# Contents

## Code location

The python code for the app is located in src/main/python

.

# Todo

**Initializing the app**

The app uses fbs to build. To load external files such as icons, csv, text files when initialising the app, an ApplicationCOntext class is used. The ctx will have attributes @cached_property that point to the external files. This is passed to the main window which itself has the application context as an attribute. Everything needs to be stored under src/main/resources/base/[folder_name].

For txt and csv files, use [folder_name]=config

For images and icons, use [folder_name]=images

InspectXrays is the main class with the following main classes as attributes

1. xray_selection_menu: *class xray_selection_menu in MenuWidgets.py*
2. widget_area_menu: *class area_menu_widget in MenuWidgets.py*
3. image_widget: *class ImageHandler in ImagingWidgets.py*

**Icons:**

Buttons using icons are initialised using a dictionary of icons from the app context

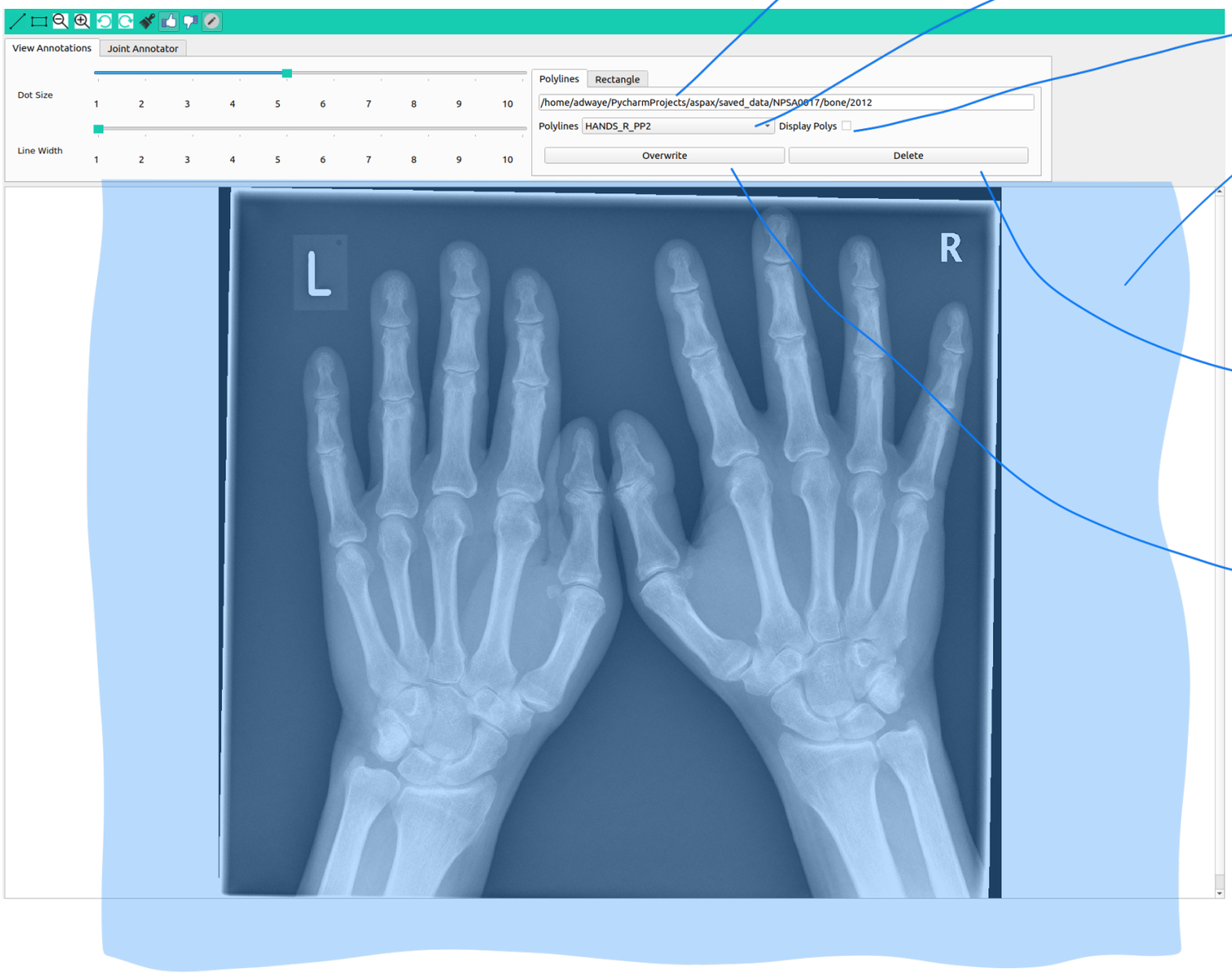
**App context**

Attributes:

1. main_window: the pyqt window to be launched by FBS
2. score_profiles: h5 files that contain the metadata used to load up the score sheet for each type of scoring method
3. image_handler_icons: dictionary of icons to be used in the image handling toolbar

