CSCI3130 grading helper

0.9

Generated by Doxygen 1.8.16

1 README

2 Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

3 Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

4 Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

5 File Index

5.1 File List

Here is a list of all files with brief descriptions:

6 Namespace Documentation

6.1 create_dates_diag Namespace Reference

Classes

- class Ui_Create_dates_dialog

6.2 dates_window Namespace Reference

Classes

- class Ui_dates_window

6.3 db_init Namespace Reference

Functions

- def settings_db_create (db_name=SETTINGS_DB_NAME, force=False)
- def settings_db_read_settings (db_name=SETTINGS_DB_NAME)
- def update_settings (paths, local, db_name=SETTINGS_DB_NAME)
- def grades_db_create (db_name, force=False)
- def load_student_list_into_grades_db (db_name, year, semester, filename='students_list3.txt')
- def insert_students (ids, fname, lname, db_name='./grades.sqlite3')
- def register_students_in_class (pipeline_ids, year, semester, db_name='./grades.sqlite3')
- def get_pipeline_ids (db_name='./grades.sqlite3')
- def get_ids_in_class_by_year_semester (year, semester, db_name='./grades.sqlite3')
- def import_previous_grades_into_db (year, semester, db_name='./grades.sqlite3', filename='./grades.xls')
- def gen_filenotfound_resp (lab_id, stud_path, corr_file, grader, att=None, next_date=None, db_name='./grades.↩ sqlite3')
- def get_resp_and_grade (grade_id, db_name='./grades.sqlite3')
- def get_prev_resp (grade_id, class_id, lab_id, db_name='./grades.sqlite3')
- def save_a_grade_to_db (grade_id, grade, grader_comment, extra_comment, grader_name, graded=True, pass_fail=True, db_name='./grades.sqlite3')
- def init_new_lab (stud_id, lab_name, att, submitted, lab_path, db_name='./grades.sqlite3')
- def get_lab_names (db_name='./grades.sqlite3')
- def update_lab_submissions_paths (db_name, repository_root, year, semester)
- def get_empty_grades_by_lid (lab_id, att, db_name='./grades.sqlite3')
- def get_all_grades_by_lid (lab_id, att, db_name='./grades.sqlite3')
- def reconstruct_grades_and_comments (db_name='./grades.sqlite3')

- def generate_final_grades (db_name, year, semester)
- def get_max_grade_for_lab (lid, year, semester, db_name='./grades.sqlite3')
- def get_grades_by_lab_and_att (lid, att, db_name='./grades.sqlite3')
- def get_lab_filename (lab_id, db_name='./grades.sqlite3')
- def get_lab_max_value (lab_id, db_name='./grades.sqlite3')
- def get_full_path (paths, local)
- def sync_files (self=None)
- def export_pdf (self=None)
- def save_grade_and_report (grade_id, grade, report, user_comment, grader, db_name='./grades.sqlite3')
- def commit_gen_report (grade_id, db_name='./grades.sqlite3')
- def get_lab_id (ltype, lab_num)
- def register_lab_in_semester (ltype, lab_num, year, semester, due_dates, db_name='./grades.sqlite3')
- def get_labid_in_schedule (lid, year, semester, db_name='./grades.sqlite3')
- def get_due_date_by_labid (lid_sem, att=None, db_name='./grades.sqlite3')
- def get_import_dates_by_labid (lid_sem, att=None, db_name='./grades.sqlite3')
- def gen_report (lid_sem, att=None, db_name='./grades.sqlite3')
- def get_pipids_in_class_by_year_semester (year, semester, db_name='./grades.sqlite3')

Variables

- string SETTINGS_DB_NAME = 'settings.sqlite3'

6.3.1 Function Documentation

6.3.1.1 commit_gen_report() def db_init.commit_gen_report (grade_id, db_name = './grades.sqlite3')

Definition at line 752 of file db_init.py.

```
00752 def commit_gen_report(grade_id, db_name='./grades.sqlite3'):
00753     if not os.path.isfile(db_name):
00754         raise Exception("DB not found")
00755     with lite.connect(db_name) as con:
00756         cur = con.cursor()
00757         cur.execute("UPDATE grades SET report_generated=strftime('%s','now') WHERE id=?", (grade_id,))
00758         con.commit()
00759
00760
00761
```

6.3.1.2 export_pdf() def db_init.export_pdf (self = None)

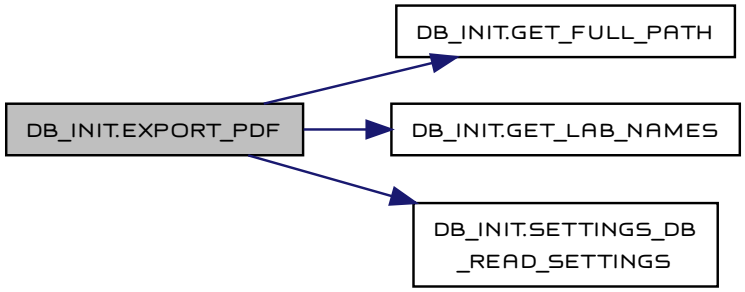
Definition at line 712 of file db_init.py.

```
00712 def export_pdf(self=None):
00713     import subprocess
00714     import os
00715 
00716     paths, local = settings_db_read_settings()
00717     lab_ids, lab_types, lab_nums = get_lab_names()
00718     lab_names = []
00719     for i in range(len(lab_types)):
00720         lab_names.append(lab_types[i] + '_Lab_' + str(lab_nums[i]))
00721 
00722     full_path = get_full_path(paths, local) + "/"
00723     for lab_name in lab_names:
00724         nums_to_sync = '_{'
00725         i = 1
00726         while os.path.isdir(full_path + lab_name + '_' + str(i) + '/Answers'):
00727             nums_to_sync += str(i) + ','
00728             i += 1
00729         if i == 1:
00730             continue
00731         nums_to_sync = nums_to_sync[0:-1] + '}'
00732         # for case when we have only one directory to sync
00733         if len(nums_to_sync) == 4:
00734             nums_to_sync = '_1'
00735         if len(nums_to_sync) > 1:
00736             command = local[4] + ' ' + full_path + lab_name + nums_to_sync + '/Answers/*.pdf ' + os.path.expanduser(paths[2]) + lab_name + '/'
00737             process = subprocess.Popen(os.path.expandvars(command), stdout=subprocess.PIPE, shell=True)
00738             process.communicate()
00739             # print(output)
00740             # print(error)
00741 
00742 
```

