varargout] = bullseyeaha(m,ax,n,valuetype)

```
Calculate and/or plot AHA 17 segment model.
- m is a matrix in polar coordinates. It could also be a vector of 17
  segments. Please note in such cases then the order is not the same as
  the standard AHA numbering. The names are given by the function
  aha17nameandpos.

- ax is axis to plot it in.
- n is the number of pixels
- valuetype is how to treat the merging of data into sectors; mean, sum,
  max.
```

bullseyelistbox_Callback

```
Callback for bullseye listbox. Updates plot.
```

close_Callback
Properly close the GUI.

colormaplistbox_Callback
Callback for colormap listbox. Updates plot.

defineslices(sign,part)
manual define slices to include in the bullsye plot.

drawsectorimage
Draw image of sector division in new figure.

endocenter
Called when the endocenter checkbox is clicked. Updates SET.EndoCenter.

varargout = export
Export data to clipboard.

value = getdata(type)
Helper function to extract data from the module. Used for instance from reportsheet generator.

init
Initialize the GUI.

initlaximage

invertcolors_Callback
Callback for invert colors checkbox. Updates plot.

maxedit_Callback
Callback for edit to change max value. Updates plot.

minedit_Callback
Callback for edit to change min value. Updates plot.

nedit_Callback
Callback for edit to change value of n. Updates plot.

plotmethodpanel_SelectionChange
Callback for changing type of plot (sector, smooth or AHA model).

rotationfromannotation_Callback
Finds rotation by looking at RV insertion points.

rotationslider_Callback
Callback for rotation slider. Update slice image.

pos = sectorrotationhelper(no)
Find suitable sector rotation based on RV insertion points.

sectors
called when number of sectors is updated.

separatewindow_Callback
Callback for separate window checkbox. Updates plot.

setdata(type,datain)
interface function to gui to set data from code, for instance reportsheet generator.

sliceslider_Callback
Callback for slider to toggle slice.

startbullseye
start up the bullseye analysis.

thissliceonly_Callback
Callback for this slice only checkbox.

updateall
Update entire GUI, calls rotationslide and listbox. Thats all.

updatelongaxisimage
Update which slices are active in the longaxis image.

updateplot
Plot different type of data. Calculate / retrieve data, and perform graphical update. This is a main workhorse.

updatepushbutton_Callback
Callback for update pushbutton. Updates plot.

updatesliceimage(plotseperate)
Changed the rotationslider, new slice image.

volumeconsistent
Called when volumeconsistent checkbox is called.

15.5.7 Pulse Wave Velocity unit

The purpose of the Pulse Wave Velocity unit is to provide the user with tools for calculating and reporting pulse wave velocity data, defined as the user-defined distance between two planes of aortic flow images divided by the separation in time of the flow curve upslopes of these flow images.

Interactions

This unit calls the Flow quantification unit to access the flow data used to determine separation in time of the pulse wave.

Datastructure

The measurement information is taken from the `Measure` field and the ROI information for flow analysis from the `Roi` field of the `SET` struct.

Functions

close_Callback
`Close Pulse Wave Velocity GUI.`

export
`Export data to clipboard.`

getflows
`Get values of flow calculation from Flow report.`

init
`Initiate Pulse Wave Velocity GUI.`

sigmaslider_Callback
`Callback for when using slider to change the value of sigma.`

updateaorticimage
`Update full view image of aorta.`

updateflowplots
`Update plot of flow curves.`

15.5.8 Qp/Qs / Valve unit

The purpose of the Qp/Qs / Valve unit is to provide the user with tools for reporting Qp/Qs and valvular analysis.

Interactions

This unit calls the Flow quantification unit to access the flow data used to determine Qp/Qs or regurgitant fractions.

Datastructure

The ROI information for flow analysis is taken from the `Roi` field of the `SET` struct. No datastructure is updated by this unit.

Functions

[svaorta,svpulmo,fwaorta,bwaorta,fwpulmo,bwpulmo] = getsv

```
Calculate and return stroke, forward and backward volumes both for the
aorta and the pulmonary artery.
```

result = qpqs

```
Calculate Qp/Qs and display result in a message box.
```

[mitrfrac,tricfrac,mitrvol,tricvol] = regurg

```
Calculate regurgitant fractions for mitralis and tricusp.
```

15.5.9 Unwrap unit

The purpose of the Unwrap unit is to provide the user with tools for performing correction of wrapped phase values in phase contrast image stacks.

Interactions

Interaction with the calculation unit to perform underlying calculations.

Datastructure

The VENC information for flow analysis is taken from the `VENC` field of the `SET` struct. The `SET.VENC` field is updated by this unit.

Functions

applyandexitpushbutton_Callback
```
Apply and Exit push button.

 Saves the phase images to where they came from, overwriting.
 and using the VENC, double relative the original.
```

autounwrappushbutton_Callback
```
Perform automatic phase unwrapping on all flows.
```

cancelpushbutton_Callback
```
Cancel pushbutton: Exit window, save nothing.
```

flows = checkflowdata(NO_to_check)
```
Check what flow data, if any, exists and return an array of stack
 numbers.
```

closewindow
```
Everything that is common to the 'Apply and Exit' and 'Cancel' buttons.
```

flowdirectionlistbox_Callback
```
Callback for the flow direction listbox.
```

pixel_over_time = getcurrentpixelovertime
```
Returns the currently selected pixel over time, in units of the
 N.B: ORIGINAL VENC.
```

tms = getcurrenttime
```
Returns the current time in ms.
```

slice = getcurrentvelocityslice
```
Returns the current velocity slice.
```

[X, Y] = getoverlayplotcoords
```
Computes the coordinate vectors for the overlay plot

 Corners:  (x-0.5, y+0.5)
           (x+0.5, y+0.5)
           (x+0.5, y-0.5)
           (x-0.5, y-0.5)
           (1st again).
```

[X, Y] = gettimeframemarkercoords
Computes the timeframe marker (vertical line in temporal plot) coordinates.

init(force)
Initialize the Flow Unwrap GUI.

initializeflowdirectionlistbox
Initializes the listbox where flow direction can be selected.

initializegui
Performs startup tasks, like setting slider limits. A full update is also done after this initialization, so there is no need to duplicate code in both.

initializehidejumpsizeselect
Hides the jump size selector.

initializehideunnecessary
Hide unnecessary UI elements:

* If there is only one slice, hide slice selection.

* If zoom is 0, hide pan sliders.

initializesliders
Initializes the sliders.

initializetemporalplot
Initializes the temporal plot.

initializevelocityimage
Initializes the velocity image.

jumpsizebuttongroup_Callback(src, event)
Jump size selector callback.

keypress_Callback(src, event)
Handles keypress events

```
Up      30  Slice -
Down    31  Slice +
Left    28  Timestep +
Right   29  Timestep -

wW      119 Pixel selector up
aA      97  Pixel selector left
sS      115 Pixel selector down
dD      100 Pixel selector right

space   32  Play current slice

shift       Wrap up
control     Wrap down.
```

movecurrentpixel(direction)
Moves the current pixel in requested direction, but stops at boundaries.

panimageslider_Callback
Callback for the pan sliders.

playcurrentslice
Plays the current slice.

refocus
Gives back focus to slider4 - which has keypressedfcn set to the callback handling all keyboard shortcuts. Of course we would like to give focus to the figure itself and let the figure handle the keypresses, but matlab won't let us do that. Yes, this is a hack to get around a matlab limitation.

slider4 is placed at (-100, -100), so it shouldn't cause any trouble.

sliceslider_Callback
Slice slider callback.

timeframeslider_Callback
Timeframe slider callback.

unwrapcurrentpixel(updown)
Unwrap current pixel in specified direction.

unwrapped = unwraptimeseries(wrapped, jumpsize, varargin)

```
  function unwrapped = unwraptimeseries(wrapped, jumpsize, jumpfraction,
                                        debug)

   Unwraps a timeseries of samples.

   unwrapped:  The unwrapped phase

   wrapped:      The wrapped phase
   jumpsize:     Size of the expected jumps in the data
   jumpfraction: Fraction of jumpsize to consider a jump. Default is 0.5.
   debug:        Perform sanity checks on input. Default is true, but can be
                 set to false for speed.

   Example usage, when SET(NO) is a phase stack:

   SET(NO).IM(x,y,:,slice) = ...
         unwraptimeseries(SET(NO).IM(x,y,:,slice), 1, 0.5).
```

updatecurrentpointoverlay

Updates the current point overlay (red square in velocity image).

updatefullgui

Updates all gui elements to reflect current state.

updatepanimagesliders

Updates the pan image sliders - invisible if zoomlevel is 0.

updatesliders

Updates the time and slice sliders to reflect current positions.

updatetemporalplotaxes

Updates the temporal plot of the current pixel.

updatetextelements

Updates all text elements in the GUI.

updatevelocityimage

Updates the axes showing the velocity image.

updatevelocityimagezoomandpan

Updates zoom and pan in the velocity image.

velocityimage_ButtonDown
`Callback for clicks in velocityimage.`

zoompushbutton_Callback(inout)
`Zoom pushbuttons. Change zoom level, update GUI.`

15.5.10 Report Sheet unit

This unit is used to generate a comprehensive patient report. It consists of the m-file `reportsheet.m`, the superclass definition `htmlgenerator.m` used for formatting the output into a HTML page, and its subclass definition `figgenerator.m` used for formatting the output into Matlab figures that can then be converted into image files for exporting data to PAF or PACS.

Interactions

This unit calls other units from the Report superunit to insert their data into the report sheet. One such unit is the Flow unit. Another such unit is the Report Bullseye unit. The File superunit is called to generate reports in DICOM format and upload them to PACS.

Datastructure

This unit uses the field `Report` of the `SET` structure to contain information of what to include in the report, and the field `PatientInfo` to contain patient information used in the report.

Functions

ECV_Callback

IncludeStrain_Callback
`Callback from Strain checkbox.`

addECV(report)
`adds ecv results to reporter.`

adddistances(report)
`Adds distance measurements to report.`

addflowanalysis(report, floop, dohtml)
Add flow analysis from image stack 'floop' to report.

addfreetext(report)
Add free text patient evaluation to report.

stri = addimagesax(report)
Add shortaxis image to report.

strlax = addimageslax(report,dohtml)
Add table of longaxis images to report.

addlvanalysis(report,dohtml)
Add LV analysis to report.

addmaranalysis(report, dohtml)
Add MaR analysis to report.

addpatientdata(report)
Add patient demographics to report.

addperfusion(report)
Add perfusion analysis to report.

addrvanalysis(report)
Add RV analysis to report.

addscaranalysis(report, dohtml)
Add scar analysis to report.

addscreenshot(report,dohtml)
Add screenshot image/s to report.

addshuntvalveanalysis(report)
Add Qp/Qs and Shunt and Valve analysis to report.

addstrain(report,dohtml)
Add Strain analysis/images to report.

addt2star(report)
Add T2* analysis/image to report.

addwallthickness(report)
```
Add wallthickness analysis to report.
```

age = calcage(AcqDate,BirthDate)
```
calculate patient age by using Aqcuisition date and Birth date.
```

close_Callback
```
Close reportsheet GUI.
```

distance_Callback
```
Callback from distance measurements checkbox.
```

flowanalysis_Callback
```
Callback from flow analysis checkbox.
```

generatereport_Callback(arg)
```
This is the main function that actually makes the report.
```

[valuecells,outlier] = generatevaluecells(value,refvalues)
```
Creates data cells used for LV data table. Also returns an indication of
whether a value is outside the range of the reference.
```

gensvar_Callback
```
This callback opens a GUI for sending report entries to Gensvar.
```

gensvarcancel_Callback
```
Callback for cancel of sending report entries to Gensvar.
```

gensvarok_Callback
```
Callback to approve sending of report entries to Gensvar.
```

gensvarpasswordedit_KeyPress(hObject,eventdata)
```
Callback activated when changing Gensvar password.
```

image_Callback(imagetype)
```
Callback from image checkboxes (all of them).
```

init
```
Initialize GUI.
```

initreferencelistbox(modality)
```
Initiates listbox of reference values.
```

lvanalysis_Callback
Callback from LV analysis checkbox.

maranalysis_Callback
Callback from MaR analysis checkbox.

perfusion_Callback
Callback from perfusion analysis checkbox.

stri = removehats(stri)

reset_Callback
Sets all checkboxes to false and empties text edits, reinserts template.

resetall_Callback
Resets the entire report.

rvanalysis_Callback
Callback from RV analysis checkbox.

scaranalysis_Callback
Callback from scar analysis checkbox.

screenshot_Callback
Callback from screenshot checkbox.

sendtopacs_Callback
Callback for making a report, saving it as a DICOM, and sending it to PACS.

shuntvalve_Callback
Callback from Shunt and Valve analysis checkbox.

t2star_Callback
Callback from T2* analysis checkbox.

textedit_Callback
Callback from comments textedit.

update
Update SET struct with changes in user interface.

wallthickness_Callback
Callback from wallthickness analysis checkbox.

stri = box(hg, text, width)
BOX Method to insert a text box.

stri = columns(hg, varargin)
COLUMNS Method to insert a table without borders, used for dividing input into columns.

stri = conc(varargin)
Concatenate two or more function calls.

gsstri = ftext(hg)
FTEXT Method to write a paragraph of text with formatted headlines.

stri = headline(hg, headtext, prio)
HEADLINE Method to write a headline in bold.

stri = hline(hg)
HLINE Method to write a horizontal line.

stri = html_footer(hg)
Return string used as HTML footer.

stri = html_header(hg)
Return string used as HTML header.

hg = htmlgenerator(title, pathname, filename, pagewidth)
Constructor.

stri = image(hg, imgname, imgsource)
IMAGE Method to add an image. Also stores it to disk in the same folder as the HTML file. imgsource can be a location or an image matrix.

stri = link(hg, ref, text)
LINK Method to write a link to a file or URL.

stri = newline(hg)
NEWLINE Method to insert a line break.

newpagepos = pagebreakchk(hg, pagepos, nextblock)
```
  PAGEBREAKCHK Method to insert a page break
  Checks if it is time to insert a page break using 'pagepos' parameter.
```

hg = start(hg)
```
  START Method to open the HTML file and write headers.
```

hg = stop(hg)
```
  STOP Method to write necessary HTML footers and close file.
```

stri = table(hg, content, boldcells, width)
```
  TABLE Method to insert a table with content specified by a cell.
```

stri = text(hg, paragraph, alignment)
```
  TEXT Method to write a paragraph of text.
```

write(hg, str)
```
  Writes a string of characters to file.
```

columns(fg, varargin)
```
  COLUMNS Method for dividing input into columns.
```

call = conc(varargin)
```
  Concatenate two or more function calls.
```

fg = figgenerator(title, pathname, filename, pagewidth, lineht)
```
  Constructor.
```

gsstri = ftext(fg)
```
  FTEXT Method to write a paragraph of text with formatted headlines.
```

call = headline(fg, headtext, lpos)
```
  HEADLINE Method to write a headline in bold.
```

hline(fg)
```
  HLINE Method to write a horizontal line.
```

call = image(fg, imgname, imgsource, lpos)
```
  IMAGE Method to add an image. imgsource can be a
  location or an image matrix.
```

call = newline(fg)
```
NEWLINE Method to insert a line break.
```

newpagepos = pagebreakchk(fg, pagepos, nextblock)
```
PAGEBREAKCHK Method to insert a page break
Checks if it is time to insert a page break using 'pagepos' parameter.
```

start(fg, figtitle)
```
START Method to open a new figure.
```

stop(fg)
```
STOP Save all figures to image files, then close them.
```

call = table(fg, content, boldcells, width, lpos)
```
TABLE Method to insert a table with content specified by a cell.
```

call = text(fg, paragraph, alignment, lpos, weight)
```
TEXT Method to write a paragraph of text.
```

zoom(fg)
```
ZOOM Zoom to current page size.
```

15.5.11 Export unit

The exporting tools exports data from the `SET` structure or `.mat` files to the clipboard.

Interactions

The functions in the Calculation superunit are used frequently and the function `cell2clipboard` from the Main superunit is used extensibly.

Datastructure

The exporting routines use the `SET` structure.