## InspectXrays.image_widget

Class: ImageHandler




**AnnottationModelOptions.poly_loc_line_edit**

**ImageHandler.polyline_dropdown**

**ImageHandler.display_polylines_box**

ImageHandler.image_view:class MyView<-QGraphicsView

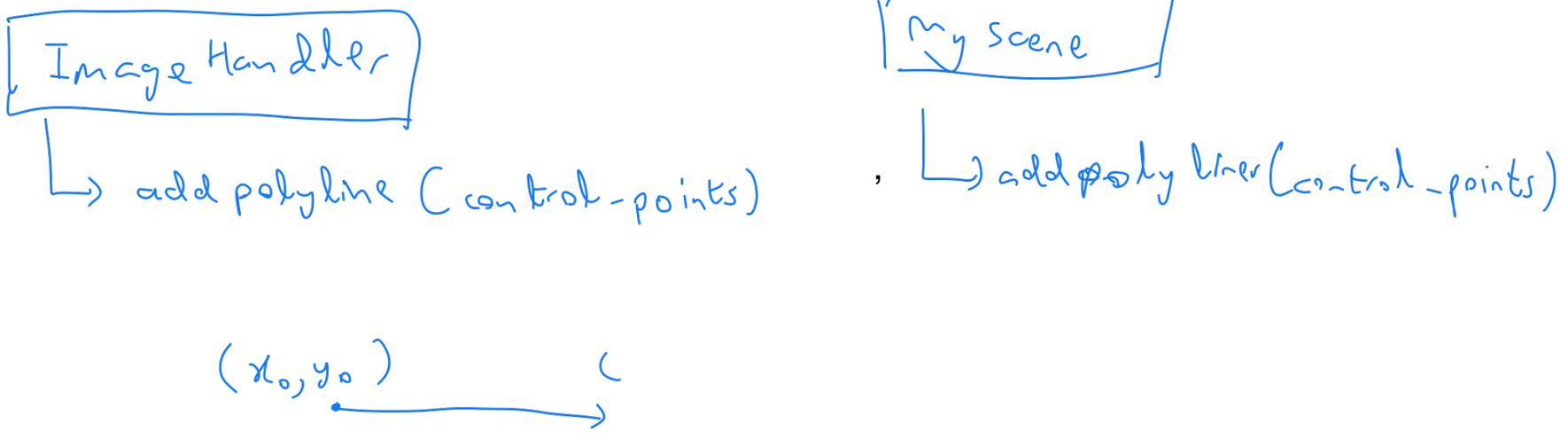
It has corresponding scene
ImageHandler.image_scene: class MyScene<-QGraphicsScene

ImageHandler.image_scene deals with the polylineItem drawing

**ImageHandler.delete_poly_button**

**ImageHandler.overwrite_poly_button**

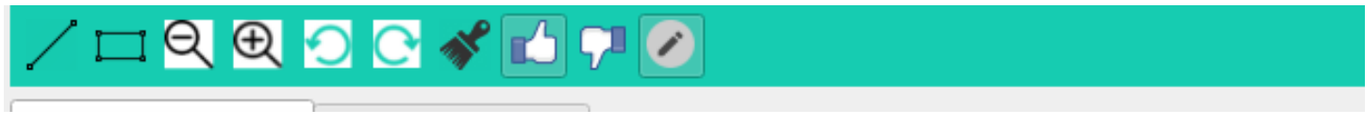
Create method in imaging-widget that creates a polyline Tran control pointe and adds it to the scene



ImageHandler.toolbar.buttons Is initialised by the ApplicationContext with passes an icon_lib to ImageHandler.toolbar

ImageHandler.toolbar.buttons is a dictionary with keys (left to right in diagram below)

1. ‘Draw Polyline’ :
2. ‘Draw Rectangle’ :
3. ‘Zoom Out’
4. ‘Zoom In’
5. ‘Undo’
6. ‘Clear Label’
7. ‘Good Image Quality’
8. ‘Bad Image Quality’
9. ‘Annotate’



# **ImageHandler.annotation_options:** class AnnotationModelOptions<-QWidgets

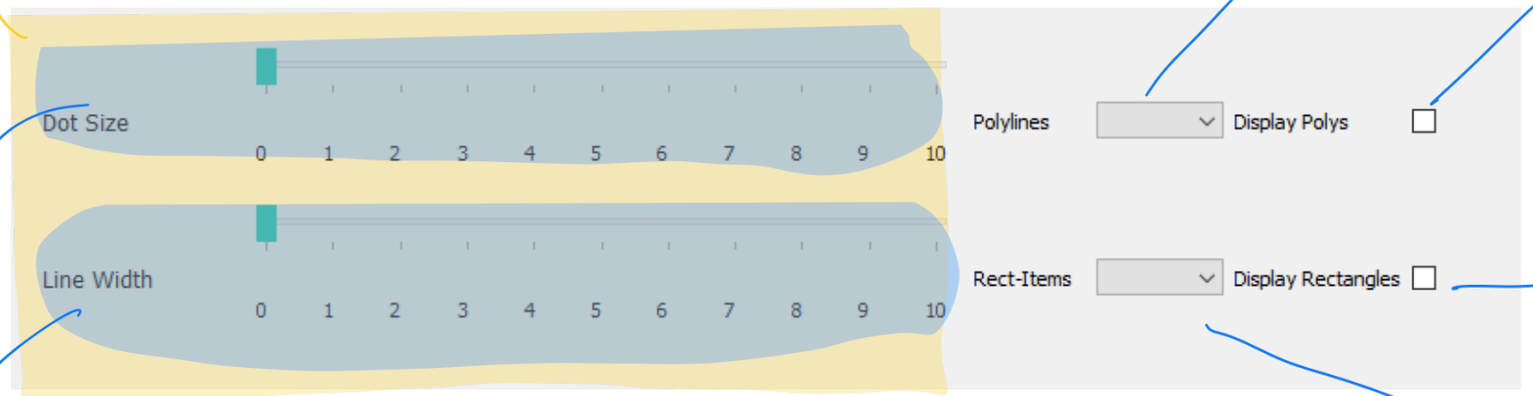
**AnnotationModelOptions.score_slider_layout**: class score_sliders<-QVBoxLayout

Initialising score_sliders requires a list of slider names and a list of values the slider should span.

The sliders are stored as a dictionary where the key is the title of the label rent to the slider and the value is the slider itself. AnnotationModelOptions.score_slider_layout.sliders or AnnotationModelOptions.score_sliders

Selected Methods:

1. get_slider_value: calls score_sliders.get_slider_values() returns a dictionary with key slider name and value the value of the slider




**AnnotationModelOptions.polyline_dropdown:** when a new xray is loaded by selecting a new xray from combobox_xrayid, bone annotations names corresponding to the year, xrayid and anatomical object is loaded: calls self.show_selected_annotation_bone which calls self.image_widget.image_scene.add_polyline is display_polylines_box is checked