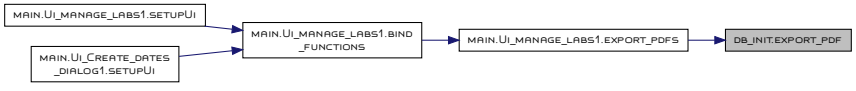
References get_full_path(), get_lab_names(), and settings_db_read_settings().

Referenced by main.Ui_manage_labs1.export_pdfs().

Here is the call graph for this function:



Here is the caller graph for this function:



6.3.1.3 gen_filenotfound_resp() def db_init.gen_filenotfound_resp (

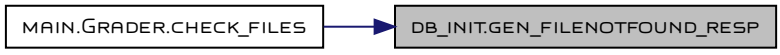
```
                lab_id,
                stud_path,
                corr_file,
                grader,
                att = None,
                next_date = None,
                db_name = './grades.sqlite3' )
```

Definition at line 411 of file db_init.py.

```
00411 def gen_filenotfound_resp(lab_id, stud_path, corr_file, grader, att=None, next_date=None, db_name='./grades.sqlite3'):
00412     resp_text = 'file with name "{}" was not found.</br>'.format(corr_file)
00413     file_found = os.listdir(stud_path)
00414     potential_files = list()
00415     for file in file_found:
00416         if file not in ['grade.txt', 'penalty.txt', 'responce.txt', 'tech_info.txt', ]:
00417             potential_files.append(file)
00418     if potential_files:
00419         resp_text += '\nNext files|folders were found:</br>\n'
00420     for file in potential_files:
00421         if os.path.isdir(os.path.join(stud_path, file)):
00422             resp_text += file + ' - directory.</br>\n'
00423         else:
00424             resp_text += file + ' - regular file.</br>\n'
00425 
00426     if att and att < 4 and next_date:
00427         resp_text += 'Please submit your file by next due date ({}).</br>\n'.format(next_date)
00428 
00429     if not os.path.isfile(db_name):
00430         raise Exception("DB not found")
00431     with lite.connect(db_name) as con:
00432         cur = con.cursor()
00433         cur.execute("UPDATE grades SET graded=strftime('%s','now'), pass_fail=FALSE, grader_comment=?, grader=? WHERE id=?", (resp_text,
      grader, lab_id))
00434         con.commit()
00435 
00436 
```

Referenced by main.Grader.check_files().

Here is the caller graph for this function:



6.3.1.4 gen_report() def db_init.gen_report (

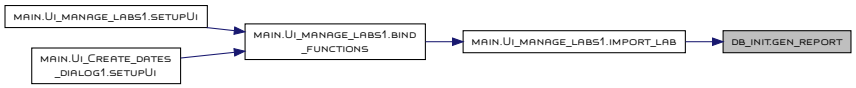
```
                lid_sem,
                att = None,
                db_name = './grades.sqlite3' )
```

Definition at line 813 of file db_init.py.

```
00813 def gen_report(lid_sem, att=None, db_name='./grades.sqlite3'):
00814     if not os.path.isfile(db_name):
00815         raise Exception("DB not found")
00816     with lite.connect(db_name) as con:
00817         cur = con.cursor()
00818         cur.execute("UPDATE lab_schedule SET imported_{}=strftime('%s','now') WHERE id=?".format(att), (lid_sem,))
00819         con.commit()
00820 
00821 
```

Referenced by main.Ui_manage_labs1.import_lab().

Here is the caller graph for this function:



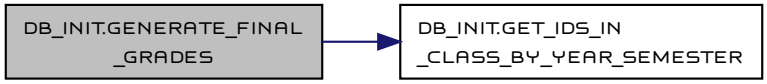
6.3.1.5 generate_final_grades() def db_init.generate_final_grades (db_name, year, semester)

Definition at line 598 of file db_init.py.

```
00598 def generate_final_grades(db_name, year, semester):
00599     ids = get_ids_in_class_by_year_semester(year, semester, db_name)
00600     with lite.connect(db_name) as con:
00601         cur = con.cursor()
00602 
00603         labs = list()
00604         for sid in ids.values():  # using JOIN here will add too much extra data
00605             result = cur.execute('SELECT lab, MAX(grade * (select percent from penalties where id=GRADES.attempt)/100) '
00606                                  'FROM GRADES WHERE class_id=? and attempt > 0 group by lab order by lab', (str(sid),))
00607             labs.append(result.fetchall() )
00608 
00609         stud_info = list()
00610         for sid in ids.keys():
00611             result = cur.execute('SELECT first_name, second_name FROM students WHERE pipeline_id=?', (str(sid),))
00612             stud_info.append(result.fetchall() )
00613 
00614     df_stud_info = pd.DataFrame(dict(zip(ids.keys(), stud_info)))
00615     df_grades = pd.DataFrame(dict(zip(ids.keys(), labs)))
00616     # id_list = list(ids.keys())
00617     # a = id_list[list(ids.values()).index(class_id)]
00618 
00619 
```

References get_ids_in_class_by_year_semester().