Functions

export2stl_Callback
```
Exports mesh in current image stack as STL file. Asks for surface to
export and then takes current timeframe to export.
```

export2stl_helper(no,x,y,resolution,tf,closeapex,fignr,pathname,filename)
```
Helper function that fixes with coordinates etc before exporting.
```

exportall2stl(no,pathname,filetemplate,fignr,resolution,tf)
```
Helper function to exportall2stl.
```

exportall2stl_Callback
```
Exports all contours and ROI's to stl file(s)

LV endo  and  epi  gives  left ventricle
RV endo  gives  right ventricle with 1mm thickness
RV epi  gives  left atrium closed in base

ROI's  gives  tubes with 1 mm thickness.
```

exportbatch2stl_Callback
```
Export all2stl for multiple .mat files.
```

exportclosesurfaces(no1,x1,y1,no2,x2,y2,closebase,closeapex,resolution,fignr,fid)
```
Helper function to close to surfaces (basal) and export to STL file.
Please not that this takes a mesh in form of points * slices (i.e
timeframe already selected. This is different from elsewere in STL export.
no1 and no2 are required to be able to calculate 3D coordinates (patient
system).
```

exportcontour_Callback
```
Export contour. Ask what contour to take.
```

varargout = exportdata(doheader,includenormalized,no)
```
This is the workhorse of export functions.
- doheader tells whether to include a header.
- includenormalized tells whether to include BSA normalized data.
- no tells whether to export ONLY image stack no.
```

exportendoepitoensight_Callback
```
Exports endocardium and epicardium in Ensight format.
```

exportimage_Callback(image2store)
Export an image to an image file. If no input image exist, export current image wi

exportleftatria2stl(no,tf,fignr,filename,resolution)
Exports RV epicardium surface to STL but names it LV atria. Closed in
base. Wall thickness is set to 1mm.

exportlv2ensight_Callback
Export left ventricle to Ensight.

exportlv2stl(no,tf,fignr,filename,resolution)
Exports LV surface to STL (with inner and outer contour, closed in apex
and surfaces merged in the base.

exportlv2stl_Callback
Exports LV (endo and epi) as one surface and takes care of closing the
surface suitable to read into CAD software.

exportmovie_Callback
Function to export a movie without contours. This is a quick method to
generate a movie of the current image stack.

exportmovierecorder_Callback(arg)
Movie recorder GUI.

exportmultiplePerf_Callback
Creaty summary of Perfusion
from multiple matfiles in one folder.
This function is very useful for research.

exportmultiplePerfsplit_Callback
Creaty summary of Perfusion split
from multiple matfiles in one folder.
This function is very useful for research.

exportmultipleROI_Callback(dosegdicom)
Creaty summary of multiple matfiles in one folder
This function is very useful for research. The user
performs all delineations and then exports all data to
one spread sheet.

exportmultiple_Callback(dosegdicom)

```
Creaty summary of multiple matfiles in one folder
This function is very useful for research. The user
performs all delineations and then exports all data to
one spread sheet.
```

exportmultipleinfo_Callback

```
Exports information of image stacks for a folder of mat files.
This function is useful for debugging and checking purposes of
the integrity of multiple .mat files.
```

exportmultiplestrainRV_Callback(type)

```
Creaty summary of strain result (tagging or cine strain analysis)
from multiple matfiles in one folder.
This function is very useful for research. The user
performs all strain analysis and then exports all data to
one spreadsheet.
```

exportmultiplestrainRVgraphs_Callback(type)

```
Creaty summary of strain result (tagging or cine strain analysis)
from multiple matfiles in one folder.
This function is very useful for research. The user
performs all strain analysis and then exports all data to
one spreadsheet.
```

exportmultiplestrain_Callback(type)

```
Creaty summary of strain result (tagging or cine strain analysis)
from multiple matfiles in one folder.
This function is very useful for research. The user
performs all strain analysis and then exports all data to
one spreadsheet.
```

exportmultiplestraingraphs_Callback(type)

```
Creaty summary of strain result (tagging or cine strain analysis)
from multiple matfiles in one folder.
This function is very useful for research. The user
performs all strain analysis and then exports all data to
one spreadsheet.
```

exportrv2stl(no,tf,fignr,filename,resolution)

```
Exports RV surface to STL (with inner and outer contour, closed in apex
and surfaces merged in the base. OBS this function ignores RV epicardium
and fakes in an epicardium. Wall thickness is set to 1mm.
```

exportrv2stl_Callback
```
Exports RV to STL file. This file ignores epicardium and fakes in a
epicardium.
```

ok = exportsavemovie(mov,left,right,up,down,fps)
```
Exports a move as an avi file or a set of png-files.
- mov is a movie struct.
- left,right,up,down are crop coordinates.
- fps is frame rate.

Allows user to select different codecs.
```

outdata = exportslicehelper(outdata,rowoffset,coloffset,type,x,y)
```
Helper function to export slice based data.
```

exportslicevolume_Callback
```
Export slicebased volume.
```

exportthisstack(doheader)
```
Export data from current image stack to clipboard.
```

exporttoclipboard_Callback(doheader,no)
```
Export data to clipboard. Calls another function to do the export.
```

exporttoclipboardthisstack_Callback(doheader)
```
Exports data for current image stack to clipboard.
```

exportvolumecurve_Callback
```
Export volume curve to clipboard.
```

[outdata,ind] = header(onlyone)
```
Helper function to write header when exporting data.
```

[xnew,ynew] = innersurface(x,y,no,dist)
```
Takes a surface x,y and returns a new surface inside of this surface with
a distance of dist mm. x are assumed to be size NumDataPoints*Z.
```

[x,y] = meshfixer(x,y)
```
Fix a mesh if the "layers" are twisted.
The mesh is assumed to be rows of coordinates.
```

mmodemeasurements_Callback
`Export mmode measurements.`

stri = removenumerics(stri)
`Remove numbers from a filename.`

screenshot_Callback
`Create an image file of a screenshot of the main axis.`

15.6 Segmentation superunit

The Segmentation superunit contains functions that are used for doing segmentations on images. These are used to perform quantifications and usually leave visible traces that require graphical updates performed by the Draw superunit.

15.6.1 Measurement unit

The purpose of the Measurement unit is to make linear measurements of details in images. Some of these measurements can be used for other quantifications.

Interactions

This unit calls the Helper functions superunit to handle mouse input from the user and the Draw superunit to make graphical updates.

Datastructure

The `SET` struct has a field `Measure`, which is a struct that contains measurements in the image stack. If no measures exist, then `Measure` is an empty array. If measures exist then it is a structure array where each measurement has the following fields:

- `X` A 2 x 1 vector with X-coordinates.
- `Y` A 2 x 1 vector with Y-coordinates.
- `Z` scalar with the slice.
- `Length` Length of the measure in mm.
- `Name` String containing the name of the measure.

Functions

`measure_Buttondown(panel)` Button down function for measurements.

`measure_Motion(ind)` Motion function for measurements.

`measureexport_Callback` Export measurements.

`measuremove_Callback(dx,dy)` Helper function to move measurements.

`measurepoint_Buttondown(panel)` Buttondown function fore a measurement point/marker.

`measureput_Buttondown` Called when a measurement point (except the endpoint) is placed

`measure_Buttonup` Button up function for measurements.

`[stri,lstr] = measureasklabel(g)` Asks for a label of a measurement.

`measureclearall_Callback(g)` Clear all measurements.

`measureclearthis_Callback(g)` Clear current measurement.

`measurefontsize(g,panel,index)` Sets measure font size.

`measurerenamethis_Callback(g)` Rename current measurement.

15.6.2 Annotation point unit

The purpose of the Annotation point unit is to allow the user to place annotation points in an image.

Interactions

This unit calls the Helper functions superunit to handle mouse input from the user and the Draw superunit to make graphical updates.

Datastructure

The `SET` struct has a field `Point`, which is a Struct that contains position of annotation points. The struct has the following fields:

- `X` 1 x N vector with X-coordinates, where N is the number of annotation points.
- `Y` 1 x N vector with Y-coordinates.
- `T` 1 x N vector with time frames (i.e. one for each point).
- `Z` 1 x N vector with slice (i.e. one for each point).
- `Label` 1 x N cell array with string with the labels.

Originally all points were non time resolved. Thereafter where time resolved points implemented by making T points from one point. Non time-resolved points are shown as bold text. Time-resolved points are only implicitly connected by their position and more importantly the label.

Functions

point_Buttonup
```
Button up function for points.
```

point_Motion
```
Motion function for points.
```

pointat_Buttondown(panel,type)
```
Buttondown function when clicked at a point.
```

pointclearall
```
Clear all points.
```

pointclearall_Callback
```
Clear all points.
```

pointcleartemplate_Callback
```
Clear points using naming template.
```

pointdeletethis_Callback
```
Delete point.
```

pointexportall_Callback
```
Export all point data.
```

ind = pointfind(silent)
```
Find neareast point.
```

pointforward_Callback
```
Track point forward in time.
```

pointmaketimeresolvedthis_Callback
```
Converts a none time resolved point to a time resolved point.
```

pointmove_Callback(dx,dy)
`Helper function to move points.`

pointrenametemplate_Callback
`Rename points according to a renaming template.`

pointrenamethis_Callback
`Rename point.`

15.6.3 LV segmentation unit

The purpose of the LV segmentation unit is to provide tools for automated and semi-automated segmentation of the left ventricle from either Cardiac MR, Cardiac CT or Myocardial Perfusion SPECT (MPS). The used segmenation algorithm for CMR is described in [8]. The used segmentation algorithm for MPS is decribed in [9] and [10].

Interactions

This unit calls the Helper functions superunit to handle mouse input from the user and the Draw superunit to make graphical updates. It is related to the Contours unit described in Section 15.6.5 that are used to copy and import segmentations between image stacks. The RV segmentation unit described in Section 15.6.4 shares code with the LV segmentation unit.

Datastructure

The LV segmentation is stored in the `SET` variable and specifically the fields `EndoX` and `EndoY`. The units are pixels.

Functions

The functions for lv segmentation in CMR images are described in the file `lv.m`. The functions for lv segmentation in CT images are described in the file `CTLVSegmentation.m`. The functions for lv segmentation in MPS images are described in the file `spectlvsegmentation.m`. These files are not autodocumented in this Technical manual, since it contains propriety algorithms. Please consult the `lv.m` and `spectlvsegmentation.m` files for details instead.

15.6.4 RV segmentation unit

The purpose of the RV segmentation unit is to provide tools for semi-automated segmentation of the right ventricle. The used segmenation algorithm is a quick modification of the LV segmentation algorithm(s). The code is implemented in the file `rv.m`, but does also share code with the LV segmentation unit in Section 15.6.3.

Interactions

This unit calls the Helper functions superunit to handle mouse input from the user and the Draw superunit to make graphical updates. It is related to the Contours unit described in Section 15.6.5 that are used to copy and import segmentations between image stacks. The LV segmentation unit described in Section 15.6.3 shares code with the RV segmentation unit.

Datastructure

The RV segmentation is stored in the `SET` variable and specifically the fields `RVEndoX` and `RVEndoY`. The units are pixels.

Functions

The functions are described in the file `rv.m`. This file is not autodocumented in this Technical manual, since it contains propriety algorithms. Please consult the `rv.m` files for details instead.

15.6.5 Contours unit

The purpose of the Contours unit is to do operations on segmentation contours. The unit is implemented in the file `segmentation.m`.

Interactions

Since many of its operations change the segmentation, this unit frequently calls Draw unit operations to update the graphic display.

Datastructure

The segmentation information on which to operate is taken from the `SET` struct and the stack number is taken from `NO` variable.

Functions

clear_helper
Helper fcn to clear segmentation.

clearall_Callback
Clear all segmentation, both endo and epi, lv and rv.

clearalllv_Callback(silent)
Clear all lv segmentation, both endo and epi.

clearallrv_Callback
Clear all rv segmentation, both endo and epi.

clearslices(no,ind,timeframes,endo,epi,rvendo,rvepi)
Workhorse in clearing slices.

clearslices_Callback(endo,epi,rvendo,rvepi)
Helper function to clear segmentation in selected slices.

clearslicesthis_Callback(endo,epi,rvendo,rvepi)
Clear segmentation for selected slices according to mode.

clearthis_Callback(endo,epi,rvendo,rvepi)
Helper function to clear segmentation in this (current slice) slice.

[sourceslice,destslice,sourcetime,desttime,zdirsource,zdirdest] = findmatchingslices ...
(tono,fromno,doendo,doepi,dorvendo,dorvepi,takefromclosestseg,importtf)
Find matching slices between source image stack and destionation image stack
The matching is based on camera position.

importadjust(no)
Snap segmentation to another frame.

importadjust_Callback
Snaps imported segmentation to image by trying to look at edge detection.

importfromcine2scar_Callback
Import segmentation from cine to scar image stack.

importsegmentation_Callback(no,importtf)
```
Import segmentation from another image stack.
Imports to current image stack NO from no or if called with no input
arguments user is asked.
```

[desttimeframes,destslices,sourceslices] = importsegmentationhelper ...
(tono,fromno,doendo,doepi,dorvendo,dorvepi,importtf)
```
Helper function to segmentimportsegmention Callback.
- tono is destination of segmentation.
- fromno is source.

The function is capable of handeling slice offsets and different
pixelssizes as well as situations when number of timeframes differ. When
destination is not timeresolved and source is timeresolved then user is
asked from what timeframe to take the segmentation from.

Work horse in importing. This function would benefit from anti-cut
and paste treatment.
```

importsegmentationwithsnap_Callback(no,importtf)
```
Same as importsegmentation but also snaps contour (rigid registration).
```

interpolatedelineationovertime_Callback
```
Interpolate LV or RV delineation over time from existing delineations.
```

removeallinterp_Callback(silent,no,arg,indarg)
```
Remove all interp points.
```

removeallpins_Callback(silent,endo,epi,rvendo,rvepi)
```
Remove all pins.
```

removeallpinsthisslice_Callback(current,endo,epi,rvendo,rvepi)
```
Remove all pins in this slice.
```

removepin_Callback(type,m)
```
Removes pins.
```

removethispins_Callback(endo,epi,rvendo,rvepi)
```
Remove pins in this slice and timeframe.
```

resetedgedetection
```
Reset the edge detection.
```

rvcopyfromlvendo_Callback
`Copy RV from LV segmentation (endocardium).`

rvcopyfromlvepi_Callback
`Copy RV from LV segmentation (epicardium).`

z = score(im,x,y)

15.6.6 Viability unit

The purpose of the viability module is to quantify scar from delayed enhancement images. The viability tools have been extensible described and validated in [11], and [12]. The transmurality is documented in [13]. The term noreflow that is used in the code is a legacy term since this was used in the scientfic literature at the time of implementation of the module instead of the now accepted term microvascular obstruction or MO.

There is no dedicated user interface, all interactions are performed in the main window and some user changable defaults are displayed in the menu, and stored in the `SET` struct.

Interactions

The viability unit have interactions with the following units:

- Draw superunit. The unit calls routines in the viability unit to perform drawing of colour overlays indicating regions of manual corrections. Calls to `reshape2layout`.
- Helper functions superunit. Find slices with endocardial and epicardial segmentation in which the scar segmentation can be performed. Handle mouse interaction when drawing scar manually.
- Calculation superunit. Volume calculations of scar, finding slices, creating masks, Volume unit

Datastructure

The data is stored in the `Scar` field of the `SET` struct.

- `IM` contains the image itself (a copy). This is from historical issues and may be removed in future versions.
- `Auto` a logical array that contains true in the positions that the algorithm marks as 'infarcted'.
- `Manual` an `int8` array that contains 1 for the pixels that the user have manually to be infarcted and -1 for the pixels that the user have marked as non-infarcted.
- `Result` a logical array containing the final viability delineation with the manual corrections.
- `NoReflow` a logical array containing one in the regions that have been identified as no reflow regions, or regions with microvascular obstruction.
- `MyocardMask` a logical array containing true in the myocardium and false otherwise.
- `beta` a scalar controlling the ruffness of the surface. See the algorithm description for further details [11].
- `stdlimit` a scalar determine the number of standard deviations from remote to take.
- `radius` Radius of the fast level set algorithm, see [11] for details.
- `Percentage` a scalar representing the percentage of the pixels that are marked as infarcted.
- `OnlyEndo` true if only endocardium delineation exists.
- `UseWeighting` true if weighted mode is used.
- `MOPercentage` percentage microvascular obstruction.
- `UpdateDirectly` true if direct update on volumes should be performed after adjustments
- `minweight` this is the minimal weight used in the weighting algorithm
- `MOThreshold` threshold used for microvascular obstruction. The MO threshold is defined as threshold for myocardium times `MOThreshold`.