**AnnotationModelOptions.display_polylines_box:** if ticked, then selecting the

**AnnotationModelOptions.display_rectItem_box**

**AnnotationModelOptions.score_sliders['Dot Size']**

**AnnotationModelOptions.score_sliders['Line Width']**

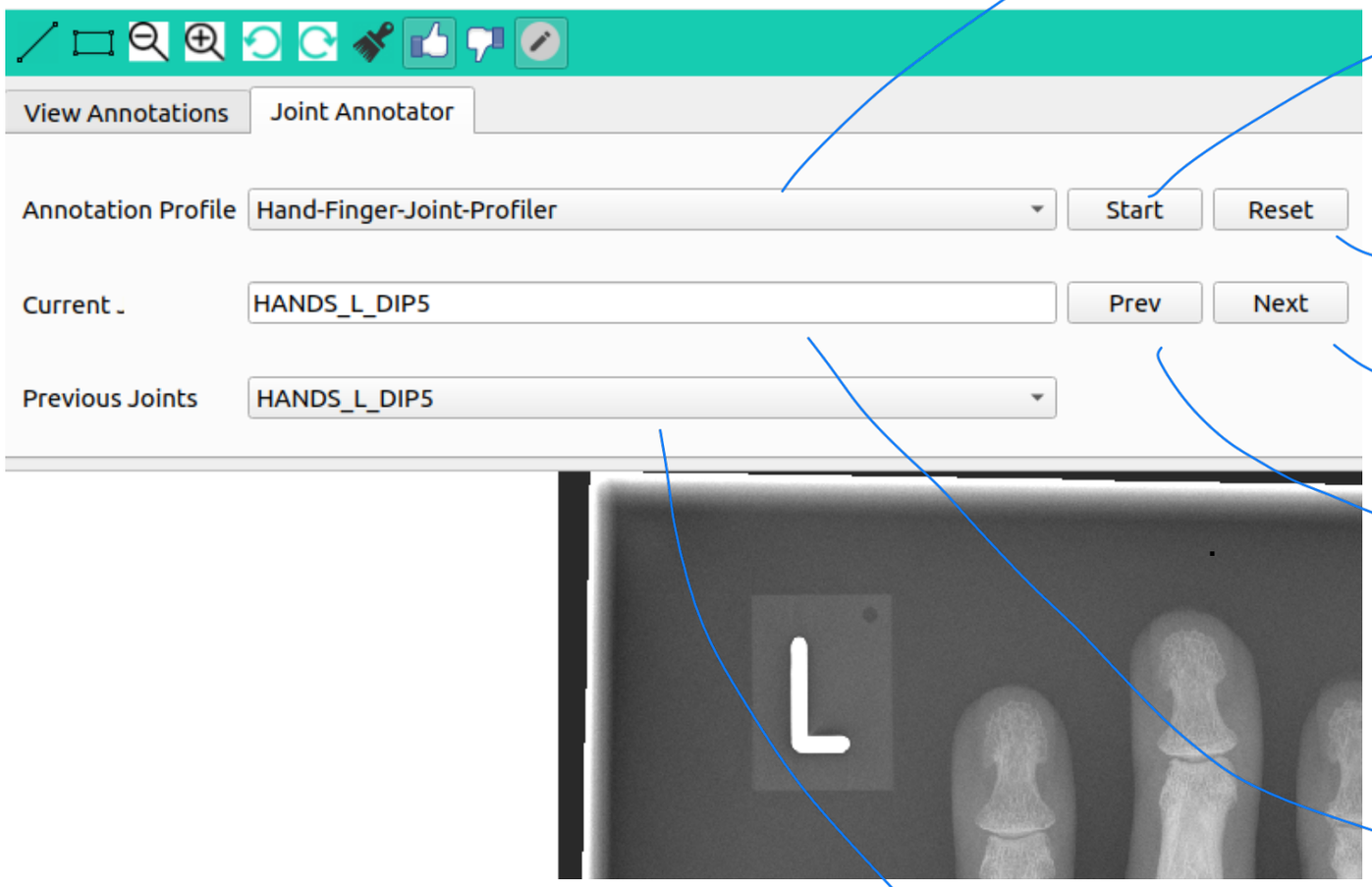
**AnnotationModelOptions.rectItem_dropbown:** when a new xray is loaded by selecting a new xray from combobox_xrayid, bone annotations names corresponding to the year, xrayid and anatomical object is loaded: calls self.show_selected_annotation_joint which calls self.image_widget.image_scene.add_rectItem is display_rectItem_box is checked

Self.imaging_widget.toolbar_buttons[‘Annotate’] is connected in InspectXrays to InspectXrays.trigger_annotation_mode which calls IMageHandler.trigger_annotation_mode and passes InspectXrays.output_loc to ImageHandler.trigger_annotation_mode. Hence one can pass the output)loc to the profiler.

When self.imaging_widget.toolbar_buttons[‘Annotate’] is called, profilers are initialised and added to the dropdown. Clicking the dropdown triggers the profiler annotate event where one annotates in turn: this is also when the next and previous button is connected

**ImageHandler.joint_annotator_panel:** Class JointAnnotatorPanel

Created by clicking on self.toolbar.button['Annotate'] which calls ImageHandler.trigger_annotation_mode which initialises the output_loc of JointAnnotatorPanel, and adds it to a tab; links JointAnnotatorPanel.start_button to ImageHandler.start_annotation; links JointAnnotatorPanel.reset_button to ImageHandler.reset_annotation; also initialises an item with key being the current text of the profile_dropdown, the item is HandJointAnnotationProfiler




**JointAnnotatorPanel.profile_dopdown**

**JointAnnotatorPanel.start_button**

**JointAnnotatorPanel.reset_button**

**JointAnnotatorPanel.next_button**

**JointAnnotatorPanel.previous_button**

**JointAnnotatorPanel.joint_name_qline**

**JointAnnotatorPanel.joint_dropdown**

Click start → connect next button

Click next button ⇒ add a rectItem. (calls imagescene method)