Here is the call graph for this function:



6.3.1.6 get_all_grades_by_lid() def db_init.get_all_grades_by_lid (

lab_id,

att,

db_name = './grades.sqlite3')

Definition at line 543 of file db_init.py.

```
00543 def get_all_grades_by_lid(lab_id, att, db_name='./grades.sqlite3'):
00544     with lite.connect(db_name) as con:
00545         cur = con.cursor()
00546         result = cur.execute("SELECT submitted, class_id, id, lab_path FROM grades WHERE lab=? AND attempt=? ", (lab_id, att))
00547         try:
00548             subm, class_id, lab_id, lab_path = zip(*result.fetchall())
00549         except Exception as e:
00550             print(e)
00551             return None, None
00552 
00553     return subm, class_id, lab_id, lab_path
00554 
00555 
```

6.3.1.7 get_due_date_by_labid() def db_init.get_due_date_by_labid (

lid_sem,

att = None,

db_name = './grades.sqlite3')

Definition at line 790 of file db_init.py.

```
00790 def get_due_date_by_labid(lid_sem, att=None, db_name='./grades.sqlite3'):
00791     with lite.connect(db_name) as con:
00792         cur = con.cursor()
00793         if att:
00794             result = cur.execute('SELECT due_date_{} FROM lab_schedule WHERE id=?'.format(int(att)), (lid_sem,))
00795         else:
00796             result = cur.execute('SELECT due_date_1, due_date_2, due_date_3, due_date_4 FROM lab_schedule WHERE id=?', (lid_sem,))
00797         return result.fetchone()
00798     return None
00799 
00800 
```

Referenced by main.get_grading_period().

Here is the caller graph for this function:

6.3.1.8 get_empty_grades_by_lid() def db_init.get_empty_grades_by_lid (

lab_id,

att,

db_name = './grades.sqlite3')

Definition at line 530 of file db_init.py.

```
00530 def get_empty_grades_by_lid(lab_id, att, db_name='./grades.sqlite3'):
```

```
00531     with lite.connect(db_name) as con:
00532         cur = con.cursor()
00533         result = cur.execute("SELECT submitted, class_id, id, lab_path FROM grades WHERE lab=? AND attempt=? AND graded is NULL", (lab_id,
      att))
00534         try:
00535             subm, class_id, lab_id, lab_path = zip(*result.fetchall())
00536         except Exception as e:
00537             # print(e)
00538             return None, None, None, None
00539 
00540     return subm, class_id, lab_id, lab_path
00541 
00542 
```

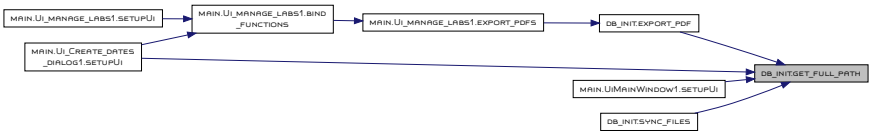
6.3.1.9 get_full_path() def db_init.get_full_path (paths, local)

Definition at line 675 of file db_init.py.

```
00675 def get_full_path(paths, local):
00676     import os
00677     return os.path.expanduser(paths[1]) + str(local[1]) + "_" + str(local[2])
00678 
00679 
```

Referenced by export_pdf(), main.UiMainWindow1.setupUi(), main.Ui_Create_dates_dialog1.setupUi(), and sync_files().

Here is the caller graph for this function:



6.3.1.10 get_grades_by_lab_and_att() def db_init.get_grades_by_lab_and_att (lid, att, db_name = './grades.sqlite3')

Definition at line 638 of file db_init.py.

```
00638 def get_grades_by_lab_and_att(lid, att, db_name='./grades.sqlite3'):
00639     with lite.connect(db_name, detect_types=lite.PARSE_COLNAMES) as con:
00640         cur = con.cursor()
00641         result = cur.execute('select a.due_date_{0} as due_date, a.imported_{0} as import_date, '
00642                              'b.type, b.num, b.max_grade, '
00643                              'c.id as grade_id, c.submitted, c.graded, c.grade, c.pass_fail, c.grader_comment, c.extra_comment, c.grader,
      c.lab_path, '
00644                              'd.pipeline_id, e.first_name, e.second_name, f.percent, c.grade*f.percent/100 as final_grade '
00645                              'from lab_schedule a '
00646                              'join lab_names b on a.lab_id=b.id '
00647                              'join grades c on c.lab=a.id '
00648                              'join class d on d.id=c.class_id '
00649                              'join students e on e.pipeline_id=d.pipeline_id '
```

```
00650                           'join penalties f on f.id=c.attempt '
00651                           'where c.attempt={0} AND a.id=? ORDER BY d.pipeline_id'.format(int(att)), (lid,))
00652         info_tup = result.fetchall()
00653         info_desc = result.description
00654     return info_tup, info_desc
00655
00656
```

Referenced by generate.generate_answers3().