Functions

calchelper(force,slicetodo); %This function sets UpdateDirectly to true after one go.

```
;d;.
```

[patchinessindex,fullindex] = calcpatchinessindex(no)

```
Calculates a patchiness index. Roughtly it is the perimeter of the scar
divided with twice the length of the scar.
```

[res,varargout] = calctransmuralityline(numsectors,no)

```
Calculates transmurality.

Output in order
- Mean transmurality in the number of sectors
- Max transmurality in each sector
- Mean transmurality calculated only over infarcted areas.
- Total Extent
- "Start" transmurality
- "End" transmurality

Note that you need to set StartSlice and EndSlice!!!.
```

[res,varargout] = calctransmuralityweighted(numsectors,no)

```
Calculate transmurality according to the weighted method.
```

[core,greyzone,scarregion] = calcweightedgreyzone(no)

```
Calculates greyzone according to weighted method.
```

checklinkage

```
check if current image stack is linked, if so aske user to unlik the image
to perform scar analysis.
```

grayzoneclear_Callback

```
Removes Gray Zone analysis from current image stack.
```

moslider_Callback

```
Callback for MO slider.
```

moslideredit_Callback

```
MO slider Callback.
```

mosliderhelper(status,no)

v = nanremove(v)
Replace NaN with zeros.

final = remove_holes(a)
The function removes hole in an ND image.

sdfromremoteedit_Callback
This callback is executed when user changes editbox coupled to sdfromremote.

sdfromremoteslider_Callback
This Callback is called when user adjust SD from remote slider. Slider1 is coupled for this purpose in SD from remote mode.

sdsliderhelper(status,no)
Helper function to toggle sd slider visibility.

sliderupdate(no)
This function is called to set up proper slider behaviour in viability mode.

viabilityautocalc(force)
Automatically calculate viability. This is the workhorse.
Different options for calculation se as Scar.Mode.
If force is not true then ask if stdlimit ~= 1.8.

viabilityautoem
Set EM mode.

viabilityautoewa
Automatic viability using Janes method.

viabilityautofwhm
Set FWHM mode.

viabilityautootsu
Automatic viability using the method in the Outsu method.

viabilityautosdfromremote
```
Automatic viability using the method SD from remote.
```

viabilityautoweighted
```
Automatic viability using the method in the
Radiology paper.
```

viabilitycalc(force)
```
Automatically calculate viability. This is the workhorse.
Different options for calculation se as Scar.Mode.
If force is not true then ask if stdlimit ~= 1.8.
```

viabilitycalchelper(force,slicetodo)
```
Automatically delineates viability. This is the workhorse. For weighted
mode this is run every time changes are made, for other modes it is
typically only run once.
```

viabilitycalchelperfwhm(no)
```
Help function for FWHM.
```

viabilitycalchelpersdfromremote(no)
```
Helper function for SD from remote implementation.
```

viabilitycalchelperweighted(force,slicetodo)
```
Old weighted infarct calculation.
```

[slicetodo,ok] = viabilitycalcinit
```
Preprocessor for viabilitycalc.
```

viabilitycalcvolume(no,intweight)
```
Calculates the volume of the viability.
```

viabilityclear_Callback
```
Callback to clear scar data.
```

viabilityclearmanual_Callback
```
Callback to clear manual interactions.
```

viabilityclearmanualslice_Callback
```
Callback to clear manual interaction in current slice.
```

viabilityclearslice_Callback
Callback to clear viability for current slice.

viabilitycreatemask(no)
Helper function to create the myocardial mask.

viabilitygetsd_Callback
calculate nbr of sd based on auto/manual scar segmentation.

viabilitymanual
Set manual mode.

viabilitymenu(no)
Update the viability menu (helper function).

intweight = viabilitynewweight(no)
Experimental new weighting.

viabilityreset_Callback(mode,no)
Callback to reset all scar settings.

viabilitysetbeta_Callback
Set the curvature beta, used in the levelset part of the viability algorithm(s).

viabilitysetminvolume_Callback
Sets minimal volume for a single infarct.

viabilitysetsdfromremote_Callback
Set number of standard deviations from remote.

viabilityshowgrayzone_Callback
Show overlays of core (dark red) and greyzone (dark yellow).

viabilityshowinfarctaswhite_Callback(show)
Show infart as white overlay.

viabilityupdatewithmanual
Upates the scar with manual interactions and removes small infarcts (depending on mode). Also updates the volumes. This function was previously part of viabilitycalc and are now called from that function.

[intweight,varargout] = viabilityweight(no)
```
Finds intensity mapping of infarct for weighted algorithm.
This function is used both for old weighted algorithm and EWA.
```

weightedslider_Callback
```
Weighting slider, only HMG for now.
```

weightedsliderautopushbutton_Callback
```
Automatic adjustment of weighted slider for a fix infarct size.
```

weightedslideredit_Callback
```
Weighting slider edit box, only HMG for now.
```

weightingsliderhelper(status,no)
```
Helper function to toggle weightd slider on/off.
```

15.6.7 Myocardium at risk unit

The purpose of the Myocardium at risk unit is to be able to do manual or automatic segmentation of the ischemic myocardium at risk of infarct from CMR or Myocardial Perfusion SPECT (MPS).

Interactions

This unit interacts with the followint units:

- Draw superunit to draw the contours and manual interactions for myocardium at risk.
- Calculation superunit to update volume calculations
- Helper functions unit to find slices with endocardial and epicardial segmentation in which the segmentation of myocardium at risk can be performed

Datastructure

The data is stored in the `SET` variable in the field `.MaR` with the following subfields.

- `Auto`: Stores the automatic segmentation
- `Result`: Stores the resulting segmentation from `Auto` and `Manual`

- `Manual`: Stores manual segmentation or manual interactions
- `MyocardMask`: Stores a mask of the myocardium i.e. a mask of the pixels between endocarium and epicardium
- `MR`: Stores parameters specific to segmentation in CMR
- `MPS`:Stores parameters specific to segmentation in SPECT
- `Percentage`: Stores the resulting volume of myocardium at risk as percentage of left ventricular mass
- `Mode`: set to manual if manual drawing of myicardium at risk
- `UpdateDirectly`: Parameter to decide if graphical update should be performed directly

Functions

All functions for myocardium at risk delineation is implemented in mar.m and t2wmarsegmentation and spectmarsegmentation for the automatic segmentation in t2w CMR, respectively SPECT.

15.6.8 ROI analysis unit

The purpose of the ROI analysis unit is to provide general tools for region of interest analysis and to be able to extract signal intensities and related statistics.

Interactions

- Draw superunit since it is responsible to graphically draw the regions of interest on screen.
- Report superunit uses the export to clipboard features to export data.
- The report unit uses the `calctruedata` function in the Calculation superunit.

Datastructure

The data is stored in the `SET` variable in the field `Roi` and contains the following fields:

- `X`, X-coordinates of regions of interests. They are stored in an array as N x T, where N is the number of points along the contour (to be precise it is actually `DATA.NumPoints`), T is the number of time frames. For non time resolved ROI's `X` can also be a vector of length N.
- `Y`, Y-coordinates of regions of interest. For details about size, see above.
- `T`, T-coordinates of the region of interest.
- `Z`, Z-coordinate of the region of interest.
- `Sign`, sign of the region of interest. This is used for flow quantifications and is stored as a vector with elements that are either -1 or 1.
- `Name`, name of the region of interest.
- `LineSpec`, line specification for the region of interests. Stored as a string as Matlab compatible line specifications. Examples are `'b-'` (blue line), `'y:'` (yellow dotted line).
- `Area`, area of the ROI for each timeframe.
- `Mean`, mean intensity of the ROI content for each timeframe.
- `StD`, standard deviation of the ROI content for each timeframe.

Related are also the fields `RoiArea`, `RoiCurrent`, and `RoiN`.

Functions

expandcontract_Callback(stepsize)

removefromallbutthistimeframe(n)
`Remove ROI from all but this time frame.`

removefromthistimeframe(n)
`Remove ROI from this timeframe.`

roi_Buttondown(panel)
`button down function for drawing roi.`

roiaddfixsize_Callback
`Function for adding roi with fix size.`

roiaddinsector_Callback(angle, width,percentfromendo, ...
percentfromepi, numsectors, name)

```
Add sectors in ROI, input arguments is angle of where to start, the
width of the sector given as an angle, how many percent from endo the
sectors will be placed, how many percent from epi the sectors will be
placed, the number of sectors and finally the name under which the
sector/roi will be stored.
```

name = roiasklabel(varargin)

```
Lets user pick name of roi from a picklist.
Input: roitoname is the number of the roi to name, if this input argument
is empty the choice of naming the roi to 'ROI-n' is disabled (the function
can not handle a cell so if multiple rois are to be named use '' as first
input), roinamein is the name of the roi to be renamed (this functionality
is used when renaming using template). Both input arguments are optional
but if the first argument is not supplied the choice of naming the roi to
'ROI-n' is disabled.
```

name = roiasktemplate(no)

```
function used for asking for name of roi template.
```

roiclearall_Callback

```
clear all ROIs.
```

roicopyalltimeframes_Callback(n)

```
copy ROI to all time frames.
```

roicopydownwards_Callback(n)

```
copy ROI one slice downward.
```

roicopyendo_Callback

```
Copy from endocardium to a ROI.
```

roicopyepi_Callback

```
Copy from epicardium to a ROI.
```

roicopyfromotherimagestack_Callback

```
Copy roi from other image stack. Let user select from
which image in an input dialog.
```

roicopymar_Callback

```
Copy from scar to a ROI.
```

roicopyscar_Callback
Copy from scar to a ROI.

roicopyupwards_Callback(n)
copy ROI one slice upward.

roidelete_Callback(n,draw)
delete ROI n.

roiexportvalues_Callback
Get what ROI's to take.

no = roifindmag(no)
Find magnitude no.

roiforce_Callback
Function to set force shape over time in menu checked and unchecked.

roiforceapply
function to force shape over time.

m = roiget(stri,arg)
Gets roi either by getting closest ROI to clicked coordinates
if two input arguments or by selecting in a pick list.

roisign = roiguessign(nom,nop,currentroi)
Guesses sign of the roi to make the net flow through the roi be positive.

roihistogram_Callback
Make intensity histogram from ROI with either normalized intensities or true intensities. Uses roiselector to let user decide what data to be analysed in the histogram. Also let's user decide how many bins in the histogram and if zero should be excluded.

values = roihistogram_helper(rois,timeframes,normalized,excludezero,export)
Helper function to roihistogram, extracts values.

roiimportroi_Callback(no)
Import roi from image stack. Let's user select from which image in an input dialog.

roilabel_Callback(x,y)
Function for putting name on a roi.

roiputroi_Buttondown(panel)
Buttondown function for putting circular roi.

roi = roireset(no)
Reset ROI values when stack is emptied.

[rois,timeframes,normalized] = roiselector(usealltimes,thissliceonly,template,normalized)
Dialogbox in which user decides if all timeframes, only this slice
and normalized intensity values should be used. All values can be
changed seperately.

roisetcolor_Callback(n)
Change color of roi. Which roi to use is either decided by closest roi from
clicked coordinate if input argument is -1 else by user selecting from a
pick list.

roisetlabel_Callback(n)
Change name of roi. Which roi to use is either decided by closest roi from
clicked coordinate if input argument is -1 else by user selecting from a
pick list.

roiswitchsign_Callback(m)
Switch sign of roi. The sign is used when calcualting flow through roi.

roitemplatedelete_Callback
Function to delete roi by template.

roitemplatesetcolor_Callback
Function to rename roi by template.

roitemplatesetlabel_Callback
Function to rename roi by template.

roithresholdnumeric_Callback
numeric threshhold inspector.

roithresholdvisual_Callback
visual threshhold inspector.

roitoendo_Callback
`Copy from endocardium to a ROI.`

roivisualslider_Callback(arg)
`gui for visual threshhold.`

roivisualsliderkeypressed
`key pressed function for visual slider.`

selectroi_Buttondown(panel,roinbr)
`Buttondown function used for changing current ROI.`

15.7 Resources superunit

This superunit contains functions called by the user and whose interactions with other functions are few.

15.7.1 Help unit

The purpose of the Help unit is to contain functions that are called from the Help menu.

Interactions

Interactions are negligible from this unit.

Datastructure

Use of datastructures is negligible.

Functions

about_Callback
`Displays help information about the software.`

aboutrvq_Callback
`Displays help information about RVQ.`

bug_Callback
`User bug report function.`

generalhelp_Callback
Open Medviso homepage in browser.

hotkeys_Callback
Shows help of hot keys in Segment.

instructionsforuse_Callback
Open instructions for use file.

openlogfiles_Callback
Open list of log files in browser.

openthislogfile_Callback
Open log file for this session in browser.

support_Callback
Open mail composer to submitt email to support.

supportreq_Callback(call)
Callback to open support request GUI.

tutorials_Callback
Opens the webpage for showing tutorials on the software.

usermanual_Callback
Open reference manual.

15.7.2 Preferences unit

The purpose of the preferences unit is to provide a mechanism of storing user preferences in Segment.
For a complete description of the preferences and their usage, see the Reference Manual. The preferences are read when Segment is started or when the preferences GUI are opened. Segment tries to find a suitable place for the preferences file by looking in the environment variables `APPDATA`, `USERPROFILE`, `HOMEPATH`. Place can be found by calling the Segment function `getpreferencespath`.

Interactions

The preferences data is read from multiple places in Segment but is only assigned in `segpref.m` with the exception of loading that sets latest loaded file.

Datastructure

The preferences are stored in the `DATA` object with the fields:

- `datapath`. Path to image data. This is the path first opened when the fileloader GUI is started.
- `exportpath`. Path where image data is exported to.
- `AnonymMode`. True if patient details should no be shown on screen. Note that this does only affect the screen. To permanently anonymize an image stack, see Reference Manual for details.
- `AddPoints`. True if pins should be added when manually drawing a part of a contour.
- `EndoCenter`. True if the center of the endocardium is used for drawing spikes and regional wall motion analysis. If false the the center of the epicardial surface is used.
- `BlackWhite`. True if the lines contours should be drawn in white color instead of object specific colors.
- `LineWidth`. Width of the lines. Default is 1.
- `NumPoints`. Number of points to evaluate the endocontour along.
- `LearnMode`. True if learning messages should be displayed.
- `UndoHistory`. The maximum length of the undo history.
- `reportsheetpath`. Path to where the files for the report sheets are generated.
- `IncludeAllPixelsInRoi`. If true, then also pixels that are touched by the ROI are included in the subsequent processing. Default is false, and in this mode only pixels whose centrum are inside the contour are included.
- `AutoSave`. If true then the segmentation is autosaved every fifth minute under the name `autosave.seg`.
- `ContourAdjustDistance`. The maximum distance to a contour a user can click before the contour is not acknowledged as a click on that contour. Measured in pixels.

- `PacsTempStoragePath`. The path where the temporary files for the PACS retreival are stored. This field might be obsoleted in future versions.
- `ExcludePapilars`. True if the papilars should be excluded in the automatic segmentation. See Reference Manual and [2, 3] for details.
- `UseLight`. True if to use current brightness and contrast as a cue in the Segmentation process. For further details see the Reference Manual.

Functions

- `segpref.m` contains all callbacks and GUI code for the main and advanced preferences GUI's.
- `pacspref.m` contains all callbacks and GUI code for the PACS preferences GUI.
- `loadpreferences` loads the preferences file and stores the information in the data structure. This function contains an important subfunction called `preferencesbackward` that handles backward compability for older Segment versions.

15.8 Tools superunit

The purpose of the image tool unit is to provide tools to perform on image stacks.

Interactions

This unit uses the Calculation superunit to make calculations for some advanced tools, and the Helper functions superunits to check for existence of image content to be copied to other stacks.

Datastructure

No specific datastructure is used for the tools.

Functions

('There is no built-in undo function for this. Are you sure?');

addnoise_Callback(f)
Adds noise to current image stack.

anonymous_Callback(silent,newname)
Makes a data set anonymous by removing
- PatientInfo.Name
- PatientInfo.ID
- PatientInfo.BirthDate
- FileName
- OrigFileName
- PathName.

anonymoustotal_Callback(silent,newname)
Makes a data set completely anonymous by removing
- PatientInfo.Name
- PatientInfo.ID
- PatientInfo.BirthDate
- PatientInfo.Sex
- PatientInfo.Age
- PatientInfo.AcquisitionDate
- PatientInfo.Length
- PatientInfo.Weight
- PatientInfo.BSA
- PatientInfo.Institution
- FileName
- OrigFileName
- PathName.

applylight_Callback
Apply current light setting to current image stack. Makes permanent changes in the IM field.

autoesed_Callback(silent,no)
Autodetect and store ED, and ES.

closesetimagedescription_Callback
Close the gui for setting the image description.

copydownward_Callback(type,silent,dolv,lvalg)
Copy segmentation in current slice downwards and refine.

copyforwardselected_Callback(lv)
Copy segmentation of selected slices forward in time.

copytoalltimeframes_Callback(type,silent,dolv)
Copies segmentation in the selected timeframe and slice to all timeframes.
type is either endo or epi.
silent is true if to avoid graphic update.
if dolv false then copy rv.

copyupward_Callback(type,silent,dolv,lvalg)
Copies upward and refine.
type is either endo or epi.
silent is true if to avoid graphic update.
if dolv false then copy rv.

crop_Buttondown(panel)
Buttondown function to crop the image stack.

crop_Buttonup
Buttonup function for cropping of image stacks.

crop_Motion
Motion function to crop the image stack.