→ creates a dictionary of label coord pair e.g

my_dict['HANDS_L_DIP3'] = [x, y, w, h] or [

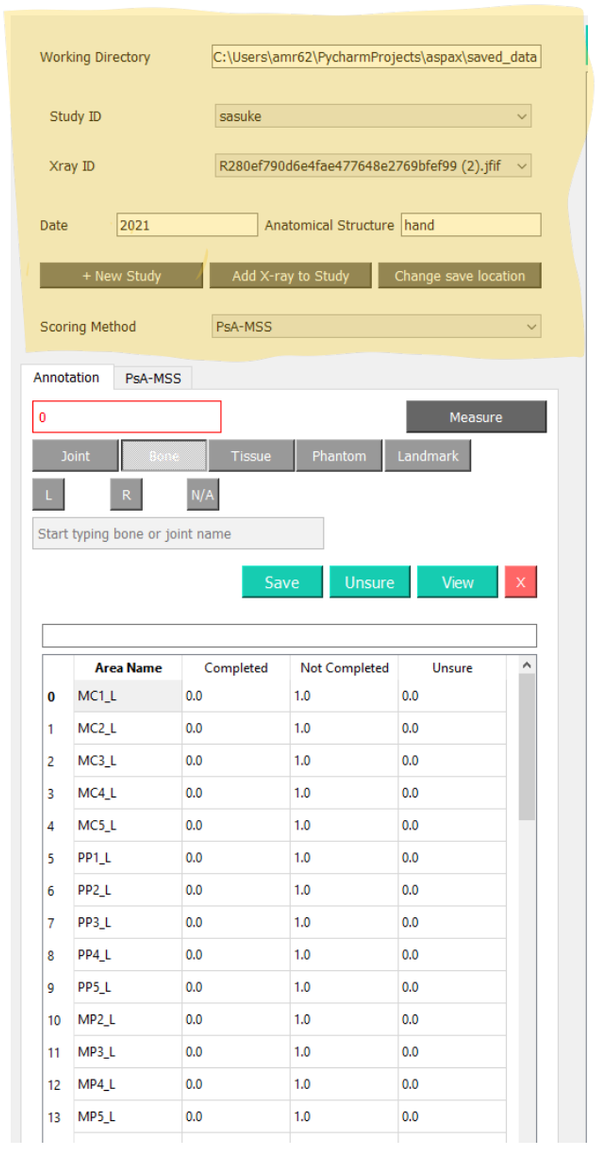
↳ save this as.

$$\begin{bmatrix} x+w & y-h \\ x-w & y-h \\ x-w & y+h \\ x+w & y+h \end{bmatrix}$$

→ when reached end of label file, asks user to save all annotations.

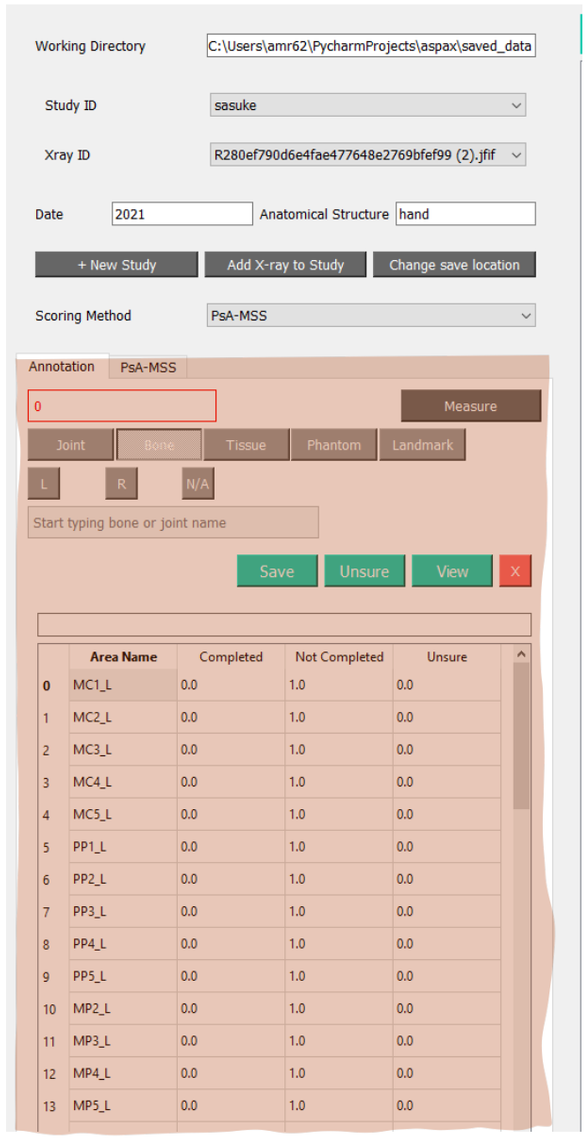
## InspectXrays.xray_selection_menu

Class: xray_selection_menu

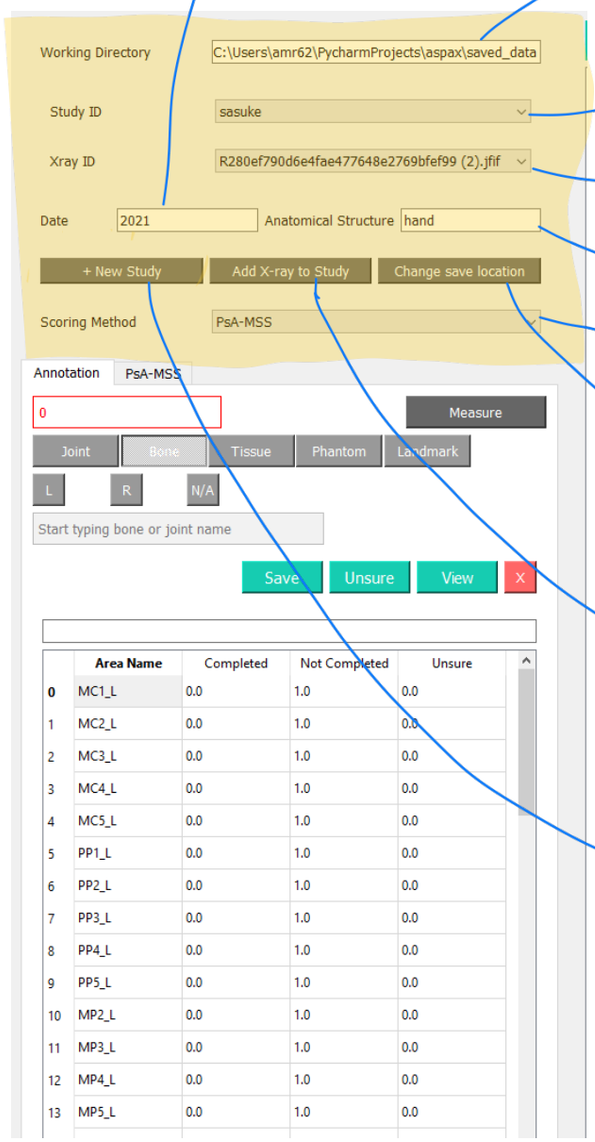



## InspectXrays.widget_area_menu

Class: area_menu_widget



# **InspectXrays.xray_selection_menu** class xray_selection_menu




xray_info_box_date: *QLineEdit*

wd_info *QlineEdit: displays the project working directory*

combobox_studyid *QComboBox: Drop down to select a different study. Triggers InspectXrays.display_xrays*