Here is the caller graph for this function:

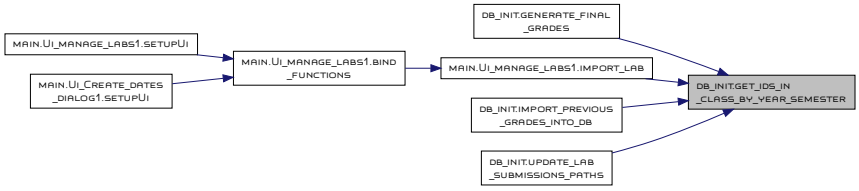
6.3.1.11 get_ids_in_class_by_year_semester() def db_init.get_ids_in_class_by_year_semester (year, semester, db_name = './grades.sqlite3')

Definition at line 312 of file db_init.py.

```
00312     """
00313     with lite.connect(db_name) as con:
00314         cur = con.cursor()
00315         result = cur.execute("SELECT pipeline_id, id FROM class\
00316                               WHERE year=" + str(year) + " and semester=" + str(semester))
00317         try:
00318             res = result.fetchall()
00319             pip_to_id = dict (res)
00320             to_id_to_pip = dict ([(res_id[1], res_id[0]) for res_id in res])
00321         except Exception as e:
00322             print(e)
00323             return None
00324     return pip_to_id, to_id_to_pip
00325
00326
00327 def import_previous_grades_into_db(year, semester, db_name='./grades.sqlite3', filename='./grades.xls'):
00328     """
00329     Takes xls file with grades from previous semester(s) and loads all grades into DB.
00330     In case students are not found in the DB and xls file contains ids - loads them too
00331      year: year when grades were assigned
00332      semester: semester when grades were assigned
00333      db_name: specific name of the grades DB
00334      filename: xls file to load
00335     :return: nothing
```

Referenced by generate_final_grades(), main.Ui_manage_labs1.import_lab(), import_previous_grades_into_db(), and update_lab_submissions_paths().

Here is the caller graph for this function:



6.3.1.12 get_import_dates_by_labid() def db_init.get_import_dates_by_labid (lid_sem, att = None, db_name = './grades.sqlite3')

Definition at line 801 of file db_init.py.

```
00801 def get_import_dates_by_labid(lid_sem, att=None, db_name='./grades.sqlite3'):
00802     with lite.connect(db_name) as con:
00803         cur = con.cursor()
00804         if att:
00805             result = cur.execute('SELECT imported_{} FROM lab_schedule WHERE id=?'.format(int(att)), (lid_sem,))
00806         else:
00807             result = cur.execute('SELECT imported_1, imported_2, imported_3, imported_4 FROM lab_schedule WHERE id=?', (lid_sem,))
00808         return result.fetchone()
00809     return None
00810
00811
00812 # save_grade_and_report(self.grade_ids[self.cur_idx], self.final_grade, self.user_comment, self.grader)
```

Referenced by main.get_grading_period().

Here is the caller graph for this function:

6.3.1.13 get_lab_filename() def db_init.get_lab_filename (lab_id, db_name = './grades.sqlite3')

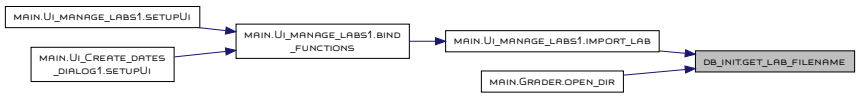
Definition at line 657 of file db_init.py.

```
00657 def get_lab_filename(lab_id, db_name='./grades.sqlite3'):
00658     with lite.connect(db_name) as con:
00659         cur = con.cursor()
00660
00661         result = cur.execute('SELECT mandatory_files FROM lab_names WHERE id=? ', (str(lab_id),))
```

```
00662         return result.fetchall()[0]
00663     return None
00664
00665
```

Referenced by main.Ui_manage_labs1.import_lab(), and main.Grader.open_dir().

Here is the caller graph for this function:



6.3.1.14 get_lab_id() def db_init.get_lab_id (ltype, lab_num)

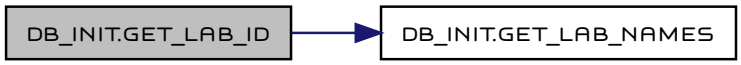
Definition at line 762 of file db_init.py.

```
00762 def get_lab_id(ltype, lab_num):
00763     lab_ids, lab_types, lab_nums = get_lab_names()
00764     for i, lid in enumerate(lab_ids):
00765         if lab_types[i] == ltype and lab_num == lab_nums[i]:
00766             return lid
00767     return None
00768
00769
```

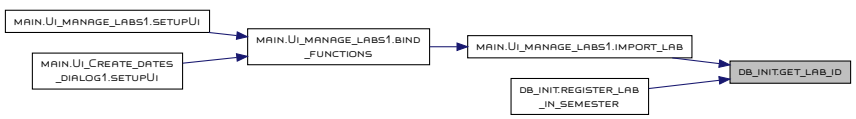
References get_lab_names().

Referenced by main.Ui_manage_labs1.import_lab(), and register_lab_in_semester().

Here is the call graph for this function:



Here is the caller graph for this function:



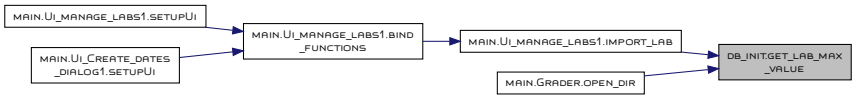
6.3.1.15 get_lab_max_value() def db_init.get_lab_max_value (lab_id, db_name = './grades.sqlite3')

Definition at line 666 of file db_init.py.

```
00666 def get_lab_max_value(lab_id, db_name='./grades.sqlite3'):
00667     with lite.connect(db_name) as con:
00668         cur = con.cursor()
00669 
00670         result = cur.execute('SELECT max_grade FROM lab_names WHERE id=? ', (str(lab_id),))
00671         return int(result.fetchone()[0])
00672     return None
00673 
00674 
```

Referenced by main.Ui_manage_labs1.import_lab(), and main.Grader.open_dir().