[outx,outy] = cropcontour(inx,iny,isroi)
Used to crop and reinterpolate contours.

nos = crophelper(no0,xind,yind)
This function is the enginge in cropping.

cropspecial
undocumented function.

cropupdate(nos)
Update image after a crop.

disableundo(no)
Disable undo function. Also copies segmentation data
to make sure data is consistent.

duplicateimagestack_Callback
Duplicates current image stack. This function breaks links to
linked datasets, parent and children.

enableundo(no)
```
Enables undo function in menus and icons. Also
copies old segmentation data so undo i possible
This should allways be called before routines that
change the segmentation.
```

enddiastole_Callback(noupdate)
```
change current time frame to end diastole.
```

enddiastoleall_Callback
```
Change current timeframe of all image stacks to end diastole.
```

endsystole_Callback(noupdate)
```
Change current time frame to end systole.
```

endsystoleall_Callback
```
Change current timeframe of all image stacks to end systole.
```

extraslice_Callback(type)
```
Add an extra slice. Type is either 'basal' or 'apical'
Takes also care of delineations.
```

flip_Callback(dim)
```
Helper function to image stack flipping tools.
```

flipt_Callback
```
Helper function to flip in t direction. Takes care of segmentation.
```

[a,b] = flipvars(b,a)
```
Flip variables. Simple and elegant.
```

flipx_Callback
```
Flip x direction of current image stack.
Need to flip both x and z to maintain a righthand system.
```

flipx_helper
```
Helper function to flip in x direction. Takes care of segmentation.
```

flipxy_helper
```
Interchange z  and  t of current image stack. Takes care of segmentation.
```

flipxz_Callback
Interchange z and x of current image stack. Takes care of segmentation.

flipy_Callback
Flip y directon of current image stack.
Need to flip both y and z to maintain a righthand system.

flipy_helper
Helper function to flip in y direction. Takes care of segmentation.

flipz_Callback
Flip z direction of current image stack.
Need to flip both z and x to maintain a righthand system.

flipz_helper
Helper function to flip in z direction. Takes care of segmentation.

flipzt_Callback
Interchange z and t for current image stack. Takes care of segmentation.

imageenhancement_Callback
Image enhancement using adapthisteq (CLAE), please see adapthisteq for details.

imageinfo_Callback(arg)
Displays image information of current image stack.

intensitymapping_Callback
Plots intensity mapping function.

invertcolors_Callback
Invert colors in current image stack (essentially 1-x).
Need to fix more to get intensity offset correct.

kspace_Callback
Shows KSPACE of image (Fourier Transform). Displays the log of the Fouries Transform.

mirrorx_Callback
Mirros in x direction.
Need to flip both x and z to maintain a righthand system.

normalize_Callback
Normalize image data of current image stack. Make sure image intensities are in the range [0..1]. Also stores in so that true image itensities can be retrieved, see calctruedata.

opensetimagedescription_Callback
Open the gui for setting the image type, image view plane and imaging technique for current image stack.

precomp_Callback
Function to make intensity precompensation of MR gradient echo images. This function is somewhat obsoleted.

removeallbutedes_Callback
Remove all timeframes in current image stack except diastole and systole.

removeallbutthistimeframe_Callback
Remove all timeframes except current timeframe.

removecurrenttimeframe_Callback
Remove current timeframe.

removeduplicatetimeframes_Callback(duplicates)
Removes duplicate timeframes, askes for the number of duplicates. Two means keep every second frame.

removenexttimeframes_Callback
Remove next timeframe and upto and including last timeframe.

removepreviousstimeframes_Callback
Removes previous timeframe and all frames to first timeframe.

removeselected_Callback
Remove selected image stacks from current image stack.

removeslices_Callback(ind,force)
Remove slices from current image stack. This is the workhorse when removing slices.

removethis_Callback
Remove current slice.

removetimeframes(ind)
```
Helper function to remove timeframes. This is the workhorse
when removing timeframes.
```

removeunselected_Callback
```
Remove unselected slices from current image stack.
```

x = resamplehelper(f,x)
```
Helper function to resample image stacks.
factor 2  gives  x' = 2*x-0.5
factor 3  gives  x' = 3*x-1
factor 4  gives  x' = 4*x-1.5
factor 5  gives  x' = 5*x-2
factor 2.5  gives  x' = 2.5*x-1
factor 3.5  gives  x' = 3.5*x-1.5
factor 0.5  gives  x' = x'*0.5 (a odd)
factor 0.5  gives  x' = x'*0.5+0.5 (a even).
```

rotate90right_Callback
```
Rotate current image stack 90 degrees right. Currently not working
properly.
```

setcolormap_Callback(type)
```
Set colormap for current image stack.
```

setcurrenttimeframeasfirst_Callback(applyto,showmessage)
```
Sets current timeframe as first time frame by cycling data in time.
Works with existing contours, but not general segmentation module, or
time resolved annotation points.
```

seted_Callback
```
Set diastole to be current timeframe.
```

setes_Callback
```
Set systole to be current timeframe.
```

setheartrate_Callback
```
Set heart rate of current image stack.
```

setimagedescription_Callback
```
Set the image type, image view plane and
imaging technique for current image stack.
```

setimageinfo_Callback(arg)
Asks for and sets image details of current image stack.

slidingaverage_Callback
Performs sliding average of an image.

smoothsegmentation_Callback(no)
Smooth latest segmentation (LV, RV and ROI).

temporalmeanvalue_Callback(silent)
Calculate temporal mean image by averaging over time.

translatecontour_helper(dx,dy,translateimage)
Helper function to translate contours and image.
If translateimage is true then image is also translated.

Not yet supported:
- Scar
- MaR
- Pins
- Levelset.

translatecontours(dx,dy)
Translate all contours. Called by hotkeys.

x = translatecontours_helper(x)
Fix out of bounds indices.

translatecontoursandimage(dx,dy)
Translate all contours and the image. Called by hotkeys.

undosegmentation_Callback(no)
Revert segmentation from undo history.

unlinkimages_Callback
Unlink images.

upsampleimage_Callback(f, inputNO)
Upsamples current image stack (in slice only). Takes care of segmentation in the upsampling process.

newvol = upsampleslices(f,vol,silent)
```
Upsamples along last dimension.
Limitation input must be single or double.
```

upsampleslices_Callback(f)
```
Upsample current image stack in slice direction. Takes care of
segmentation.
```

newvol = upsampletemporal(f, vol, varargin)
```
Helper function to upsample in time. f is factor and vol is volume to
upsample. Third argument is optional and is type;
 (image, segmentation (default), and vector.
```

upsampletemporal_Callback(f)
```
Usamples current image stack in time. f is upsampling factor.
Takes care of segmentation.
```

newvol = upsamplevolume(f,vol,silent)
```
Helper function to upsample a volume vol.
```

viewpatientinfo_Callback(arg,no)
```
GUI to view and adjust patient information.
```

viewtrueintensity_Callback
```
View true image intensities in current timeframe and slice.
```

15.9 File superunit

The File superunit contains functions for loading, saving, transferring and doing operations on files. An overview of its subunits and how they communicate is provided in Figure 5.

15.9.1 Open File unit

This is the GUI part of the file loader.

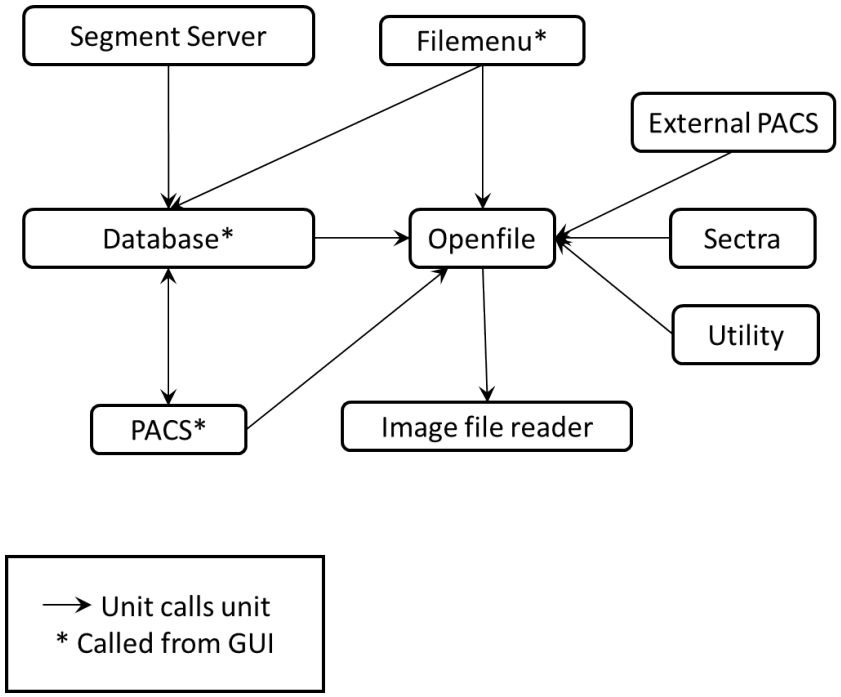

Figure 5: Overview of the internal structure of the `File` function superunit and transaction analysis.

Interactions

This function interacts with:

- Main Segment superunit `segment_main` with the `renderstacksfromdicom`, setting title, clearing SET variable and other preparations for loading data, enabling and disabling GUI options.
- Tools superunit with normalization.
- Image file reader unit to do the low level data reading.

Datastructure

The function heavily use the DATA.Preview field for communication with the `segment_main` in the loading process.

Functions

This subsection describes the subfunctions of `openfile.m`

bothmagnitudeandphase(no)

```
Called when image stack contains both magnitude and phase.
```

browsebutton_Callback

```
Callback when user wants to browse for new folder.
```

cancelpushbutton_Callback

```
Dismiss the figure.
```

continueloading

```
This function is called from cropbox buttondown.
It continues to load the whole images stack and finally call Segment
to tell that finished loading one image stack.
```

[im,tempmax] = contrastfixer(im)

```
Takes a 2D image that should be presented and crops the histogram
somewhat. This is particularly important if there are some pixels that are
very bright (overlayed images with timestamp from old Siemens machines
are one such example.

tempmax is the maximum value that the image is truncated to.
```

[im] = contrastfixerrescale(im)
Same as contrastfixer, but scales between 0-1
tempmax is the maximum value that the image is truncated to.

cropbox
Called from continue loading and sets up GUI to ask user to select region of interest to load.

cropbox_buttondown
Buttondown function for crop. This is equal to select.

cropbox_motion
Motion function for cropbox.

dicomsorter_Callback
Simply calls dicomsorter, but with current folder.

enablesegmentgui(no)
Helper function that enables vital GUI options in main gui.

exitgui
Quits and exits the gui.

factor_Callback
Help function to split number of selected frames in to prime numbers.

filtercheckbox_Callback
This refreshes and thus enables filtering to be displayed.

z = getfilenumber(stri)
Take only the last digits of a filename if numbers exist.

files2load = getfiles2load(ind2load)
Gets files to load as a cell removes chache files, dicom dirs, .seg files etc.

getpathinfo
Scans directory and builds a structure of
relevant information. The function populates the file listbox
and stores DATA.Preview.FileList variable
If there is a directory, return the number of files.
If it is a file, simply the filename

Later add also if it is selected.

The generated information is stored in the file folders.cache in every folder Segment visits. This function also handles saving, loading and regenerating this cache when needed.

[hasdcm, description] = getseriesdescription(folder, varargin)

Assumes that folder is a folder with a DICOM image series (one or more dicom files).

In the second optional argument a list of files in the directory can be passed, in the same format as the matlab builtin dir() returns. This is useful when a dir (this function needs the results of one) is expensive, such as when listing very large directories on a network share (why this is expensive nobody knows).

Searches for the first dicom file, and creates a nice string with a description (max. maxlength characters long). Used in getpathinfo.

hasdcm is true if a dicom file was found, false otherwise.
If hasdcm is false, 'Unknown' is returned.

If the function can't come up with a nice description, 'Unknown' is returned.

initset(no)

Initialize the set structure.

keypressed(fignum,evnt)

Helper function to handle keypressed events.

loadandmerge_Callback

Loads one image at a time and attempts to merge them.

loadfiles(dicomfiles, showprogress, cropbox)

Load files.

loadmultidataset(setstruct)

Load .mat file. Sets up display and exits GUI and return back to segment.

loadpreview
Loads a preview of, the function takes a filename or a pathname.

loadpushbutton_Callback
This function is called when user press load.

ote that the function only works for non time resolved ' ... 'images. Are you sure you want to continue?']));
Uturn;.

pathlistbox_Callback(updir)
This function is called when user selects something in the file/dir selection listbox. If called with on input variable then go up one directory level.

plotslicelocation_Callback
Plots graphical display of slicelocation of DICOM files.

refresh_Callback
Refresh GUI.

roisizelistbox_Callback(insize)
Callback for ROI size selection.

selectall_Callback
Select all files callback.

setimagetype_Callback
Set image type callback.

setimageviewplane_Callback
Set image type callback.

setimagingtechnique_Callback
Set imaging technique callback.

setupstacksfromdicom(no)
This function is called when managed to load an image volume from DICOM files. It is essential for enabling, and setting up things.

setupstacksfrommat(no)
This function is called when managed to load an image volume from mat files. It is essential for enabling, and setting up things.

updir_Callback
Fake a double click on ..

15.9.2 Image file reader unit

This is the reader of .mat files and DICOM files. The file `segloader.m` defines a class used for loading files into Segment, the file `segdicomtags.m` defines a class for handling DICOM tags and the file `segrawstack.m` defines a class used for rendering Segmentimage stacks from DICOM data. This class also has a subclass defined in `rotrawstack.m` for loading image stacks where the image plane has been rotated between capture of slices. For loading DICOM images, a `segloader.m` object is created, which in turn creates a `segdicomtags.m` object for keeping track of DICOM tags, and a `segrawstack.m` or `rotrawstack.m` object for rendering an image stack.

Interactions

Interactions are negligible.

Datastructure

No use of shared datastructures occurs.

Functions in `segloader.m`

adddicomfiles(self, filenames)
```
Adds DICOM files to the object.
```

addmatfile(self, filename)
```
Adds a mat file to the object.
```

r = getpreviewdata(s, fname)
```
Gets an element from a struct if it exist and has type char.
 else return ''. Used for generating preview.
```

h = hour(tfrac)
```
Convert from tfrac to hours.
```

[r, imaxis] = isrotated(self)
```
Checks if the loaded files is a rotated image stack.
```

eq = linecmp(line1, line2)
```
Used to compare two lines generated by uniquelines.
 (Accepting some errors).
```

m = minute(tfrac)
```
Convert from tfrac to minutes.
```

curdicoms = removeduplicates(curdicoms)
```
Removes any duplicates in curdicoms.
```

[type, r] = render(self, datapath, cropbox)
```
Renders the images in the loader object to a
 Preview or a set struct suitable for passing on
 to segment.m.
```

[im, desc, filetype, resolutionx, resolutiony, cancrop] = renderpreview(self)
```
Renders a preview of the files in the loader object.
```

r = renderrotstacks(self, datapath, cropbox, lines, imaxis)
```
This rendering method is used when the files loaded are
 dicom files. It's called from the render method.
```

r = renderstacks(self, datapath, cropbox)
```
This rendering method is used when the files loaded are
 dicom files. It's called from the render method.
```

s = second(tfrac)
```
Convert from tfrac to minutes.
```

self = segloader()
```
Constuctor. Initiate all properties.
```

setpreviewmode(self)
```
Set preview mode to on.
```

r = uniquelines(self)
```
Get all the unique lines from the dicom files in the loader object.
 A line consist of the orientation vectors (vector parallel to
 the x and y axis of the picture) and a projection of the
 imageposition onto the subspaces that the orientation vectors
 span. We need this projection to separate projection to
 separate stacks with the same orientation but diffrent positions.
```

r = uniquenormals(self)
```
Gets all the unique normal from the dicom files in
 the loader object. Used in isrotaded method.
```

Functions in segdicomtags.m

a = addstructs(a, b)
Add fields from struct b to struct a.

implicit = checkFunctionalGroup(group, groupnames)
Check functional groups to see if they are implicit.

createnewtags(self)
Create a new struct of tags.

r = getaccessionnumber(self)
Returns the accessionnumber field.

r = getacquisitiontime(self)
Returns the Acquisition Time.

r = getbitsstored(self)
Returns the BitsStored field.

r = getechotime(self)
Returns the Echo Time.

r = getflipangle(self)
Returns the FlipAngle field.

images = getimages(self)
```
Return the images contained in the dicom file.
 The return data will be a struct with the fields
 'im' - The pixeldata as NxM single matrix
 'spacetimepos' - the position of the image in space and time
 'triggertime' - The trigger time of the image
 'instancenumber' - The instance number of the image
 'multiframenumber' - The position in PixelData buffer of
 the image.
```

images = getimagesframetime(self, nFrames)
```
Used by get images when numberofframes are greater
 then one and the frame incremental pointer
 is set to frametime.
```

images = getimagesmultirrtimeslice(self, nFrames)
```
  Used by get images when numberofframes are greater
   then one and the frame incremental pointer
   is set to rr timeslice.
```

images = getimagesmultislice(self, nFrames)
```
  Used by getimages as a default multiframe parser.
```

images = getimagessingleframe(self)
```
  Used by get images when there is only
   one frame.
```

r = getimagetype(self)
```
  Returns the image type.
```

r = getinstancenumber(self)
```
  Returns the instance number of the dicom.
```

line = getline(self)
```
  Gets the line of the image. A line consist
   of the orientation vectors and the projection
   of the position onto the orientation vectors.
```

r = getmodality(self)
```
  Returns the Modality.
```

normal = getnormal(self)
```
  Returns the normal to the picture, that i
   the cross product of the orientation vectors.
```

r = getnumberofaverages(self)
```
  Returns the NumberOfAverages field.
```

orientation = getorientation(self)
```
  Get the orinentation vectors as a 2x3 matrix.
```

r = getpatientinfo(self)
```
  Returns a Patientinfo struct with fields
   'PatientName'
   'PatientID'
   'PatientBirthDate'
```

```
   'PatientSex'
   'PatientAge'
   'HeartRate'
   'AcquisitionDate'
   'PatientWeight'.
```

r = getphotometricinterpretation(self)
```
  Returns the photometric interpretation.
```

pixelData = getpixeldata(self)
```
  Parses and returns the PixelData field.
```

pos = getposition(self)
```
  Returns the image position in 1x3 vector.
```

r = getrepetitiontime(self)
```
  Returns the RepetitionTime.
```

r = getresolutionx(self)
```
  Returns the resolution in the x direction.
```

r = getresolutiony(self)
```
  Returns the resolution in the y direction.
```

scanner = getscanner(self)
```
  Return the scanner based on the Manufacturer tag.
```

r = getsegmentdata(self)
```
  Parses and returns the SegmentData.
```

r = getsequencename(self)
```
  Returns the Sequence Name.
```

r = getseriesdescription(self)
```
  Returns the SeriesDescription.
```

r = getseriesnumber(self)
```
  Returns the Series Number.
```

z = getsliceposition(self)
```
  Returns slice position.
```

r = getslicethickness(self)
Returns slicethickness.

r = getspectspecialtag(self)
Returns a special tag used in SPECT images.

r = getstudyid(self)
Returns study id field.

r = getstudyuid(self)
Returns the study instance uid.

r = gettriggertime(self)
Returns the trigger time of the image.

type = gettype(self)
Find the image type of a dicom image
Old values for type is
0: 'mag', 1: 'phase', 2: Unknown, 3: Unknown.

r = getvelocityencodescale(self)
Returns the VelocityEncodeScale field.

r = getvenc(self)
Returns the venc.

r = getvencpos(self)
Returns the vencpos.