combobox_xrayid *QComboBox: Drop down to select a different xray from the same study. Triggers InspectXrays.load_selected_xrays*

xray_info_box_organ: *QLineEdit*

score_selector: *QLineEdit*

set_wdir_button *QPushButton: change the location where xrays/annotations are stored. If location contains a project filestructure, will load the studies it finds. Triggers InspectXrays.open_output_folder_selector*

addXrayToStudy_button *QPushButton: opens up the file dialog to allow the user to select a new xray to add to the current study. Triggers InspectXrays.open_xray_adder*

new_study_button *QPushButton: opens up the file dialog to allow the user to create a new study by selecting a new image xray file. Triggers InspectXrays.open_study_creator*

**Xray_selection_menu methods**

1. **getFiles:** launches a file dialog that allows one to select a file from the system returns tuple (str path/to/filename, str filetype) also sets xray_selection_menu.temp_name to be path/to/filename

2. **getDirectory:** launches a file dialog that allows one to select a folder from the system returns str path/to/foldername also sets xray_selection_menu.temp_name to be path/to/foldername

3. **create_new_study:** calls getFiles and sets the text in xray_selection_menu.current_study_info to be the full path to the selected image file used to initialise the study:- linked to xray_selection_menu.new_study_button.clicked

4. **add_xray_to_study:** calls getFiles and sets the text in xray_selection_menu.current_file_info to be the full path to the selected image file used to initialise the study:-linked to xray_selection_menu.addXrayToStudy_button.clicked

5. **change_wd:** calls getDirectory and sets the text in xray_selection_menu.wd_info to be the full path to the selected folder used to be the output directory:-linked to xray_selection_menu.set_wdir_button.clicked

**InspectXrays methods**

**Image Data handling methods**

1. open_output_folder_selector: deprecated

2. change_wd. : reads the text in self.xray_selection_menu.wd_info and sets self.output_loc to be this, clears xray_selection_menu.combobox_xrayid and xray_selection_menu.combobox_studyid and calls self.display_studies:- linked to self.xray_selection_menu

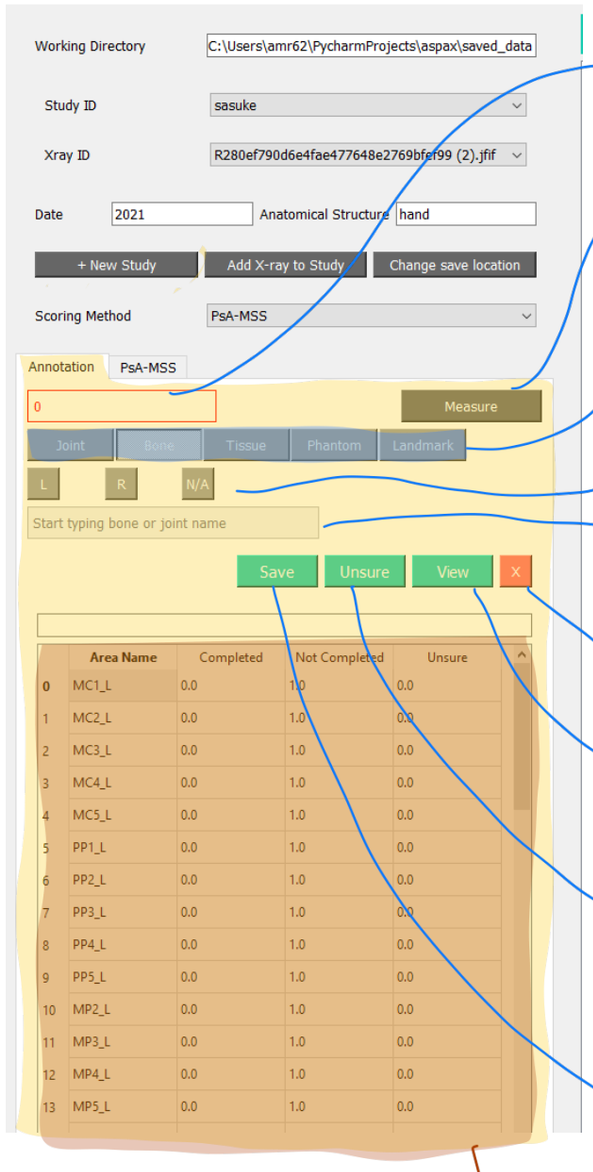
3. open_study_creator : loads the image file corresponding to self.xray_selection_menu .temp_name by calling self.image_widget.load_image, sets the quality to be good by pushing the like button: creates a XrayDataCreationDialog which asks the user if the image needs to have a scoresheet……

4. open_xray_adder. : loads the image file corresponding to self.xray_selection_menu .temp_name by calling self.image_widget.load_image, sets the quality to be good by pushing the like button: creates a XrayDataCreationDialog which asks the user if the image needs to have a scoresheet…….

5. display_xrays. : checks if there are more than 0 entries in combobox_studyid, reads the current index in the combobox_studyid and loads the metafile self.output_loc/studyname/metdadata.csv, pullds out the list of filenames under metadata[‘file_name’] and fills combobox_xrayid with these filenames. Then sets self.xray_record to be an XrayData class initialised with the meta_loc, then calls self.load_selected_xrays

6. load_selected_xrays. : Loads up the xray whose filename is currently selected in the combobox_xrayidand displays it by calling self.image_widget.load_image. Also sets the the image quality value if this is present in the metadata file