Here is the caller graph for this function:



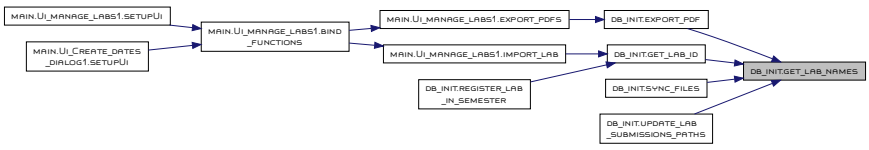
6.3.1.16 get_lab_names() def db_init.get_lab_names (db_name = './grades.sqlite3')

Definition at line 490 of file db_init.py.

```
00490     """
00491     with lite.connect(db_name) as con:
00492         cur = con.cursor()
00493         result = cur.execute("SELECT id, type, num FROM lab_names")
00494         try:
00495             lab_id, lab_type, lab_num = zip(*result.fetchall())
00496         except Exception as e:
00497             print(e)
00498             return None, None, None
00499     return lab_id, lab_type, lab_num
00500 
00501 
```

Referenced by export_pdf(), get_lab_id(), sync_files(), and update_lab_submissions_paths().

Here is the caller graph for this function:



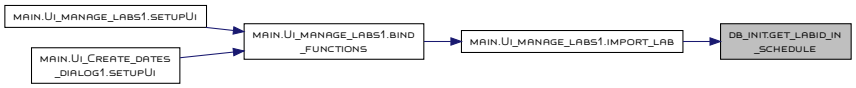
6.3.1.17 get_labid_in_schedule() def db_init.get_labid_in_schedule (lid, year, semester, db_name = './grades.sqlite3')

Definition at line 782 of file db_init.py.

```
00782 def get_labid_in_schedule(lid, year, semester, db_name='./grades.sqlite3'):
00783     with lite.connect(db_name) as con:
00784         cur = con.cursor()
00785         result = cur.execute('SELECT id FROM lab_schedule WHERE lab_id=? AND year=? AND semester=?', (lid, year, semester))
00786         fetched_red = result.fetchone()
00787     return int(fetched_red[0]) if fetched_red is not None else None
00788
00789
```

Referenced by main.Ui_manage_labs1.import_lab().

Here is the caller graph for this function:



6.3.1.18 get_max_grade_for_lab() def db_init.get_max_grade_for_lab (lid, year, semester, db_name = './grades.sqlite3')

Definition at line 620 of file db_init.py.

```
00620 def get_max_grade_for_lab(lid, year, semester, db_name='./grades.sqlite3'):
00621     with lite.connect(db_name) as con:
00622         cur = con.cursor()
00623         result = cur.execute('SELECT e.pipeline_id as pipid, IFNULL(MAX(k.final_grade), 0) as grade '
00624                              'FROM class e '
00625                              'LEFT OUTER JOIN '
00626                              '  (SELECT d.pipeline_id, c.grade*f.percent/100 AS final_grade '
00627                              '   FROM grades c '
00628                              '     JOIN class d ON d.id = c.class_id '
00629                              '     JOIN penalties f ON f.id = c.attempt '
00630                              '   WHERE c.lab = ? ) k '
00631                              'ON e.pipeline_id = k.pipeline_id '
00632                              'WHERE year=? AND semester=? '
00633                              'GROUP BY e.pipeline_id '
00634                              'ORDER BY e.pipeline_id ', (lid, int(year), int(semester)))
00635         return result.fetchall()
00636
00637
```

Referenced by generate.generate_answers3().

Here is the caller graph for this function:

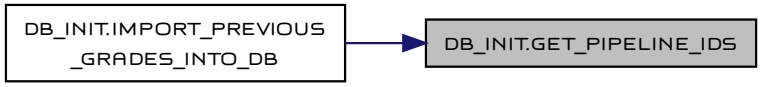
6.3.1.19 get_pipeline_ids() def db_init.get_pipeline_ids (db_name = './grades.sqlite3')

Definition at line 293 of file db_init.py.

```
00293     """
00294     with lite.connect(db_name) as con:
00295         cur = con.cursor()
00296         result = cur.execute("SELECT pipeline_id FROM students")
00297         try:
```

```
00298             resut = (ids[0] for ids in result.fetchall())
00299         except Exception as e:
00300             print(e)
00301             return None
00302     return resut
00303
00304
00305 def get_ids_in_class_by_year_semester(year, semester, db_name='./grades.sqlite3'):
00306     """
00307
00308      year:
00309      semester:
00310      db_name:
00311     :return:
```

Referenced by import_previous_grades_into_db().
Here is the caller graph for this function:



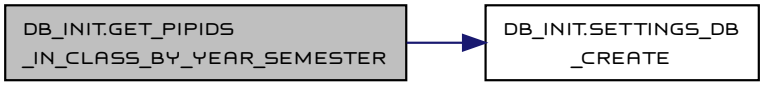
6.3.1.20 get_pipids_in_class_by_year_semester()

```
def db_init.get_pipids_in_class_by_year_semester (
                year,
                semester,
                db_name =  './grades.sqlite3' )
```

Definition at line 822 of file db_init.py.

```
00822 def get_pipids_in_class_by_year_semester(year, semester, db_name='./grades.sqlite3'):
00823     if not os.path.isfile(db_name):
00824         raise Exception("DB not found")
00825     with lite.connect(db_name) as con:
00826         cur = con.cursor()
00827         result = cur.execute('SELECT pipeline_id FROM class WHERE year=? AND semester=?', (year, semester))
00828         all_ids = result.fetchall()
00829     return [elem[0] for elem in all_ids]
00830
00831
00832
```

References settings_db_create().
Referenced by generate.generate_answers3().
Here is the call graph for this function:



Here is the caller graph for this function:

6.3.1.21 get_prev_resp()

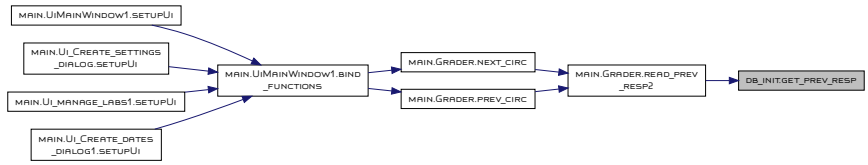
```
def db_init.get_prev_resp (
                grade_id,
                class_id,
                lab_id,
                db_name =  './grades.sqlite3' )
```