Magnitude images get number 1, through-plane flow gets number 2.
The other directions are more tricky - Philips and Siemens do it differently.

Velocity directions:

PhilipsVENC example	Siemens SequenceName	Flow dir.	Segment vencpos
[100 0 0]	ends in 'rl'	RL/L	2
[0 100 0]	ends in 'ap'	AP/P	3
[0 0 100]	ends in 'fh	FH/S	4

```
      -------        ends in 'in'    (*)     (*)

   *: Siemens datasets so far contain 'in', 'ap' and 'fh' stacks.
      The 'in' stack is just 'through-plane', so we have to figure
      out which direction it is by looking at ImageOrientation.
```

r = haspixeldata(self)
Checks to see that there is pixeldata.

r = hasrepetitiontime(self)
Returns the Repetition Time.

r = hassegmentdata(self)
Returns true if the tag SegmentData is present.

r = hastriggertime(self)
Returns true if has trigger time information.

r = hasvelocityencodescale(self)
Returns true if VelocityEncodeScale field is set.

r = ignoreme(self)
Returns true if this image should be ignored from the loading
process.

r = isduplicate(self, other)
Checks if other is the same as self.

r = parsefloatstr(data)
Parses a string with a number as a
number of type double.

r = parsesingle(data)
Parses a uint8 matrix with four elements as
a single number of type single.

timenum = parsetime(timestr)
Parse a dicom time string as number of seconds.

r = parseuint16(data)
Parses a uint8 matrix with two elements as
a single uint16 number.

dicoms = readfiles(files)
```
Returns a matrix of segdicomtags objects
 contaion DICOM info from 'files'.
```

[s,ind] = removechars(stri)
```
Removes everything except numbers and '.'
 from a string.
```

self = segdicomtags(tags)
```
Constructor. Sets the tags property.
```

r = spacetimepos(self)
```
returns the spacetimepos of the images.
```

switchtags(self)
```
Switch to new tags.
```

r = unpack(self)
```
Unpack data from dicom file.
```

tags2find = unpackhelper(tags2find,nested2find,sequence,explicit)
```
Helper function to unpack data. EH:.
```

Functions in `segrawstack.m`

r = checkifcommon(self, method)
```
Calls 'method' for all dicom files in the
 rawstack object. If all return values are the same
 return true. If not return false.
```

cropbox = fixcropbox(self, cropbox)
```
If cropbox is invalid or empty return a cropbox
 that doesn't crop anything.
```

vencskip = fixphases(self)
```
Make sure phases are numbered from 1 to numphases.
That is remove 'holes' from the phase numbering.
```

r = getaccessionnumber(self)
```
Returns acessionnumber. Takes the accession number from the first
image.
```

r = getacquisitiontime(self)
```
  Return the first ( lowest ) aquisitiontime found
   in the dicom files.
```

r = getbitsstored(self)
```
  Return BitsStored if all dicoms agree else throw error.
```

dimsizes = getdimensionsizes(self, cropbox)
```
  Gets the size of each dimension, i.e number of frames,
   depth, x-size, y-size.
```

r = getechotime(self)
```
  EH: If different values then return a vector
  This is necessary since a "timeresolved" image stack
  may have different echo times. This feature is used for
  T2/T2* mapping. See the function t2star.m for more details.
```

r = getflipangle(self)
```
  Return flipangle if all dicom files agree else
   return 0.
```

r = getifcommon(self, method)
```
  Calls 'method' for all dicom files in the
   rawstack object. If all return values are the same
   return that. If not throw error.
```

r = getimagetype(self)
```
  Returns image type if all dicoms agree else return
   'Mixed Image Types'.
```

r = getmodality(self)
```
  Return modality if all dicoms agree else throw error.
```

r = getnumberofaverages(self)
```
  Return number of averages if all dicom files agree else
   return 1.
```

r = getpatientinfo(self)
```
  Return patient info struct if all dicoms agree.
   If they don't agree try setting all heartrates to
   zero and return if they agree. If that didn't work
   either throw error.
```

r = getrepetitiontime(self)
Return the repetitiontime if all dicom
files agree else return 0.

r = getresolutionx(self)
Return resolutionx if all images agree else throw error.

r = getresolutiony(self)
Return resolutiony if all images agree else throw error.

r = getscanner(self)
Return scanner if all dicom files agree else throw error.

r = getsequencename(self)
Returns sequence name if all dicom files agree else
return 'Mixed Sequence Names'.

r = getseriesdescription(self)
Return series description if all dicom files agree
else return 'Mixed Series Description'.

r = getseriesnumber(self)
Return series number if all dicom files agree
else return 0.

r = getslicethickness(self)
Return slicthickness if all dicom files agree else
return 1.

r = getspectspecialtag(self)
Return spect special tag if all dicoms agree else throw error.

r = getstudyid(self)
Returns StudyID. Takes StudyID from first image. Should be same for
all images in the entire study.

r = getstudyuid(self)
Return study uid if all dicoms agree eles throw error.

r = gettimeincr(self, nFrames)
Calculates the timeincrement. If there is only one frame
return 0. If one can't find any triggertime information
return 1000/(nFrames - 1) (that is totaltime should be 1 second).

[r,sortind] = gettimevector(self,tsize)
```
  Return time vector from acquisiontimes, if available.
```

r = getvenc(self)
```
  Return venc if all images agree or some images
   agree and other specify zero. Else throw error.
```

r = hascollisions(self, imbase, offset)
```
  Returns true if two images in the object gets
   the same coordinates using imbase and offset.
```

stri = hascollisionshelper(self,impos)
```
  Returns error message with appropriate slice distances.
```

r = hasrepetitiontime(self)
```
  Return true if all dicom files has repetition time.
```

r = hastriggertime(self)
```
  Returns true if all dicom files has trigger time.
```

r = ismatch(self, dicom)
```
  Check if a dicom files has the same line as all dicom files
   should have in this stack.
```

[base, offset] = makeimbase(self, dimsizes)
```
  Return a base for the loaded image stack.
```

im = makeimdata(self, imbase, offset, dimsizes, cropbox)
```
  OCombines all images in the stack to an im suitable for SET.im.
```

r = render(self, datapath, cropbox)
```
  Renders the dicom files and images in this stack
   to a preview and set struct suitable for
   passing on to segment.m. If cropbox == []
   load entire image.
```

self = segrawstack(line)
```
  Constuctor. Initiate all properties.
```

setdicoms(self, dicoms)
```
  Takes a list of segdicomtags objects and saves
   them in self.dicoms. Also save the images they
   contain to self.images.
```

settimes(self, dimsizes, timeDist)
```
Set the self.images(:).spacetimepos(4),
 that is time coordiante, according to
 timeDist and a certain sorting.
```

Functions in `rotrawstack.m`

dimsizes = getdimensionsizes(self, cropbox)
```
Gets the size of each dimension, i.e number of frames,
 depth, x-size, y-size.
```

r = hascollisions(self, imbase, offset)
```
Returns true if two images in the object gets
 the same coordinates using imbase and offset.
```

r = ismatch(self, dicom)
```
Check if a dicom files has one of the lines in this stack.
```

[base, offset] = makeimbase(self, dimsizes)
```
Return a base for the loaded image stack.
```

im = makeimdata(self, imbase, offset, dimsizes, cropbox)
```
Combines all images in the stack to an im suitable for SET.im.
```

r = render(self, datapath, cropbox)
```
Renders the dicom files and images in this stack
 to a preview and set struct suitable for
 passing on to segment.m. If cropbox == []
 load entire image.
```

self = rotrawstack(lines, imaxis)
```
Constuctor. Initiate all properties.
```

15.9.3 DICOM file write unit

Writing DICOM files from Segment is implemented by the class `segdicomfile`.

Interactions

Interactions are limited to calling arguments from the class creator.

Datastructure

No use of shared datastructures occurs.

Functions/methods in `segdicomfile.m`

create(filename, data, study_uid, pat_name, pat_id, ...
pat_birth, pat_sex, switchtags)
RCreats a dicom file. Serialize data arg and.

mem = create_chunk(indata)
ESerializes a 1xn cell array of uint8 1xn arrays.

str = create_chunk_va(varargin)
Avarargin shortcut to create_chunk.

r = create_metaheader(instance_uid)
EReturns a memorybasket containing a dicom.

r = generate_uid()
EReturns a new random UID (uses the matlab root UID).

tags = get_tags()
EReturns a tags struct with tag names as fieldnames and tag as value.

tag = name_to_tag(name)
OConvert a tag name to a tag.

outdata = parse_chunk(data)
NInverse of create_chunk.

varargout = parse_chunk_va(data)
Avarargout shortcut to parse_chunk.

stri = secondtostring(t)
Nonverts from seconds to a timestring with hhmmss.sss.

r = serialize(data)
ESerializes a matlab variable. Return a [1xn] uint8 array.

r = serialize_cell(data)
ESerialize a cell array, doesn't mind shape.

r = serialize_char(data)
ESerialize a char array, doesn't mind shape.

r = serialize_logical(data)
ESerialize a logical array, doesn't mind shape.

r = serialize_struct(data)
ESerialize a struct array, doesn't mind shape.

data = unserialize(r)
NUnserialize [1xn] uint8 array r to a matlab variable.

data = unserialize_cell(r)
NUnserialize a cell array, doesn't mind shape.

data = unserialize_char(r)
NUnserialize a char array, doesn't mind shape.

data = unserialize_logical(r)
NUnserialize a logical array, doesn't mind shape.

data = unserialize_struct(r)
NUnserialize struct array, doesn't mind shape.

write_tag(mem, tag_name, vr, data)
RWrites a dicom tag (little endian explicit VR.

15.9.4 Database unit

The purpose of the patient database unit is to allow a simple access to both DICOM data and analysed `.mat` files.

Interactions

The data base is stored in a file called `patientdatabase.mat`. The location where the patient database is set in the preferences. In the unit there is a timer that scans the index file and checks if that have been updated every 30 seconds. It looks at a timestamp to prevent for loading the entire file. In the same folder as where the patient database index file is stored there are three folders:

- `Analysed` this folder contains all `.mat` files. On the highest level there are folders with patient names. In the second level there are the study date, that each is a folder that contains `.mat` files. Characters that are not valid filename are removed by the function `removeforbiddenchars.m`.
- `DICOM` this folder contains all DICOM files. They are sorted into subfolders. The files are sorted according to the following system, where on the top level there are folders with patient name with an underscore and patient id (i.e `ALF_A_Bete_19730101010101`). The names are converted to only allow valid filename characters and spaces are changed to an underscore. Removing forbidden characters are performed by the function `removeforbiddenchars.m`. On the second level there are the studydate, a dash the characters `ID`, a dash and the for last digits of the studyuid (i.e `20061204-ID-7232`). On the next level there are folders with series and in each folder there are DICOM files. In this folder there is also stored a file called `thumbs.cache`, this file contains thumbnail previews of all the series. For more details, see the file `thumbnails.m` for details. If changes are made to the design of this filestructure, corresponding changes should also be made in the Segment DICOM server.
- `Report` this folder stores the `html` reports created by the Report Sheet unit. Please see the Report Sheet unit for further details.
- `TEMP` this folder is used to temporarily retrieve images in the PACS connection. When images have been retrieved they are automatically moved to the patient database. It should not be necessary to manually remove files from this folder.
- `TEMPSTORAGE` this folder is used by the Segment Server to temporarily store received files. After sorting they are automatically stored in the patientdatabase. It should not be necessary to manually remove files from this folder.

Datastructure

The patient database function contains a global variable called `DB`.

- `Handles` stores the handles of the GUI.
- `NumStudies` contains the number of studies.

- `CurrentStudy` points to the current study.
- `SortList` is a vector and contains the current sorting of the studies.
- `ExportList` contains a list of studies to be exported to PACS.
- `Hostname` contains hostname for exporting studies.
- `Port` contains port for exporting studies.
- `Called_AE` contains Called_AE for exporting studies.
- `StoreSCU` contains the StoreSCU for exporting studies.
- `folder2load` contains the folder to load data from, used in importing studies from disc or CD.
- `DateNum` contains a timestamp when the database was last loaded from disc.

The datastructure for the patient database is as follows:

- `PathName` a full path to where the study is stored.
- `IsDicom` binary variable, true if the study is in DICOM format.
- `Name` a string containing the name of the subject.
- `ID` a string containing the ID of the subject.
- `Sex` a string containing the sex of the subject. May also be empty if not known.
- `StudyDate` a string containing the study date as reported in the DICOM files or in the `.mat` file.
- `ReceivedDateNum` the time when the study was imported or stored in the database in numeric format (same format as the `now` command).
- `ReceivedDateString` the time when the study was imported or stored in the database in plain text format.
- `Modality` as recorded in the DICOM or in the `.mat` file.
- `FileFormat` is either `Dicom` or `Segment`.
- `CommentPath` reserved for future use, currently empty.
- `Mem` contains the amount of memory the study takes on disk (in bytes).

If the database needs to be moved, all the references in the database needs to be recomputed. This can be done by rebuilding the database (available from the menu). This also helps if the database for some reason have become corrupted.

Functions

The function is called `patientdatabase.m`, and also calls the function `patientdatabaseaddstudyhelper.m`.

15.9.5 PACS unit

The purpose of the PACS unit is to enable Segment to connect to a hospital PACS system and retrieve images. The functionality of the PACS connection unit is documented in detail in the Patient Database Manual and PACS Communication Manual.

All low level communication is performed by the DCMTK toolkit (for documentation see http://support.dcmtk.org/docs/index.html). Searches can be performed both on patient level and study level.

Interactions

All interactions with Segment is performed through the patient database.

- The connections are stored as structs in `.mat` files, and is stored in the Segment main folder.
- Logfiles are stored in the same folder as where the preferences are stored.
- It is also possible to store batch files for downloading. They are also stored as `.mat` files.
- Temporarily files in the download process are stored into the folder `TEMP` located in the same place as where the Patient Database is stored (this is configured in the preferences menu).

Datastructure

The internal data used by the GUI is stored in the mygui struct.

Functions

The PACS connection unit uses slaves that operates as different processes for the actual retreival and then in a loop checks that all the files have been received and updates the waitbars as appropriate. This usage of slaves is to prevent Segment to just go into a sleep mode while retrieving since we use DCMTK binaries for the recieve operations. This also have the advantage of splitting the work over multiple cores if available. The communication between the slaves are performed by inbox files and outbox files. These inbox files are stored in the same folder as where the preferences are stored. The underlying client-server functions are implemented in the class `myclientserver.m`. The communication is as ordinary ascii files and the protocol is defined in `encodemessage.m`, and `decodemessage.m`. The actual work is performed by a compiled function called `slave`.

Adding studies to the patient database is done by calling the function `patientdatabaseaddstudyhelper.m`.

15.9.6 Segment Server unit

The Segment Server unit is a standalone software that turns the computer into a DICOM server that can receive image data directly from an MR scanner.

Interactions

This unit interacts with the Database unit to add received studies to the patient database.

Datastructure

The `DATA` variable is used to extract network options. The unit also uses its own global variable `SERVER` to keep track of its state.

Functions

Functions are contained in the main file `segmentserver.m` as well as the file `segmentserverhelper.m` used for interactions.

15.9.7 File menu unit

The File menu unit contains functions called from the File menu.

Interactions

This unit calls the Openfile and Database units when the user accesses them through the File menu.

Datastructure

No use of shared datastructure occurs.

Functions

closecurrentimagestack_Callback(frompreviewmenu)

```
Close current image stack, i.e the current image stack is deleted. It
also takes care of eventual cross couplings between image stacks.
```

loadednext = loadnext_Callback

```
Load next .mat file in the current data folder.
```

loadsegmentation_Callback(pathname,filename)

```
Loads segmentation to current image stack from a .seg file.
```

quit_Callback

```
Quit Segment.
```

saveall_Callback

```
Saves all image stacks to one .mat file. Calls GUI method
filesaveallas_Callback which is the workhorse when saving image stacks.
```

fail = saveallas_helper(pathname,filename,topatientdatabase)

```
Save all image stacks to the file specified. It also stores current view
and modes etc.
```

savecurrent_Callback
Save current image set to file. Note that this is the old Segment file format and this fcn may soon be depreciated.

savesegdicom_Callback(filename)
Save image stack as DICOM file.

savesegmentation_Callback(pathname,filename)
Saves segmentation as a .seg file. This way of saving contours is not recommended and may be depreceiated.

savetopacs_Callback
Send image stacks to PACS. This function should display a list of available PACS (.con files) and when user has selected store files on disk temporarily and then send the files to the PACS.

savetopatientdatabase_Callback
Callback to save image stacks to patientdatabase. Uses functions in patientdatabase.