# **InspectXrays.widget_area_menu** *area_menu_widget<-distance_menu_widget*




output_box *QLineEdit: .*

output_button *QPushButton:*

annotation_button_group *QButtonGroup: auto-exclusive button group. Selected one is used to select save folder for the annotation for the annotation being saved.* ***Bone****-saves under bone, triggers XrayData.save_bone_outline;* ***Tissue****-saves under tissue, triggers XrayData.save_tissue_outline;****Joint****-saves under joint, triggers XrayData.save_tissue_outline;* ***Landmark****-saves under landmark, triggers XrayData.save_landmark;* ***Phantom****-saves under bone; triggers XrayData.save_bone_outline.*

annotation_side_group *QButtonGroup: auto-exclusive button group. Selected one is used to create a file name for the annotation being saved.*

line_edit_labels *QLineEdit: allows the user to type the name of the bone being annotated. Suggests name from self.bones_joints_list by using a completer*

discard_button *QPushButton: dummy button: need to find a use for it*

view_button *QPushButton->SaveButton: Opens a dialog that displays all the annotations drawn so far. Triggers InspectXrays.display_all_current_annotations*

unsure_button *QPushButton->SaveButton: click to mark the annotation as an unsure one that the user comes back to later. Appears in the tableView as a colored tile next to the annotation being looked at. Triggers InspectXrays.update_tracking_annotation*

save_button *QPushButton->SaveButton: Saves the latest annotation drawn. Triggers InspectXrays.save_annotation*

tableView *QTableView: displays the progress with annotating bones. Triggers InspectXrays.update_tracking_annotation*

# Annotation and Score tracking

This is handled by self.update_tracking_annotation.

1. Passes [bone_name]_[side_name] to self.widget_area_menu.update_table_view along with the text on the button pressed
2. Saves the new data in the table view under self.output_loc/xray_id/annotation_tracking_[date].csv

# Data Structures inside the app

## Xray-handling

Xrays are handled as a XrayData class found in MenuWidgets.py. Mostly handles saving annotations to XrayData.save_loc, which is set to be the InspectXrays.output_loc. Can be initialised with an image_name,xray_id,acquisition_date or with a meta_loc

Selected methods:

1. save_bone_outline(bone_id,date,plineitem): creates a folder name self.save_loc/bones/[date] and saves the plineitem in this folder as [bone_id].txt
2. save_phantom_outline(bone_id,date,plineitem): creates a folder name self.save_loc/bones/[date] and saves the plineitem in this folder as [bone_id].txt
3. save_tissue_outline(bone_id,date,plineitem): creates a folder name self.save_loc/tissue/[date] and saves the plineitem in this folder as [bone_id].txt
4. save_patch(joint_id,date,plineitem): creates a folder name self.save_loc/joint/[date] and saves the plineitem in this folder as [bone_id].txt
5. save_landmark(landmark_id,date,plineitem): creates a folder name self.save_loc/landmark/[date] and saves the plineitem in this folder as [bone_id].txt

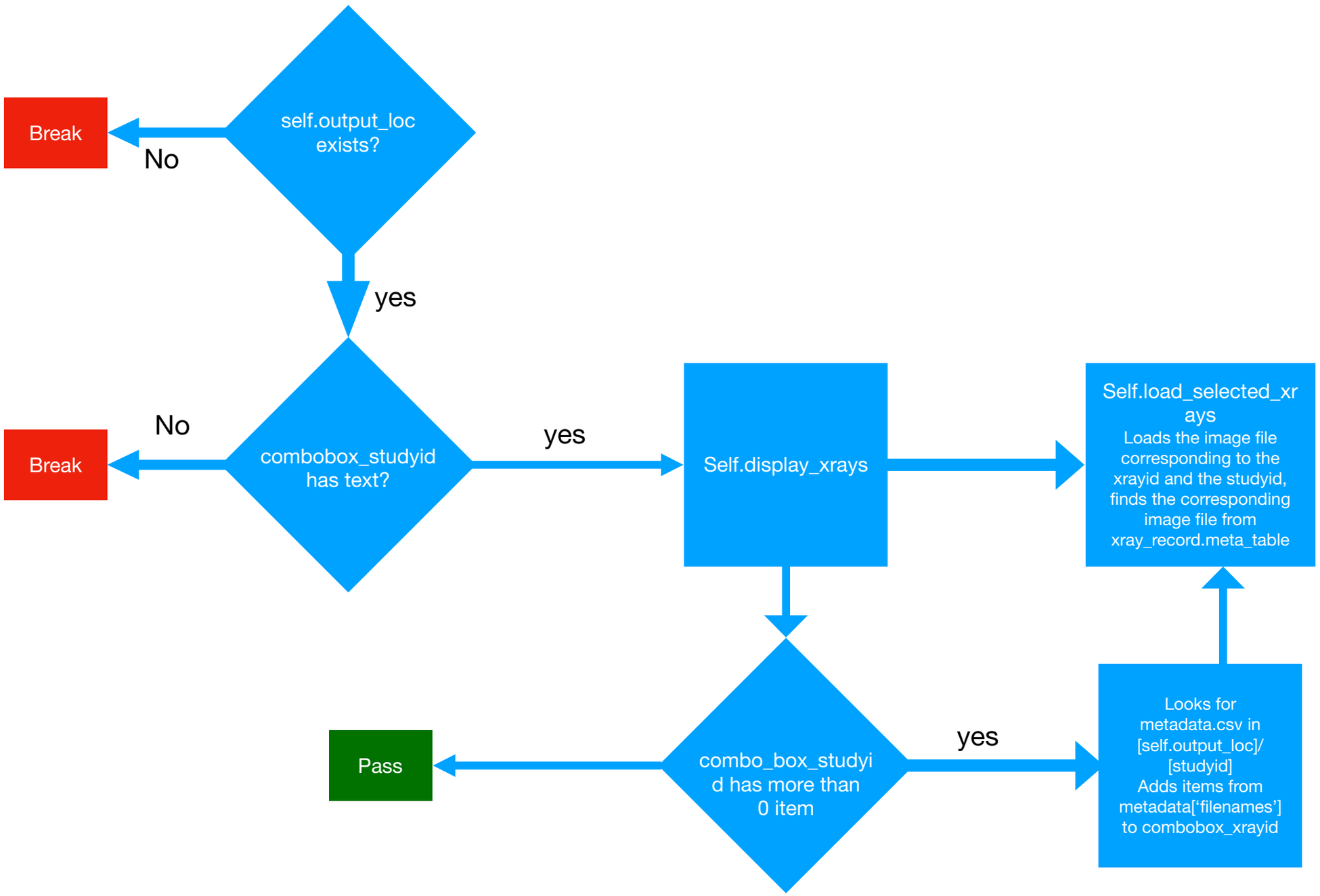
Selected Attributes

1. meta_table: dictionary with keys: [‘acquisition_date’, ‘xray_id’, ‘organ’, ‘file_name’, ‘image_quality’]