Definition at line 446 of file db_init.py.

```
00446 def get_prev_resp(grade_id, class_id, lab_id, db_name='./grades.sqlite3'):
00447     with lite.connect(db_name) as con:
00448         cur = con.cursor()
00449         result = cur.execute("SELECT grader_comment, extra_comment FROM grades WHERE class_id=? AND lab=? AND id<?", (class_id, lab_id,
      grade_id))
00450         res = result.fetchall()
00451     if len(res) == 0:
00452         return ''
00453     else:
00454         gresp, uresp = zip(*res)
00455         return '\n'.join(('{} :\n{}'.format(gresp[i], uresp[i]) for i in range(len(gresp))))
00456
00457
```

Referenced by main.Grader.read_prev_resp2().

Here is the caller graph for this function:



6.3.1.22 get_resp_and_grade()

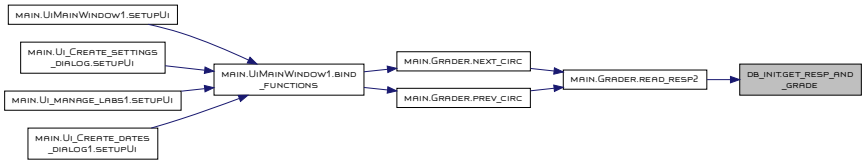
```
def db_init.get_resp_and_grade (
                grade_id,
                db_name =  './grades.sqlite3' )
```

Definition at line 437 of file db_init.py.

```
00437 def get_resp_and_grade(grade_id, db_name='./grades.sqlite3'):
00438     with lite.connect(db_name) as con:
00439         cur = con.cursor()
00440         result = cur.execute("SELECT grade, grader_comment, extra_comment, graded FROM grades WHERE id=?", (grade_id,))
00441         grade, resp, uresp, graded = result.fetchone()
00442
00443     return grade, resp, uresp, graded
00444
00445
```

Referenced by main.Grader.read_resp2().

Here is the caller graph for this function:



6.3.1.23 grades_db_create()

```
def db_init.grades_db_create (
                db_name,
                force =  False )
```

Definition at line 94 of file db_init.py.