15.9.8 Utility unit

The utility unit contains tools that operates of files (`.mat` files or `DICOM` files) rather than the loaded image data.

Interactions

The Calculation superunit is called to calculate image data. The Open File unit is called to open files.

Datastructure

The utility uses and also generally resets the `SET` structure so that unsaved image data is lost.

Functions

'This function recommends that you set RV insertion points in order to have the bullseye correctly rotated. If no insertion points are marked then current rotation are taken.');

'This function recommends that you set RV insertion points in order to have the bullseye correctly rotated. If no insertion points are marked then current rotation are taken.');

ahaexporthelper(type)
```
Helper function to export in 17 segment model.
```

type = basalselectionhelper
```
Helper function to select what to do with basal slices.
```

clearsegmentationmultiple_Callback
```
This function clears the segmentation in multiple .mat files.
This is useful for instance in research and when second observer
analysis is required.
```

newv = dividein3(v)
```
Divide vector in 3 parts and take mean over them.
```

findindicomfiles_Callback
```
recursevily finds .mat files in selected directory and exports data on
patient info and file location. Uses some heuristic to avoid checking all
files.
```

findinmatfiles_Callback
```
recursevily finds .mat files in selected directory and exports data on
patient info and file location.
```

init
```
Initialize utilitymenu, called upon starting of Segment.
Calls private initialization method which adds private
utilities for Medviso AB and Lund Cardiac MR Group.
```

marexport_Callback
```
Export mar in 17 segment model.
```

[nfound,line] = numexist(outdata,rows,name,id)
```
Helper function to findindicomfiles.
```

scarexport_Callback
```
Export scar in 17 segment model.
```

[startslice,endslice] = sliceselectionhelper(no,type)
Helper function to select slices depending on how to include basal slices. See basalselectionhelper for further details.

twelvesectorexport_Callback(type)
Helper function to export in 12 segment model.

utilityanonymizemultiple_Callback(pathname,type)
Anonymize multiple datasets from .mat files in a folder.

utilitycopyandsortfromcd_Callback
Utility function to copy and sort files from CD. Uses helper function dicomsorter to do the work.

utilitygrayzone
Add grayzone utilities.

utilitygrayzoneexport
Export grayzone analysis values.

utilitygrayzonesliceexport
Export grayzone analysis values for each slice.

wallthicknessexport_Callback
Export wall thickness in 17 segment model.

15.9.9 Sectra unit

The Sectra unit contains the Segment side implementation of the Sectra PACS plugin. It consists of a timed function that repeatedly checks for incoming data from a Sectra PACS system (once every second). This is started in the function `sectra.m`.

Sectra Process

On the sectra side, there is a code called segment `segmentplugin32.dll` or `segmentplugin64.dll` that is upon starting the Sectra interface is loaded and associated with the process. Once the user clicks on Clinical applications and Segment, then the code is invoked.

The first thing the dll files does is look for an open Segment (this is done by looking for window names using Windows API). It looks for windows that starts with '`Segment`' followed by space. Note that this might need to be changed when renaming future products. This check is done in `pipe.cpp`.

The principle is that the plugin sends all DICOMs on the series level to the Segment process by the usage of pipes. First a 'dummy' DICOM is sent as preview file. This was required for the old DICOM file loader. The DICOMs to send is found by parsing the Sectra API and asking for images. Please not there seems to be several bugs in the API that crasches Sectra. This is documented in the code. The challenge in the process is to convert data from Sectra API to usable DICOM files. The documentation to the Sectra API is on their SDK and in this there is a Windows help file. The SDK needs to be installed in order to be able to compile. For details on how to compile the plugin, please see Chapter 18. The SDK is available from `http://userweb.sectra.se`. Username and password is documented in `wiki:SectraPlugin`. The name of the code and details needs to be same for 32 and 64 bit. If both are registered then Sectra is able to automatically determine the correct version to run.

Interactions

This unit calls the Openfile unit to load data from the Sectra PACS.

Datastructure

No use of shared datastructure occurs.

Functions on Segmentside

initsectra

```
Starts the SECTRATIMER. The timer
 executes function loadsectra every second.
```

loadsectra

```
Load files from sectra pacs.
```

varargout = sectra(varargin)

```
Switchyard for sectra module.
```

stopsectra
`Stops the SECTRATIMER (if it's running).`

Files on plugin side

- `maindll` This is a main template file from the SDK.
- `sectracontrol` This file is related to the interface such as status texts etc.
- `pipe` This is the main file written by Jonatan. First it finds Segment, and then starts Segment(unless it is already started). The file contains of the following sections:

  1. `write_info` writes the control file/info file
  2. `write_dicom` called by `server_thread`
  3. `server_thread` this file can be seen as the main function that created the pipes by calling the subfunction `create_named_pipe`. The function `Connect_Named_Pipe` takes time and waits for some one to read.

- **write_dicom** Writes the DICOM files. The term `dicomattrib` is the Sectra name standard and corresponds approximately to dicomtags. The Sectra term Pyramid corresponds approximately to one DICOM file. The preview file is as mentioned above legacy and is not used. Each time a file is sent them check for errors are performed. The function `SendVR` uses `get_VR` by Sectra API. The function `SendData` sends data depending on VR and peforms changes in the data representation.

15.9.10 External PACS unit

The External PACS unit consists of a Segment side implementation of the External PACS plugin and a standalone application that is called when data is sent from the external PACS system, starts Segment (if not already open) and sends instructions to it on what data to open. The Segment side implementation consists of a timed function that repeatedly checks for incoming data from the standalone application.

Interactions

This unit calls the Openfile and Patient Database units to load data from the external PACS.

Datastructure

No use of shared datastructure occurs.

Functions

varargout = externalpacs(varargin)
```
Switchyard for EXTERNALPACS module.
```

initexternalpacs
```
Starts the EXTERNALPACSTIMER. The timer
 executes function loadexternalpacs every second.
```

loadexternalpacs
```
Load files from EXTERNALPACS pacs.
```

stopexternalpacs
```
Stops the EXTERNALPACSTIMER (if it's running).
```

15.10 Analysis superunit

The Analysis superunit contains functions for performing special kinds of analysis on image stacks.

15.10.1 Relaxometry unit

The purpose of the the relaxometry unit is to analyse MR relaxometry, and specifically T1, T2, and T2* relaxometry.

Interactions

Interaction by reading data directly from SET structure and store data there.

Datastructure

The image information on which to operate is taken from the `SET` struct and the stack number is taken from `NO` variable. `SET` is also used for storing the derived maps.

Functions

'Error: Unknown fitmode input to function "getresulttext". ')

```
Uturn;.
```

'Error: Unknown fitmode input to function "getresulttext". ')

```
Uturn;.
```

y = MAGIR(P,t)

```
Babs(P(1)-P(3)*exp(-(t)/P(2)));.
```

y = MAGIR2(P,t)

```
Babs(P(1)*(1-P(3)*exp(-(t)/P(2))));.
```

y = MAGIR_2p(P,t)

```
Babs(P(1)*(1-2*exp(-(t)/P(2))));.
```

y = MAGSR_2p(P,t)

```
Babs(P(1)*(1-exp(-(t)/P(2))));.
```

y = PSIR(P,t)

```
P(P(1)-P(3)*exp(-(t)/P(2)));.
```

y = PSIR2(P,t)

```
P(P(1)*(1-P(3)*exp(-(t)/P(2))));.
```

y = PSIR_2p(P,t)

```
(P(1)*(1-2*exp(-(t)/P(2)));.
```

y = PSSR_2p(P,t)

```
(P(1)*(1-exp(-(t)/P(2)));.
```

mask = calcmask(region)

```
Calculate mask depending on selection of regional restriction
(myocardium, ROI's or full image).
```

calctxmap(sliceno)

```
Calculates a map.
```

climedit_Callback(v)

```
Callback for edit box of display limit.
```

climslider_Callback(v)
Callback for slider to set display limit.

close_Callback
Close GUI.

colorbarhelper
Helper function to display colorbar.

copyroislices(n)
copy ROI one slice downward.

createimagestacks_Callback
Output of image stacks containing raw T2/T2* map, smoothed T2/T2* map and resolution map.

endline_Buttondown
Buttondown function for line indicating end of time period to fit.

endline_Buttonup
Buttonup function for line indicating start of time period to fit.

endline_Motion
Motion function for line indicating start of time period to fit.

export_Callback()
Export data to clipboard.

titlestring = getresulttext(guiname, k, tx, sat, info)
Updates result string.

globalfit
Plots a global fit plot.

init(varargin)
mode = 1, 2, 3; 1 gives T1; 2 gives T2; 3 gives T2*.

magpoint_Buttondown
User clicked on image gives show fit.

magpoint_Buttonup
Buttonup function for magpoint in image.

magpoint_Motion
Motion function for dragging magpoint in image.

menu_nbrofparameters_Callback(nbrofparameters)
Get number of parameters from menu.

menu_plothist_Callback(numbinsin)
Callback to plot histogram.

menu_roiselect_Callback(roinbr, asktocopy)
manual select which ROI to perform Tx analysis on.

y = monoexp(P,t)
(P(1)*exp(-(t)/P(2));.

y = monoexp_offset(P,t) %Magnitude images only since we assume the offset is a positive value.
(P(1)*exp(-(t)/P(2))+abs(P(3));.

movedown_Callback()
Move down callback.

moveleft_Callback()
Move left callback.

moveright_Callback()
Move right callback.

moveup_Callback()
Move up callback.

[dx, dy, xlim, ylim] = moveview_module(ax,dxfactor,dyfactor)
Function to move zoomed in views in the T2* module.

[xlim, ylim] = moveview_restore(ax,dx,dy)
Restore the old view.

parameter_Callback(mode)
Callback from parameter selection radiobutton.

plotfit_Callback(fitmode)
update the fit plot according to mode.

point_down
New function to control up movement using the keyboard (w):.

point_left
New function to control up movement using the keyboard (w):.

point_right
New function to control up movement using the keyboard (w):.

point_up
New function to control up movement using the keyboard (w):.

recalc_Callback(slicestocalc, waitbartext)
Recalculate previously processed maps and make graphical updates.

data2 = resortsegdatatime(data)
Resort the data into the correct order.

restriction_Callback(type)
IMPORTANT: Check compatibility with function "setregionrestriction"
Callback for selection of regional restriction (myocardium, roi or full image).

resultpoint_Buttondown
User clicked on image gives show fit.

resultpoint_Motion
Motion function for dragging resultpoint in image.

dofullmap = setmasks()
Bobal DATA guiname.

setregionrestriction
Set region restriction based on available contours.

settimeslicesliders
Set the time slides.

sliceslider_Callback(v)
Callback for slider to set time frame in echo image stack.

smoothth resedit_Callback
Callback for edit box of smoothing (error) threshold.

smoothth resslider_Callback
Callback for slider to set smoothing (error) threshold.

startline_Buttondown
Buttondown function for line indicating start of time period to fit.

startline_Buttonup
Buttonup function for line indicating start of time period to fit.

startline_Motion
Motion function for line indicating start of time period to fit.

P = t1firstguess1(S, minusti,INV, nbrofpixelsinslice)
Laling = max(abs(S),[],1); %1 x N.

P = t1firstguess1psir(S, minusti,INV, nbrofpixelsinslice)
Laling = max(abs(S),[],1); %1 x N.

P = t1fit(minusti, s, doinversion, psirtrue, nbrofunknowns, dollcorr)
Do t1 fit.

t2cpmgcheckbox_Callback
Callback function for checkbox which enables T2-CPMG correction when using MESE sequences (Multi-Echo Spin-Echo). /SB.

P = t2fit(minuste, s, nbrofunknowns)
First guess using truncated log-Least-squares fitting:
---.

timeedit_Callback(v)
Callback for edit box of time frame.

timeslider_Callback(v)
Callback for slider to set time frame in echo image stack.

updatefit
Update the fit graphically.

updateimage
```
Update view of image.
```

updatemap
```
Updates the map plot.
```

viewrestore_Callback()
```
Restore the view callback.
```

[xlim, ylim] = zoom_module(ax,f)
```
Added code for Zooming functionality in the map image in the T2star
module. 99% of the code is directly taken from JTu's code in segment_main.
```

zoomin_Callback()
```
Zoom in callback.
```

zoomout_Callback()
```
Zoom out callback.
```

15.10.2 Perfusion unit

The purpose of the Perfusion unit is to provide tools for myocardial perfusion analysis, including rotation, registration and extracting curves of signal intensity over time for quantification of upslopes to compare between rest and stress image stacks.

Interactions

The Calculation superunit is used to do calculations of slice positions, intersections and signal intensities by sector. The Find unit from Helper functions is used to identify cine and scar image stacks.

Datastructure

The image information on which to operate is taken from the `SET` struct. The field `Perfusion` is used for storing the final quantifications of upslope comparisons.

Functions

ScrollWheelFcn(hObject, eventdata)
Scroll wheel function of GUI. Moves slider to toggle between slices.

bloodpoolcheckbox_Callback(field)
Callback for checkbox toggling display of bloodpool curve and upslope.

[smoothed,t] = calcsmoothing(curves,tvec,tups,sigma)
Calculate smoothening (tups = time units per second).

calcupslopes
Calculate upslope curves and draw plots and bullseye.

close_Callback
Callback for closing GUI.

contourcheckbox_Callback
Callback for checkbox toggling display of segmentation contours.

contractsegmentationpushbutton_Callback(field)
Create ROI's defining contracted area of segmentation, to be used, if available, instead of LV data.

curvepopupmenu_Callback
Callback for changing sector to plot. Update both curve plots.

drawbullseye(ax,txth,perfvec)
Outline an AHA bullseye plot in axes with handle ax
Also updates report text with handle txth.

drawimages(field)
Do an update of all image axes.

drawlongaxisimage(zslices)
Update axis containing longaxis image, if available.

export
Export data to clipboard.

figure1_KeyPressFcn(hObject, eventdata)
Keypress functions for GUI.

generatepushbutton_Callback
Callback to generate new aligned perfusion image stacks containing timeframes between user defined start and end points.

stricell = getahastring
Return cell containing strings of segment names from AHA 17 segment model.

result = getbloodpoolcurve(no)
Calculate signal intensity curve of bloodpool.

[endox,endoy,epix,epiy] = getroisegmentation(no)
Get myocardium segmentation from ROI generated by contraction function.

init
Initiate GUI.

initimageaxis(field)
Initiate image axis with images from current stack.

maxslope = initplot(field,ahamat)
Initiate plots of curves.

roth = initrotatehandle(h,field,midpoint)
Draw handle used for rotation of image.

initsliceslider(zsz)
Initiate slider for toggling between slices in display.

inittimebar(field,analysed)
Initiate timebar axis for image specified by input parameter 'field'.

nextpushbutton_Callback(field)
Callback for next frame pushbutton for image specified by input parameter 'field'.

playalltogglebutton_Callback(hObject, ~)
Callback for toggle button playing rest and stress image stacks in a synchronized manner where they reach their end timeframes simultaneously.

playtogglebutton_Callback(hObject,~,field)
Callback for play togglebutton for image specified by input parameter 'field'.

prevpushbutton_Callback(field)
Callback for previous frame pushbutton for image specified by input parameter 'field'.

rawalignedpanel_SelectionChangeFcn(hObject,eventdata,handles)
Selection change function for raw/aligned radiobutton panel. Currently not implemented.

rotatecheckbox_Callback
Callback for checkbox toggling rotation mode.

rotatehandle_ButtonUpFcn(hObject, ~, rotobj, midpoint, field)
Buttonup function for handle used to rotate image specified by input parameter 'field'. Sets sector rotation and dectivates motion function.

rotatehandle_Buttondown(hObject,~,midpoint,field)
Buttondown function for handle used to rotate image specified by input parameter 'field'. Activates motion function.

rotatehandle_MotionFcn(hObject, ~, rotobj, midpoint, field)
Motion function for handle used to rotate image specified by input parameter 'field'. Dragging rotates image.

settimeframe(tag,tf,field)
Sets current/start/end timeframe for image stack specified by input parameter 'field'.

sliceslider_Callback
Callback of slider for toggling between slices in display.

smoothslider_Callback
Callback of slider for changing smoothening factor. Recalculates upslopes.

timebar_ButtonDownFcn(hObject, ~, field)
Buttondown function for graphical timebar object of image specified by input parameter 'field'. Activates dragging of timebars.

timebaraxes_ButtonDownFcn(hObject, ~, field)
Buttondown function for timebar axes of image specified by input parameter 'field'. Changes current timeframe to the one closest to position of clicked point.

timebaraxes_ButtonUpFcn(hObject, ~)
```
Buttonup function for timebar axes of image specified by input
parameter 'field'. Deactivates dragging of timebar.
```

timebaraxes_MotionFcn(hObject, ~, tbobj, no, field)
```
Mouse motion function for timebar axes of image specified by input
parameter 'field'. Used for dragging timebars to change current
timeframe or start/end points of timeframes in which to align images.
```

setstr = timestripset(setstr,tsz,t0,t1)
```
From a SET struct that has been stripped of some timeframes, removes
information specific to discarded timeframes.
```

updateplot(field)
```
Update plot of upslopes from image specified by 'field' argument.
```

updaterotatehandle(midpoint,ang,h)
```
Update rotation handle specified by handle 'h' from input arguments.
```

updatetimebar(h,no)
```
Update timebar axis specified by handle 'h' from input arguments.
```

15.10.3 Strain unit

The Strain unit allows to calculate strain from velocity encoded phase contrast images.