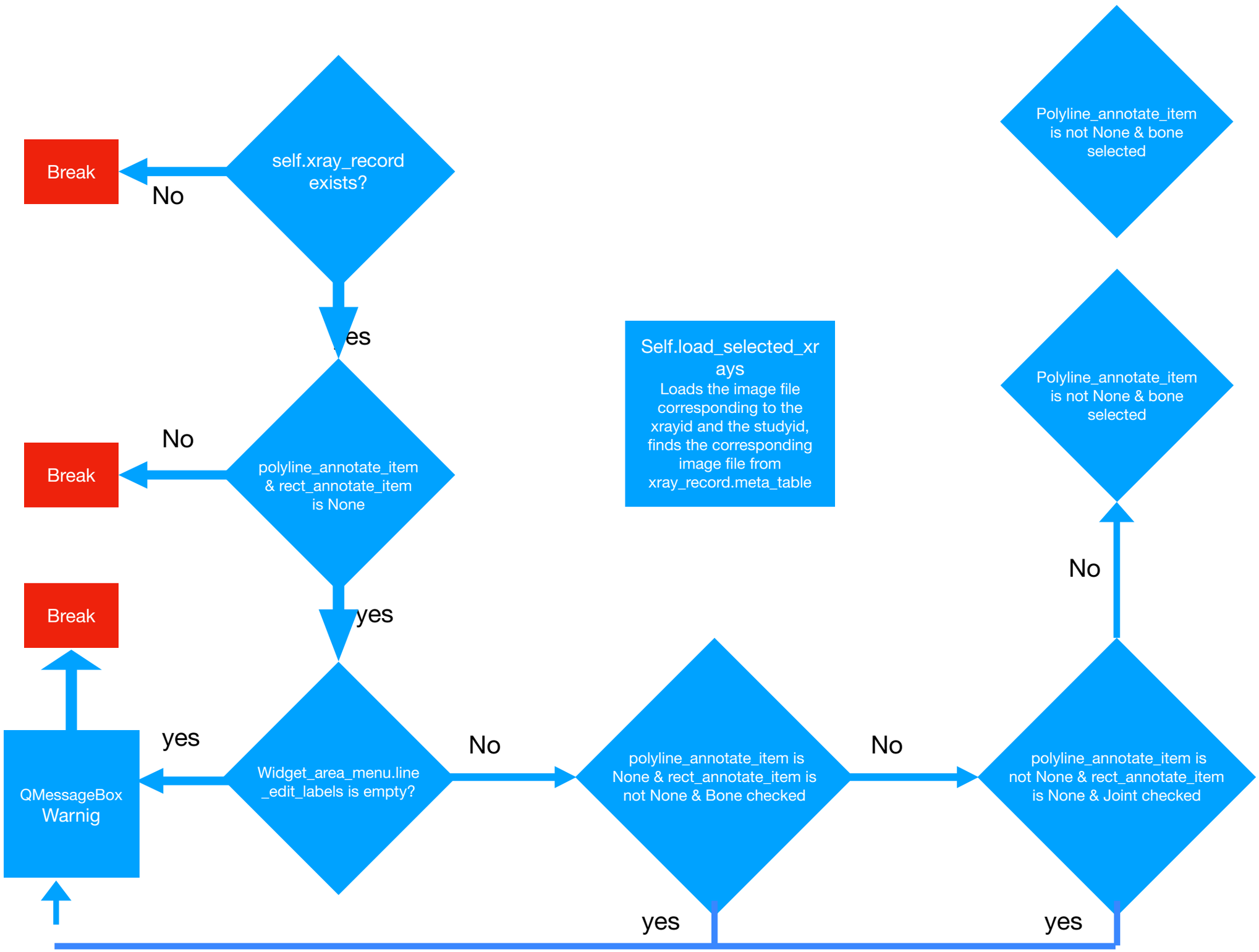
**Initializing the App**

Xrays are added by the initialise_xray_record method

1. Calls self.connect_sub_buttons which then calls self.display_studies, which looks up all folder names in self.output_loc and adds these to combobox_studyid, if self.output_loc is not present, then it breaks out
2. InspectXrays.output_loc is not None:


Break
No
self.output_loc exists?
yes
Break
No
combobox_studyid has text?
yes
Self.display_xrays
Self.load_selected_xrays
Loads the image file corresponding to the xrayid and the studyid, finds the corresponding image file from xray_record.meta_table
Pass
combo_box_studyid has more than 0 item
yes
Looks for metadata.csv in [self.output_loc]/[studyid]
Adds items from metadata['filenames'] to combobox_xrayid

## InspectXrays.save_annotation



**External Files**

1. **distance_menu_widget.bones_joint_list:** initialised when creating the widget. Uses the file is stored in src/main/resources/base/config/joint_list.txt

2. **xray_record.meta_table:** table containing details for all the xrays corresponding to the same study

polyline_annotate_item and bone selected : calls XrayData.save_bone_outline, target is *study_id/bone/date*

polyline_annotate_item and tissue selected: calls XrayData.save_tissue_outline, target is *study_id/tissue/date*

polyline_annotate_item and Phantom selected: calls XrayData.save_phantom_outline target is *study_id/bone/date* and file name is Phantom

Rect_annotate_item and Joint selected: calls XrayData.save_patch target is *study_id/joint/date* and file name is Phantom

polyline_annotate_item and landmark selected: vcalls XrayData.save_landmark target is *study_id/landmark/date* and file name is Phantom

**InspectXrays methods**

**Image Data handling methods**

1. open_output_folder_selector: launches
2. open_study_creator :
3. open_xray_adder. :
4. display_xrays. :
5. load_selected_xrays. :

ImagingWidgehandles initializing like and dislike buttons need to reinitialise like and dislike each time an image is loaded.
**Images are reloaded at the following actions**