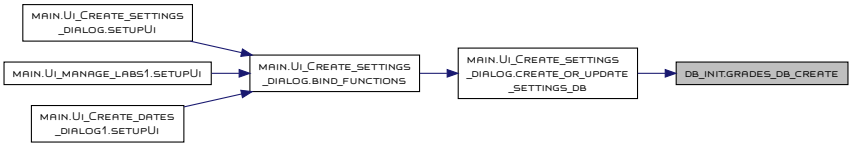
```
00094     """
00095     # from pathlib import Path
00096     print("I am going to create a grades DB with next name: ", db_name)
00097     db_name = str(db_name)
00098     if not os.path.isfile(db_name) or force:
00099         # compute some vars before the connection
00100         lab_names = list()
00101         for i in range(1, 13):
00102             lab_names.append(('CLA' + str(i), 'Closed', i, 10))
00103         for i in range(1, 9):
00104             lab_names.append(('OLA' + str(i), 'Open', i, 20))
00105         lab_names.append(('OLA9', 'Open', 9, 100))
00106         a = list(zip(*lab_names))
00107         a.append(('initial_labs.circ', 'initial_labs.circ', 'seven_seg.circ', 'RSC.circ', 'custom_reg.circ', 'RSC.circ', 'RSC.circ',
       'RSC.circ', 'PLDs.circ', 'PLDs.circ', 'PLDs.circ', ", ",", 'bin_converter.circ', 'mod_counter.circ', 'custom_reg.circ', 'RSC.circ',
       'RSC.circ', 'ram2.txt', "))
00108         b = list(zip(*a))
00109
00110         with lite.connect(db_name) as con:
00111             cur = con.cursor()
00112             # TODO: force should remove 'IF NOT EXISTS' and add 'DROP TABLE' to ensure new table creation
00113             # WISH: add TRY blocks for each CREATE and spawn new info window in case of error
00114             print('Creating students...')
00115             cur.execute("""CREATE TABLE students (
00116                             pipeline_id    TEXT    NOT NULL
00117                                                    PRIMARY KEY,
00118                             first_name     TEXT    NOT NULL,
00119                             second_name    TEXT    NOT NULL,
00120                             comment        TEXT,
00121                             cheating_ratio INTEGER DEFAULT (0)    );""")
00122             con.commit()
00123             print('Done.')
00124             print('Creating semesters...')
00125             cur.execute("""CREATE TABLE semesters (
00126                             semester CHAR (1) NOT NULL PRIMARY KEY,
00127                             name     VARCHAR      );""")
00128             con.commit()
00129             print('Done.')
00130             print('Creating class...')
00131             cur.execute("""CREATE TABLE class (
00132                             id             INTEGER PRIMARY KEY AUTOINCREMENT,
00133                             pipeline_id    TEXT    REFERENCES students (pipeline_id),
00134                             year           INTEGER,
00135                             semester       INTEGER REFERENCES semesters (semester),
00136                             cheating_ratio INTEGER DEFAULT (0),
00137                             UNIQUE (
00138                                 pipeline_id,
00139                                 year,
00140                                 semester)     );""")
00141             con.commit()
00142             print('Done.')
00143             print('Creating labs...')
00144             cur.execute("""CREATE TABLE lab_names (
00145                             id              INT     NOT NULL PRIMARY KEY,
00146                             type            TEXT    NOT NULL,
00147                             num             INTEGER NOT NULL,
00148                             max_grade       INTEGER NOT NULL,
00149                             name            VARCHAR,
00150                             description     VARCHAR,
```

```
00151                           grader_comment  VARCHAR,
00152                           mandatory_files VARCHAR );""")
00153             con.commit()
00154             print('Done.')
00155             print('Creating grades...')
00156             cur.execute("""CREATE TABLE grades (
00157                           id               INTEGER PRIMARY KEY AUTOINCREMENT,
00158                           class_id                 NOT NULL
00159                                                    REFERENCES class (id) ON UPDATE CASCADE,
00160                           lab                      NOT NULL
00161                                                    REFERENCES lab_names (id) ON UPDATE CASCADE,
00162                           attempt          INT     DEFAULT (0),
00163                           submitted        INTEGER,
00164                           graded           INTEGER,
00165                           grade            INTEGER NOT NULL
00166                                                    DEFAULT (0),
00167                           pass_fail        BOOLEAN NOT NULL
00168                                                    DEFAULT (0),
00169                           grader_comment   TEXT,
00170                           extra_comment    TEXT,
00171                           report_generated BOOLEAN,
00172                           report_time      INTEGER,
00173                           lab_path         VARCHAR,
00174                           UNIQUE (
00175                               class_id,
00176                               lab,
00177                               attempt,
00178                               pass_fail) ON CONFLICT REPLACE );""")
00179             con.commit()
00180             print('Done.')
00181
00182             print('Creating lab schedule...')
00183             cur.execute("""CREATE TABLE lab_schedule (
00184                           id         INTEGER PRIMARY KEY AUTOINCREMENT,
00185                           lab_id             REFERENCES lab_names (id),
00186                           year       INTEGER NOT NULL,
00187                           semester   INTEGER REFERENCES semesters (semester)
00188                                              NOT NULL,
00189                           due_date_1 INTEGER,
00190                           due_date_2 INTEGER,
00191                           due_date_3 INTEGER,
00192                           due_date_4 INTEGER,
00193                           imported_1 INTEGER,
00194                           imported_2 INTEGER,
00195                           imported_3 INTEGER,
00196                           imported_4 INTEGER,
00197                           posted_1   INTEGER,
00198                           posted_2   INTEGER,
00199                           posted_3   INTEGER,
00200                           posted_4   INTEGER
00201                       );""")
00202             con.commit()
00203             print('Done.')
00204
00205
00206
00207             print('Filling semesters...')
00208             cur.executemany('INSERT OR REPLACE INTO semesters\
00209                         (semester, name) VALUES (?, ?)', [(1, 'SPRING'), (2, 'SUMMER'), (3, 'FALL')])
00210             con.commit()
00211             print('Done.')
00212             print('Filling labs...')
00213             cur.executemany('INSERT OR REPLACE INTO lab_names\
00214                         (id, type, num, max_grade, mandatory_files) VALUES (?, ?, ?, ?, ?)', b)
00215             con.commit()
00216             print('Done.')
00217             print('Vacuuming...')
00218
00219             cur.execute('VACUUM;')
00220             con.commit()
00221
00222             print('Done.')
00223             print('Creation of GRADES DB finished.')
00224
00225             return True
00226
00227
00228 def load_student_list_into_grades_db(db_name, year, semester, filename='students_list3.txt'):
00229     """
00230     Imports list of students from file in format: `id % lname, fname` into Grades DB.
00231     Should be called before first grading.
```

```
00232      db_name: db that contains grades and student info
00233      year: grading (current) year
00234      semester: grading (current) semester
00235      filename: file that contains student list
00236     :return: nothing
```

Referenced by main.Ui_Create_settings_dialog.create_or_update_settings_db().

Here is the caller graph for this function:



6.3.1.24 import_previous_grades_into_db()

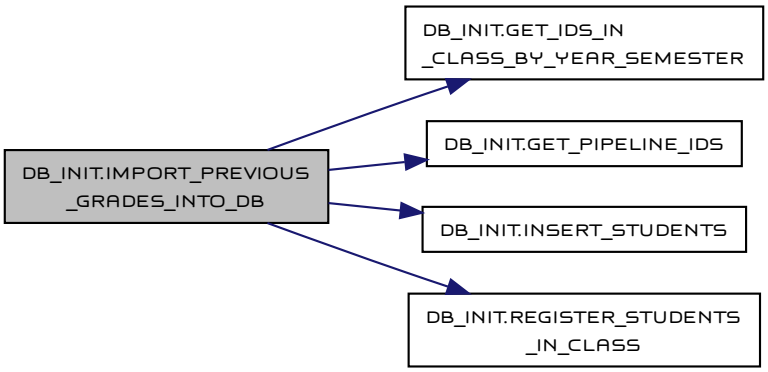
```
def db_init.import_previous_grades_into_db (
        year,
        semester,
        db_name =  './grades.sqlite3',
        filename =  './grades.xls' )
```