Interactions

The function interacts with the Helper function unit and the Calc unit.

Datastructure

The module uses an own data structure in the field `SET.Strain`.

Functions

[timestepapex timestepapicalr timestepapicall timestepmidr timestepmidl timestepbasr timestepbasl] = ...
calcmean(segmentindex,data)
```
calculate all segments meanvalue at every timestep
varible data is the parameter to calculate the mean over.
```

addicon_helper(callback,tooltip,cdata,tag,separator)
Helper function to add an icon.

[P,pos] = addlocal_helper(P,pos,arclen,localangles,radx,rady,shearx,sheary,outx,outy,xi
Helper function to add local basis functions.

[xnew,ynew] = anatomical2straincoords(x,y)
Convert from anatimical coordinates to strain coordinates.

anatomicalcorrectionpoint_Buttonup
This is called on mouse up.

anatomicalcorrectionpoint_Motion
This function is called when user moves the point.

anatomicalcorrectionpoints_Buttondown
Called when click on correction point in anatomical image.

l = arclength(x,y)
Returns arclength of a curve.

[p1,p2,gm] = avpointsfinderhelper(dc)
Finds two peaks p1,p2 of a signal dc. gm is a goodness measure.

bullseye_Callback
Plots bullseys plot of strain data.

[alpha,arclen] = calcalpha(no,centerx,centery)
Calculates the localangle for boundary node
x,y is boundary points
alpha is angle between normal and x axis.
arclen is arclength in longaxis direction.

calcandview_Callback
Calculate strain in one long-axis image and open the strain gui.

inside = calcinside(no,inside,x,y)
Calculates which points are inside of centerline.

raddist = calcraddist(x,y,centerx,centery)
Calculates radialdistance to node points x,y given the
centerline centerx,centery.

[centerx,centery,inside] = centerline(x,y)
Calculates center line and returns wich points
x,y is contour.

clearall_Callback
Clear all strain data for current image stack.

clearandcalculate_Callback
Callback to re-initialize mesh.

closestrain_Callback
close the strain gui.

index = contourindex2node(no,x,y)
Find connection between points on contour and boundary points.
index is length contour.

correctionpoint_Buttonup
This is called on mouse up.

correctionpoint_Motion
This function is called when user moves the point.

correctionpointbuttonuphelper(x,y)
Helper function to correctionpoint//_Buttonup and
anatomicalcorrectionpoint/_Buttonup. x,y are assumd to be in strain
coordinates.

correctionpoints_Buttondown
Called when someone clicks on a point on strain image.

correctionpointsbuttondownhelper(x,y,motionfcn,upfcn)
Helper function for buttondown for correction points.

ok = errorchecking(no)
Error check before calculate strain. Checks for existance of segmentation,
and if not try to import segmentation from cine images.

exportlongaxis_Callback
export strain data from one long-axis projection to clipboard.

exportstrain_Callback
Read in strain from files in a folder and export them to clipboard.

exporttocine_Callback
Written by Einar Heiberg

Export strain segmentation to SSFP image.

apex = findapex(x,y,pmin,pmax)
Find apex of the heart.

[pmin,pmax] = findavpoints(x,y)
Rewritten by Einar Heiberg, insired by old code by Helen Soneson

Find max curvature, max "curvature" is at AV-plane.

ind = findcine(no)
Helper function to find corresponding cine image.

[minrange,maxrange] = findrange(colorval)
Helper function to find the range for colordata, uses checkboxes also in GUI.

[x,y] = getanatomicalcontour(frame)
Converts the strain contour to an anatomical contour so that it can be plotted on the anatomical image. This code is heavily inspired by importsegmentation in segment_main.

outframe = getanatomicalframe(frame)
Returns timeframe in anatomical image that corresponds to frame.
frame is typical current time frame of strain image.

[xofs,yofs,fx,fy] = getconversionfactors
Get conversion factors between strain coordinate system and anatomical system.

colorval = getparameterdata
Returns the parameter data that should be plotted.

l = getpointspos(n)
Distribute the points evenly.

[radstrain,longstrain,totstrain] = getsectors(strainno)
Get strain in 17 sector format.

graph_Callback
plots strain over time from one long-axis projection.

importfromcine_Callback
Written by Einar Heiberg

Import segmentation from SSFP image.

initgui(no)
Initiates and open the strain GUI.

initialize(no)
This function initializes the mesh using data from SET(no).EndoX.

[pp,ee,tt] = initializemesh(x,y)
Initializes mesh.
x and y are the curve of the myocardium. Uses initmesh in PDE toolbox.

keypressed(fignum,evnt)
Keypress callback.

laplaceradiobutton_Callback
User clicked laplace radiobutton.

multiplebullseye_Callback
Plut multiple bullseyes from clipboard.

navierradiobutton_Callback
User clicked Navier radiobutton.

nedit_Callback
Edit number of points (n).

nslider_Callback
User changes the nslider.

nyedit_Callback
User made changes in edit box.

nyslider_Callback
```
User drags the nyslider.
```

[u,c] = optimizingcoeff(no,vx,vy,A,c,K,H)
```
Optimze the coefficients, run an optimization scheme.

vx is velocity field
vy is velocity field
c is coefficent matrix.
```

plothelper(type)
```
Helper function to plot parameter. Uses the information in the
handles.parameterlistbox.
```

plotmax_Callback
```
plot the strain value from the time frame with abs(maximum strain) over time.
```

plotmin_Callback
```
plot the strain value from the time frame with abs(maximum strain) over time.
```

polarplotaha(strainvalues,ax)
```
plot the aha model in the polar plot.
```

dist = projection(p,plane)
```
Calculates distance d to plane given by p1,p2

Calculate nhat (normal of plane).
```

propagatendoseg(no)
```
Propagate node points.
```

recursekeypressfcn(h,fcn)
```
Helper function to create callbacks to keypressed function.
```

resetstrain(no)
```
Clear strain data with new format for the internal representation.
```

retrack(no)
```
Recalculates the mesh taking corrections into account.
```

retrack_Callback
```
This is called after manual corrections using both local and global effect.
```

v = samplevelocity(vx,vy,px,py,A,c)

```
Sample velocity data, v is vector with sampled velocity data. Size is
(2*nnodes) x nframes.
```

segmentindex = section(no)

```
Divide the long-axis of the heart into 7 segments according to AHAs
17-segment model
segmentindex: contain the index for the 7 different segments:
     2CH: 1.basal anterior  2. mid anterior  3.apical anterior
          4.apical inferior  5.mid inferior  6. basal inferior  7.apex
     3CH: 1.basal anteroseptal  2.mid anteroseptal  3.apical septal
          4.apical lateral  5.mid inferolateral  6.basal inferolateral  7.apex
     4CH: 1.basal inferoseptal  2.mid inferoseptal  3.apical septal
          4.apical lateral  5.mid anterolateral  6.basal anterolateral  7.apex

segmentindex is a cellarray with length 7.

Written by Helen Soneson 2008-09-29
Modified by Shruti and Einar after change in strain representation
Modified by Einar to use dicom coordinates.
```

[Dx,Dy,M] = setupdiffoperator(no)

```
This function is inspired by pdegrad, but returns the operator
matrix instead of actually calculating the gradient.
```

G = setupmappingfunction(no)

```
G is mapping from boundary functions to the entire domain. The size of
G is (2*nodes x 2*boundarynodes).
```

L = setupmaterialoperator(no, Dx, Dy, M)

```
We will use the laplace operator for a start.
Calculate second derivative operator.
Size of Dxx is (nnodes x nnodes).
```

P = setupnewboundaryfunctionsglobal(no)

```
This is the basis function used in the strain paper 2012-08-14.
```

H = setuptemporaloperator(no)

```
H is dtt operator. The size is (nframes-1) x (nframes-1).
```

sigmaedit_Callback

```
Edit box for sigma.
```

sigmaslider_Callback
Callback for the sigma slider.

[xnew,ynew] = strain2anatomicalcoords(x,y)
Converts from coordinates in strain coordinate system to anatomical coordinate system.

[Srad,Slong,Sshear,Stot] = straintensor(no,dx,dy)
Calculates and decomposes the strain tensor to radial, long and shear strain.

timeslider_Callback
Called when user changes timeslider.

trackandcalculateglobal(no)
Perform tracking and calculate strain.

trackandcalculateglobaleval(no)
Track without using user interface.

trackmanualandcalculate()
Recalculates the mesh taking corrections into account.

translatecontour_Callback(direction)
Callback to translate anatomical contour.

translatestraincontour_Callback(direction)
Translate the strain contour callback.

translatestraincontourhelper(no,dx,dy)
Helper function to translate strain contour.

undomanualcorrections_Callback
Undo manual corrections. Currently not implemented returns messagebox.

viewlistbox_Callback
Called when user clicks in view listbox. Currently not fully implemented and not called.

viewlocal
Visualize the local basis regionality.

viewnext_Callback
User has clicked on next button. Move to next time frame.

viewparameter_Callback
Called when user adjusts colorscale or setting gui.

viewplay_Callback
play a movie of strain over time.

viewplaypushbutton_Callback
Start to play from pushbutton.

viewprev_Callback
move to previus time frame.

viewtrackupdate
Update the tracking graphically.

15.10.4 Strain tagging unit

The Strain tagging unit allows to calculate strain from tagging images.

Interactions

The function interacts with the Helper function unit and the Calc unit.

Datastructure

The module uses an own data structure in the field `SET.StrainTagging`.

Functions

analyse_Callback
Do strain analysis.

calcpoints
Calculate positions of points used for strain calculation.

close_Callback
Close straintagging GUI.

hideanatomicalcontour_Callback
Hide segmentation contours in anatomical image.

hidetaggingcontour_Callback
Hide segmentation contours in tagging image.

hidetagginggrid_Callback
Hide grid in tagging image.

hidetaggingpoints_Callback
Hide points in tagging image.

importfromcine_Callback
Import segmentation to tagging image from cine stack.

init
Initialize the GUI.

inittimebar
Initiate timebar axis.

makeslicemat
From the current slice, make .mat file used by the executable.

makevtk
Save tracking grid and points to .vtk files used by the executable.

next_Callback
Callback for changing to next time frame.

play_Callback
Callback for play togglebutton.

prev_Callback
Callback for changing to previous time frame.

sectorrotation_Callback
Show handle for sector rotation.

sectorrotationslider_Callback
Rotate sector.

sliceslider_Callback
Callback for slider to toggle slice.

timebar_ButtonDownFcn(hObject, ˜)
Buttondown function for graphical timebar object. Activates dragging of timebars.

timebaraxes_ButtonDownFcn(hObject, ˜)
Buttondown function for timebar axes. Changes current timeframe to the one closest to position of clicked point.

timebaraxes_ButtonUpFcn(hObject, ˜)
Buttonup function for timebar axes of image specified by input parameter 'field'. Deactivates dragging of timebar.

timebaraxes_MotionFcn(˜, ˜, tbobj, no)
Mouse motion function for timebar axes of image specified by input parameter 'field'. Used for dragging timebars to change current timeframe or start/end points of timeframes in which to align images.

translatecontour_Callback(direction)
Translate segmentation contours in all image stack views.

translategrid_Callback(direction)
Translate grid in tagging image.

updateimages
Update all image stacks and also timebar.

updateplot
Plot strain in axes.

updatetimebar
Update timebar axis.

viewparameter_Callback
Callback from listbox selection of parameter to plot.

15.10.5 MPR unit

The purpose of the MPR unit is to do multiplanar reconstructions of image stacks

Interactions

Interactions are negligible.

Datastructure

The image information on which to operate is taken from the `SET` struct and the stack number is taken from `NO` variable.

Functions

buttonup
`Called upon release of mouse button after center point has been pressed down.`

center_buttondown
`Called when center point is pressed down, sets motion and buttonup function.`

center_motion
`Called upon mouse motion after center point has been pressed down.`

done
`Resample the complete volume.`

init
`Initiate GUI.`

newcut
`Callback for making a new cut.`

p1_buttondown
`Called when point p1 is pressed down, sets motion and buttonup function.`

p1_motion
`Called upon mouse motion after point p1 has been pressed down.`

p2_buttondown
`Called when point p1 is pressed down, sets motion and buttonup function.`

p2_motion
Called upon mouse motion after point p2 has been pressed down.

play
Called when start play.

prevcut
Callback for changing to previous cut.

z = resample_cut
Do the resampling based on cut selected by user.

z = resample_slice
Calculates the slice, i.e left image panel.

[z,impos] = resampler(im,region,sz,no)
Calculate new image coordinates for resampled image.

reset_line(dummy)
Line reset callback.

resolutionedit
Callback for when resolution is changed by user.

sliceslider
Callback when slider is used to change slice.

slicethicknessedit
Callback for when slice thickness is changed by user.

update_all
Update everything on display.

update_line
Update line upon changed line specifications.

update_main(dummy)
Update main display.

update_preview
Update preview of the reformated image.

15.10.6 Fusion unit

The purpose of the Fusion unit is to do manual fusion of two image stacks, one anatomical and one functional.

Interactions

Interactions are negligible.

Datastructure

The image information on which to operate is taken from the `SET` struct and the stack number is taken from `NO` variable.

Functions

anatomiccolor_Callback
```
change colormap in the anatomical image stack.
```

map = calcintensitymapping(contrast,brightness)
```
calculating the colormap based on the input contrast and brightness.
```

calcrotationmatrix
```
update the functional and fusion images after flip, rotation or translation.
```

close_Callback
```
close the Fusion GUI.
```

contrast_Callback(movement,imagestack)
```
update the contrast and brightness in the anatomical or funcitonal image stack.
```

doflip(flipparameter)
```
flip the functional image.
```

done_Callback
```
add the Fusion image stack to the Segment main gui.
The Fusion image stack is built up by the functioal image and
the transformed segmentation from the anatomical image.
```

drawanatomic
```
draw the anatomical image stack.
```

drawfunctional
draw the funcitonal image stack.

drawfusion
draw the fusion image stack.

res = findtwo2(curve,point)
find the two closest points in the input variable curve, to the point in the input variable point.

flip_Callback(flipparameter)
flip the functional image and update the new images.

functionalcolor_Callback
change colormap in the functional image stack.

init_Callback
starts the fusion gui.

keypressed(fignum,evnt)
move in the image slices with the keyboard arrows.

move_Callback(button)
read in translation and rotation.

plotval = plotvalues(position,myocardium,zvalues,t)
find and return the endo- and epivalues to plot.

resamplefunction(param1,param2,param3)
resample the functional image to the same size as the anatomical image also rotate and translate the functional image.

reset_Callback
reset to the start position of the functional image.

resetcontrast_Callback
reset the contrast and brightness to the start values.

savedefault_Callback
save the current translation, rotation, flip and colormap in a default .mat-file.

plotval = segmentation(set1,set2,nr,parameter,t)
return the endocardial segmentation in the anatomical and functional image stacks.

setdefault_Callback
apply the default .mat-file (including settings for translation, rotation, flip and colormap) to the current functional image stack.

settransparent(transparentparameter)
set the intensity balance between the anatomical and functional images in the fusi

transparent_Callback(movement)
update the intensity balance between the anatomical and functional images in the f

undo_Callback
undo the last translation or rotation.

update(imagestack)
update all image stacks after left mouse click in the anatomical or functional image stacks.

updateanatomic
update the anatomic image stack.

updatefunctional
update the functional image stack.

updatefusion
update the fusion image stack.

updatesegmentation
update the segmentation and lines in all images.

15.10.7 General segmentation unit

The purpose of the general segmentation tool is to be able to do semi-automatic segmentation of general objects and not only left ventricle and right ventricle. The unit is based on using a level set method and the user can change many inputs to the levelset to get a good segmentation.

Interactions

Interactions are negligible.

Datastructure

Relevant data for general segmentation is stored in the field `.LevelSet` in both the `SET` variable and the `DATA` variable.