1. When creating a new study done
2. when adding a new xray to the current study done
3. When changing xray from the drop down, done

need to link this to metadata

└, changed the initialization of xRayData and XrayStudy to have a metadata column describing image qualify

└, need to add a handler for previous metadata.csv files that do not have an image_quality column

Changes to make:

1. Add a method to test is the metadata has an image_quality column: if not, then create one with default values of 1
2. Create a method for MainWindow that changes the image_quality status of an image in the metadata.csv files
3. Link the like, dislike button to the

Tests to run:

1. Load xrays from the aspax_small_studies to see if the handler adds the image_quality column
2. Load xrays once the image_quality handle have been chenged to see if the dislike button is set checked

**Flags:**

1. **MyScene.draw_poly_flag:** Bool, if true, causes mouse clicks to add points to a polyline object, exclusive with draw_rect_flag, switched by clicking on ImageHandler.toolbar.buttons[‘Draw Polyline’]
2. **MyScene.draw_rect_flag:** Bool, if true, causes mouse drag and release to draw a rectItem, exclusive with draw_rect_flag, switched by clicking on ImageHandler.toolbar.buttons[‘Draw Polyline’]

**Class huo_method:**

**-__call__: self.extract_hand_mask: 11-12.85 s**

**self.get_finger_midlines: 39.59 s for Gaussian filters.**

**self.extract_joint_features:** uses cv2.Sobel which is causing a bottleneck in memory:3.6-3.7  for each phalange, perhaps can run this once and reuse these features

**self.locate_joints:**

**Saving a score:**

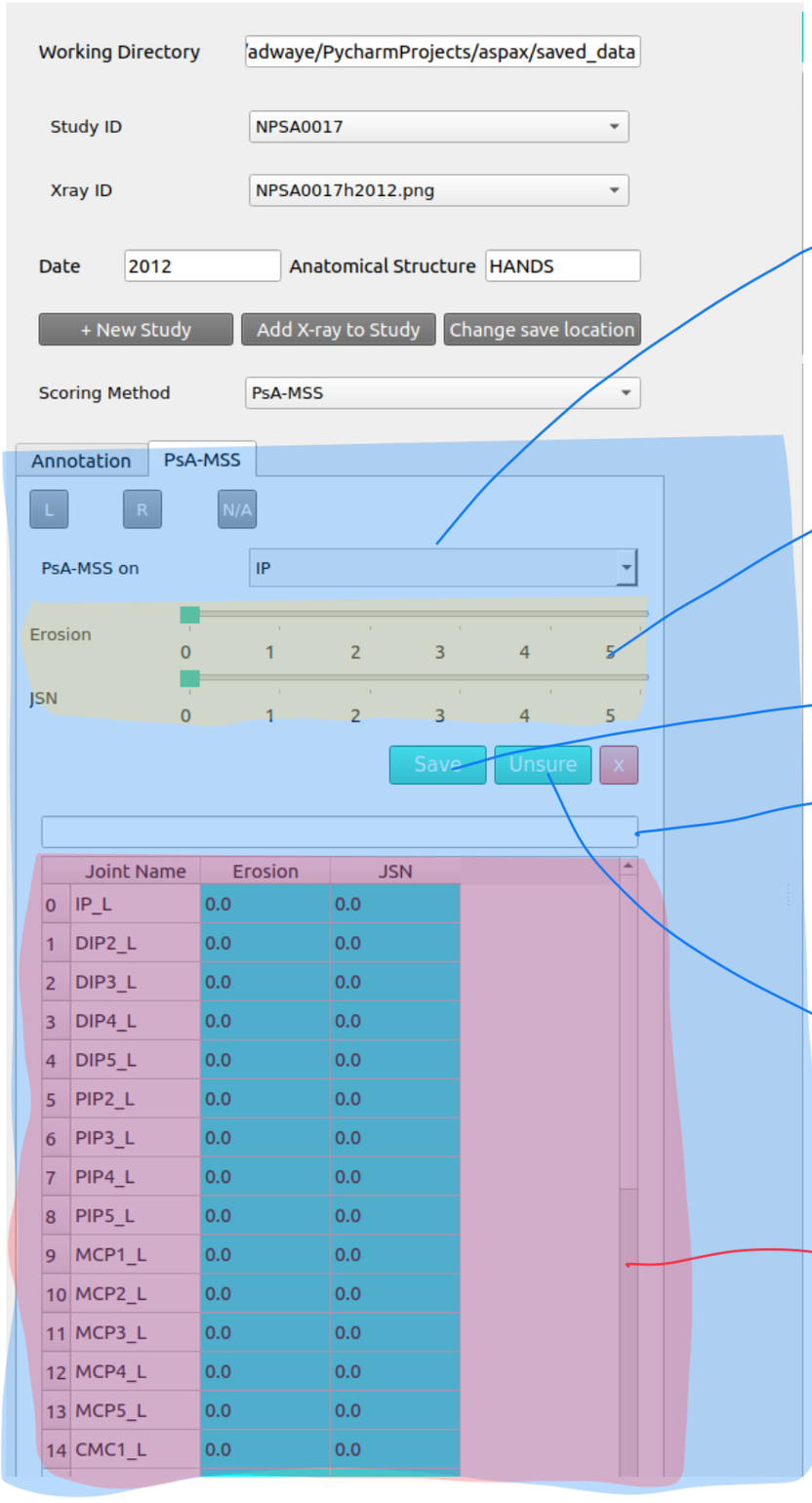
InspectXrays.widget_score_menu: class score_menu_widget: initialisation requires the passing of a profile, which is loaded from the resources of the application context.

Self.score_technique : name of the score technique, appears on the tab header

Self.damage_areas : list of joints that will be score under this scoring technique: used to populate the tableView

Self.damage_types. : list types of damage used to decide the number of sliders-one slider for each damage type

Self.damage_ranges: Tuple length N with list contain min and max score values for each damage type




**Self.score_area_box:** dropdown that contains self.damage_areas

**Self.score_sliders:** dictionary with keys item pair [damage_type,Slider()-class QSlider]

**Self.save_button:**

**Self.tableView_lineEdit:**

**Self.unsure_button:**

**Self.tableView:** initilised using self.create_table_view

-

## Loading a new score sheet

Click on xray_selection_menu.score_selector—> triggers InspectXrays.load_new_score_sheet