The most important subfield of `SET.LevelSet` are:

- `BW`: The resulting segmentation stored as 0 or 1 (black or white)
- `Man`: The manual segmentation
- `Object`: Stores information related to dividing the segmentation into seperate objects
- `Speed`: Stores information related to the speed image used in the level set segmentation
- `Segmentation`: Stores information related to the parameters in the level set segmentation
- `View`: Stores information related to the graphical viewing of the segmentation
- `Pen`: Stores information related to the pen used for manual segmentation
- `Prototype`: Stores information related to prototype based segmentation using level set
- `RegionGrowing`: Stores information related to region growing segmentation using level set

The most important subfields of `DATA.LevelSet` are:

- `BackupBWInd`: Stores the indices of SET.LevelSet.BW from previous segmentation iteration to be able to go back to previous iteration
- `BackupManAddInd`: Stores the positive indices of SET.LevelSet.Man from previous segmentation iteration to be able to go back to previous iteration
- `BackupManRemoveInd`: Stores the negative indices of SET.LevelSet.Man from previous segmentation iteration to be able to go back to previous iteration
- `SpeedIM`: Stores the speed image calculated from SET.LevelSet.Speed
- `GradientPart`: Stores the gradient calculated from SET.IM

- `Prototype`: Stores information related to prototype based segmentation which does not need to be saved in SET.LevelSet.Prototype

Functions

all functions relevant for the unit is implemented in levelset.m.

16 Module Implementation

16.1 Bruker module

The documentation for the usage of the Bruker Module is given in Segment User Manual.

16.1.1 Interactions

The module interacts with the `DATA.Preview` field in the exact same way as the standard file loader does.

16.1.2 Datastructure

It uses the structure in `DATA.Preview` to store information and pass on to code to set up `SET` structure.

16.1.3 Functions

The documentation for the usage of the Bruker Module is given in Segment User Manual.

16.2 MIP module

MIP stands for maximum intensity projection. The MIP module is still a work in progress module.

16.2.1 Interactions

No interactions besides usage of `SET` and `NO` to retreive image data.

16.2.2 Datastructure

Will be documented when fully functional.

16.2.3 Functions

Will be documented when fully functional.

16.3 Corelab module

The purpose of the Corelab Module is to implement support for working with Segment in a Corelab setting.

16.3.1 Interactions

The interactions are performed only with Medviso AB corelab database (Transfer) and local file system for downloaded image files.

16.3.2 Datastructure

Not applicable.

16.3.3 Functions

For internal use only and no public documentation will be made available.

17 Testing Segment

To ensure highest quality of Segment as possible, a complete software testing system has been implemented. The test procedures include both system integration tests as well as unit implementation tests. The automated testing system produces a comprehensive test report (`testreport.tex`) including found bugs in Segment. A summary for each of all tests that have been performed for Segment is stored in a test record file (`testrecord.tex`).

17.1 Testing functionality

The test functionality is implemented in the file `test.m`. It outputs the details of the testing result to the files `testresultdoc.tex`, `testrecordtable.tex` and `testsummarydoc.tex` that are used when the test report and test record are generated. For an exact list of the included tests, see the latest available version of the test report. For each test performed a detailed list of tested functions are produced.

17.2 How to write testcases in maketest.m

Implementing a testcase consist of two parts, writing the testfunction and adding the testfunction to the list of test to perform.

17.2.1 Writing test function

A testfunction is written as

```
function result=testfunction
result=[]; %commands to test
result=subtest(result,passcriteria,subtestdescription)
%more commands to test
result=subtest(result,another pass criteria, another subtestdescription)
```

In this function passcriteria is a string of tests that can be evaluated as either true or false by an eval in the function subtest. The subtestdescription is a string describing the commands tested.

17.2.2 Add test function to list of tests

To add test functions to the testlist it should be grouped either into one of the already available test case if clauses in the function dotestscript, for example added into the loaddicomfiles caluse, or a new test case clause should be added. In the test case if clause the test function is added to the testlist by the following commands

```
testlist = addtest(testlist,...shortdescirption,...
longertestdescritption,...  boolean,@testfunction);
```

Where short description and longer description describes the purpose of the testfunction, the boolena is either true or false, true if all image stacks should be closed before starting test oterwise false and testfunction is the name of the function to test. If a new test case is added it will have to be added in the function `testcaseselecto`r as well so that it is possible to select the test case when running maketest.

17.3 Testing broken callbacks

Testing broken callbacks is tested by the function `validatecallbacks`. Called with one input argument it tests the `.fig` file in the input argument and prints result in the standard output. If called with no input arguments, then the entire Segment project is tested and a report is written to the file `callbacksdoc.tex` which is used to produce the test report. The test script assumes that the first argument in a callback string is a sub function in the function name in the callback string (if applicable). The script tests `uimenu`, `uicontextmenu`, and `uicontrol` objects that has a callback string defined. To parse out sub functions of a function the function `document.m` is used.

18 Compiling Segment

18.1 Introduction

Compiling Segment is a complex process since there are many depending files that need to be included. Therefore, a make script `makeit.m` was created. This script contains a list of all files required in the entire Segment project. With the addition of new Segment products, the script was refactored into an object oriented approach using a hierarchy of compilation script classes. This class hierarchy is described in Chapter 11. When making new files it is therefore necessary to include the new files in the compilation class definition files for softwares in which the file is to be used. 'Compilation' is in this chapter used in its widest sense, including not only making a executable standalone application, but also the creation of a .zip file for distributing Segment in a source code format, or generating all documentation files. A major part of the script is the control of which files should be included and to ensure that the files are properly protected when distributing as a source code format.

18.2 Running the compilation script

This section is generic for all softwares by Medviso.

The first thing one need to do before compiling Segment is to update the file `changelog.m` with the current version number (i.e release number). This information is used when creating the output file. The script can run in four modes and upon running `internaltools.makeit.m` a GUI prompts the user to select one.

- Make Segment standalone, make a standalone research edition executable suitable for installation.
- Make Segment CMR standalone, make a standalone clinical MR edition executable.
- Make Segment CT standalone, make a standalone clinical CT edition executable.
- Make CVQ standalone, custom layout of Segment executable.

- Make RVQ standalone, software adapted for kidney studies executable.
- Make Matlab stubbed modules, create a .zip file of source code for release to the public.
- Make Matlab all modules but protected, create a .zip file of source code for trusted users only.
- Make Segment standalone with 4D-flow, make a standalone of Segment containing the 4D-flow module, for trusted users only.

Note: You need to have Winzip self extractor installed on your computer to be able to run the standalone option. The GUI also enables the user to select whether to run the test script before compiling, making documentation and uploading the final compiled file to FTP.

It is necessary also to check that there are no critical tickets before compiling. In some cases the combination of testing and compilation in one session fails. To remedy this problem the function `internaltools.slowcompile` was introduced. Call this function with either `segment` or `segmentcmr`.

18.3 Configuring

Settings in the files are done in the section entitled
%————————
%%% Settings %%%
%————————
Below you find a list of items that can be configured.

- Toolboxes used. You need to manually enter all required toolboxes. This is neccessary since the compiler script excludes are toolboxes but the specifically indicated toolboxes. This is done in order to minimize the size of the compiled project. Large projects will take long time to start since they need to be virus scanned before starting, and the size also affects the real-time unpacking. To help to locate the required toolboxes, use checktoolboxdepency.m.
- M-files in the project. You need to add normally used m-files here. It is not enough to only add it to the SVN tree, you need to add it to makeit.m. Failure to add a file to makeit.m will most probably not affect the standalone file, but it will affect when compiling to a source code format. See also under Section Modules below.

- Internal M-files. Here files that are only required for the maintenance of the entire software project are listed. The only reason to list them here is so that they will be included in the total project count when counting the total number of files, and lines in Segment project.
- Fig-files. Here you list all required .fig-files. Failure to include a file here will affect both the standalone version as well as the Matlab version.
- Mat-files. Here you list all required .mat-files. Failure to include a file here will affect both the standalone version as well as the Matlab version.
- Plugins. Here you list all plugins that should be available in the standalone version. You need only to add the main files (i.e files named plugin_xxx.m). Depending files is handled automatically since each plugin, should respond which depending files. For more information, see wiki:plugins.
- C-files. Here you list all required .c-files (i.e MEX-files). The files are assumed to be compiled for each target platform. See also makemex.m.
- Files to protect. Here you list all files that should be protected, and only be included in the source code format as precompiled p-code.
- Modules to include. Each time you add a new m-file to the Segment project you need to add it in makeit.m either in the sections above, or or as a module if the file is only used within a specific module. The difference between adding it as a general m-file or as a part of a module is readibility to enhance the understanding which m-file belongs to which modules. Generally modules should be the same as the modules listed in the file getmodule.m. Making a new module should be fairly straight-forward by copy and past in makeit.m.

18.4 Multi platform support

Currently Segment does not support multi platform except when running from Matlab. Therefore, compiling to standalone does only work for the Windows platform. Things to fix before the compilation is supported for other platforms are:

- Changes not to generate a self extracting executable.
- Create and upload new MCR Installer package for each target platform.

Note that there are also features that are not supported for other platforms than Windows even when running under Matlab, one such feature is the PACS connection and the Segment server.

18.5 Compiling documentation

A script to update the entire documentation base is included. Note that you need to have MikTex installed and properly configured prior to running this script. When run from the main compilation script, then the manuals are automatically uploaded to Medviso homepage. The user or process specifies which software to document and the script creates the following documents (provided that they are available) for that software:

- Instructions for use. The instructions for use are rebuilt with current data and version number taken from `changelog.m`. Depending on settings, the bibliography is also updated.
- Reference manual.
- Installation manual.
- Sectra PACS manual.
- Database manual.
- Technical manual. Function definitions and help texts are extracted from a set of m-files. Note that therefore it is very important that the coding standard is closely followed on how to make help texts about functions. Also all helper functions starting with the name my* are automatically documented.
- Test report. First validate callbacks are run to check if all callbacks are correct, then the last result of maketest are used to build a test report.
- Test record.
- Release record.

18.6 Sectra Plugin compilation

You need to compile the Sectra Plugin in Microsoft Visual Studio 2010 Professional. This version is bought by Medviso and disc are availble in main office.

To download the environment click in CSN (in Sectra web), see details in `wiki:SectraPlugin` for log in details. Click on Download, CAI SDK and take the latest release.

You start Microsoft Visual Studio by clicking on `sectraplugin.sln`. To compile, click on Build, Build solution. In the box, select 'Release' and not 'Debug'. Select platform Win32 or Win64. The the compilation is finished then the sectraplugin.dll is found in the `Release` folder. If you compiled 64 bit version then it is found in the `x64/Release` folder. Check in the version in the system as either `sectraplugin32.dll` or `sectraplugin64.dll`.

19 How to Create Own Plug-ins

The easiest method of how to learn to make own plug-ins is to study the example `plugin_template.m`. Writing own plug-ins is a great way of spreading your algorithms to users all over the world and also to contribute to the Segment project. For more details on how to contribute, or learn more about interacting with other users, please see Chapter 20.

A plugin file must have a name beginning with `plugin_*.m`. Note that the plugin may of course call other functions that can reside anywhere on the Matlab path. When Segment is started the current folder is scanned for functions with this pattern. Matching functions are called with the argument getname and a string with the name that should appear in Segment menu is expected.

Currently there are two other plugins that are shipped with the standard Segment edition:

- `plugin_imageloader.m`. Plugin to load non DICOM images into Segment. This plugin can load for instance `.jpg`, `.bmp`, `.tif`, `.png` files into Segment. This plugin gives some elementary details on the internal data structure.
- `plugin_calibrate.m`. Plugin to calibrate image resolution. This plugin gives some hints on using own GUI's and also some details about the internal data structure in Segment.
- `plugin_template.m`. Template plugin, simple template for creating plugins.
- `plugin_summarize.m`. Plugin to summarize results from multiple `.mat` files. This function is useful to read when creating own export scripts.
- `plugin_phaseflow.m`. Plugin to enable to make flow measurement when there is only phase images available.

For further documentation of the two first plug-ins, please see the Segment User Manual.

20 Segment User Community

As a result of the growing interest in Segment and as a response of numerous requests Medviso AB has started to form a user community web place. This initiative will be enlarged significantly as the members of the community both requests more and also expands the community. It is worth noting that in the user survey spring 2010, out of 169 answers 147 answered that they would follow the user community, and 45 answered that they would follow it often.

A preliminary start page of the user community can be found on the following Facebook page `http://www.facebook.com/pages/Segment/119840021370285`.

It is the aim to be able to provide the following activities on the user community pages:

1. Participate in discussion forums. Currently forums for Developers discussion and tips and tricks, Feature requests, Segment and Mac.

2. Contribute and share own plug-ins. This feature is currently not available. If you have plug-ins that you want to share, please email them to `support@medviso.com` and we will manually upload the plug-in. Currently writing own plug-ins to Segment is documented in the Segment Technical Manual.

3. FAQ sections. Currently we are gathering FAQ in our support program. All (or almost all) support request will be made available in a searchable data base. Exceptions on when support requests are not included when the user request so in conjunction with classified projects.

Staff from Medviso AB will follow the user community page closely and monitor any incoming questions or uprising discussions.

If you have any ideas or suggestions on how we should improve the user community, please send us an email to `support@medviso.com`.

Bibliography

[1] E. Heiberg, J. Sjogren, M. Ugander, M. Carlsson, H. Engblom, and H. Arheden. Design and validation of Segment–freely available software for cardiovascular image analysis. *BMC Med Imaging*, 10:1, 2010.

[2] E. Heiberg. *Automated Feature Detection in Multidimensional Images*. PhD thesis, 91-85297-10-0. Linkoping universitet, Department of Biomedical Engineering, 2004.

[3] E. Heiberg, L. Wigstrom, M. Carlsson, A. F. Bolger, and M. Karlsson. Time Resolved Three-dimensional Automated Segmentation of the Left Ventricle. In *IEEE Computers in Cardiology 2005*, volume 32, pages 599–602, Lyon, France, 2005.

[4] H. Engblom, M. B. Carlsson, E. Hedstrom, E. Heiberg, M. Ugander, G. S. Wagner, and H. Arheden. The endocardial extent of reperfused first-time myocardial infarction is more predictive of pathologic Q waves than is infarct transmurality: a magnetic resonance imaging study. *Clin Physiol Funct Imaging*, 27(2):101–8, 2007.

[5] S. Bidhult, G. Kantasis, A. H. Aletras, H. Arheden, E. Heiberg, and E. Hedstrom. Validation of T1 and T2 algorithms for quantitative MRI: performance by a vendor-independent software. *BMC Med Imaging*, 16(1):46, 2016. Bidhult, Sebastian Kantasis, George Aletras, Anthony H Arheden, Hakan Heiberg, Einar Hedstrom, Erik England BMC medical imaging BMC Med Imaging. 2016 Aug 8;16(1):46. doi: 10.1186/s12880-016-0148-6.

[6] S. Bidhult, C. G. Xanthis, L. L. Liljekvist, G. Greil, E. Nagel, A. H. Aletras, E. Heiberg, and E. Hedstrom. Validation of a new t2* algorithm

and its uncertainty value for cardiac and liver iron load determination from MRI magnitude images. *Magn Reson Med*, 75(4):1717–29, 2016.

[7] K. Dorniak, E. Heiberg, M. Hellmann, D. Rawicz-Zegrzda, M. Wesierska, R. Galaska, A. Sabisz, E. Szurowska, M. Dudziak, and E. Hedstrom. Required temporal resolution for accurate thoracic aortic pulse wave velocity measurements by phase-contrast magnetic resonance imaging and comparison with clinical standard applanation tonometry. *BMC Cardiovasc Disord*, 16(1):110, 2016. Dorniak, Karolina Heiberg, Einar Hellmann, Marcin Rawicz-Zegrzda, Dorota Wesierska, Maria Galaska, Rafal Sabisz, Agnieszka Szurowska, Edyta Dudziak, Maria Hedstrom, Erik England BMC cardiovascular disorders BMC Cardiovasc Disord. 2016 May 26;16(1):110. doi: 10.1186/s12872-016-0292-5.

[8] J. Tufvesson, E. Hedstrom, K. Steding-Ehrenborg, M. Carlsson, H. Arheden, and E. Heiberg. Validation and development of a new automatic algorithm for time resolved segmentation of the left ventricle in magnetic resonance imaging. *BioMed Research International*, In press 2015.

[9] H. Soneson, J. F. Ubachs, M. Ugander, H. Arheden, and E. Heiberg. An Improved Method for Automatic Segmentation of the Left Ventricle in Myocardial Perfusion SPECT. *J Nucl Med*, 50(2):205–13, 2009.

[10] H. Soneson, F. Hedeer, C. Arevalo, M. Carlsson, H. Engblom, J. F. Ubachs, H. Arheden, and E. Heiberg. Development and validation of a new automatic algorithm for quantification of left ventricular volumes and function in gated myocardial perfusion SPECT using cardiac magnetic resonance as reference standard. *J Nucl Cardiol*, 18(5):874–85, 2011.

[11] E. Heiberg, H. Engblom, J. Engvall, E. Hedstrom, M. Ugander, and H. Arheden. Semi-automatic quantification of myocardial infarction from delayed contrast enhanced magnetic resonance imaging. *Scand Cardiovasc J*, 39(5):267–75, 2005.

[12] E. Heiberg, M. Ugander, H. Engblom, M. Gotberg, G. K. Olivecrona, D. Erlinge, and H. Arheden. Automated quantification of myocardial infarction from MR images by accounting for partial volume effects: animal, phantom, and human study. *Radiology*, 246(2):581–8, 2008.

[13] E. Heiberg, H. Engblom, M. Ugander, and H. Arheden. Automated Calculation of Infarct Transmurality. In *IEEE Computers in Cardiology*, pages 165–168, Durham, USA, 2007.