Definition at line 336 of file db_init.py.

```
00336     """
00337     if not os.path.isfile(db_name):
00338         raise Exception("DB not found")
00339 
00340     df1 = pd.read_excel(filename)
00341 
00342     try:
00343         cls = df1.filter(like='CL')
00344     except Exception as e:
00345         print(e)
00346         cls = None  # no CLA's found
00347 
00348     try:
00349         ols = df1.filter(like='OL')
00350     except Exception as e:
00351         print(e)
00352         ols = None  # no OLAs found
00353 
00354     try:
00355         ids = df1.filter(like='sername').values.ravel().tolist()
00356         ids_len = len(ids)
00357     except Exception as e:
00358         print('Was not able to parse user ids, check xls file you are trying to import: ', e)
00359         raise e  # may be improved in the future - strange case
00360     try:
00361         names = df1.filter(like='Name').values.ravel().tolist()
00362     except Exception as e:  # either does not exist or has different name
00363         print(e)
00364         names = None
00365 
00366     class_dict = get_ids_in_class_by_year_semester(year, semester, db_name)
00367 
00368     if (not class_dict and not names) or (class_dict and len(class_dict) < ids_len and not names):
00369         raise Exception('Did not find ids in table CLASS and did not find names in xls file')
00370     elif names and (not class_dict or (class_dict and len(class_dict) < ids_len)):
00371         print('Did not find existing students, but found names in xsl\nAdding new students...\n')
00372         existing_ids = get_pipeline_ids(db_name)
00373         need_to_update_students = False
00374         # otherwise just add ids to the class list
00375         if existing_ids:
00376             for sid in ids:
00377                 if sid not in existing_ids:
00378                     need_to_update_students = True
00379         else:
00380             need_to_update_students = True
00381 
00382         if need_to_update_students:
00383             fname, lname = zip(*(name.split(', ') for name in names))
00384             fname = (name.strip() for name in fname)
00385             lname = (name.strip() for name in lname)
00386             insert_students(ids, fname, lname, db_name)
```

```
00387         register_students_in_class(ids, year, semester, db_name)
00388
00389     class_ids = [class_dict[sid] for sid in ids]
00390     if ols is None and cls is None or len(class_ids) == 0:
00391         raise Exception('No grades to load')
00392
00393     grades_tupples = list()
00394     if ols is not None:
00395         for lab_name in ols:
00396             grades = (str(grade) for grade in ols[lab_name].values)
00397             grades_tupples += list(zip(class_ids, [lab_name] * ids_len, [-1] * ids_len, grades, ['TRUE'] * ids_len))
00398
00399     if cls is not None:
00400         for lab_name in cls:
00401             grades = (str(grade) for grade in cls[lab_name].values)
00402             grades_tupples += list(zip(class_ids, [lab_name] * ids_len, [-1] * ids_len, grades, ['TRUE'] * ids_len))
00403
00404     with lite.connect(db_name) as con:
00405         cur = con.cursor()
00406         cur.executemany('INSERT OR REPLACE INTO grades\
00407                     (class_id, lab, attempt, grade, pass_fail) VALUES (?, ?, ?, ?, ?)', grades_tupples)
00408         con.commit()
00409
00410
```

References get_ids_in_class_by_year_semester(), get_pipeline_ids(), insert_students(), and register_students_in_class().
Here is the call graph for this function:

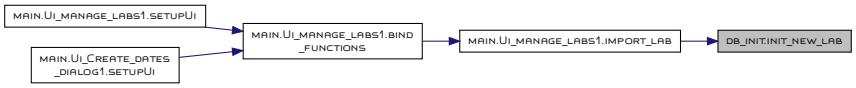


6.3.1.25 init_new_lab() def db_init.init_new_lab (stud_id, lab_name, att, submitted, lab_path, db_name = './grades.sqlite3')

Definition at line 476 of file db_init.py.

```
00476 def init_new_lab(stud_id, lab_name, att, submitted, lab_path, db_name='./grades.sqlite3'):
00477     if not os.path.isfile(db_name):
00478         raise Exception("DB not found")
00479     with lite.connect(db_name) as con:
00480         cur = con.cursor()
00481         cur.execute('INSERT INTO grades (class_id, lab, attempt, submitted, lab_path) VALUES (?, ?, ?, ?, ?)', (stud_id, lab_name, att,
       submitted, lab_path))
00482         con.commit()
00483
00484
00485 def get_lab_names(db_name='./grades.sqlite3'):
00486     """
00487
00488      db_name:
00489     Returns:
```

Referenced by main.Ui_manage_labs1.import_lab().
Here is the caller graph for this function:



6.3.1.26 insert_students() def db_init.insert_students (ids, fname, lname, db_name = './grades.sqlite3')

Definition at line 260 of file db_init.py.

```
00260     """
00261     names_tupple = list(zip(ids, fname, lname, [0] * len(ids)))
00262     with lite.connect(db_name) as con:
00263         cur = con.cursor()
00264         cur.executemany('INSERT OR REPLACE INTO STUDENTS \
00265                     (pipeline_id, first_name, second_name, cheating_ratio)'
00266                         ' VALUES (?, ?, ?, ?)', names_tupple)
00267         con.commit()
00268 
00269 
00270 def register_students_in_class(pipeline_ids, year, semester, db_name='./grades.sqlite3'):
00271     """
00272 
00273      pipeline_ids:
00274      year:
00275      semester:
00276      db_name:
00277     :return:
```

Referenced by import_previous_grades_into_db(), and load_student_list_into_grades_db().
Here is the caller graph for this function:

6.3.1.27 load_student_list_into_grades_db() def db_init.load_student_list_into_grades_db (db_name, year, semester, filename = 'students_list3.txt')

Definition at line 237 of file db_init.py.

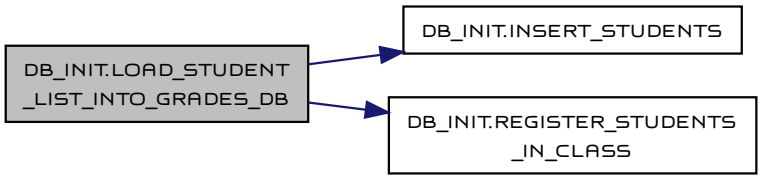
```
00237     """
00238 
00239     with open(filename, 'r') as sl:
00240         ids, names = zip(*(line.strip().split('%') for line in sl))
00241         ids = list(sid.strip() for sid in ids)
00242         names = (name.strip() for name in names)  # for case when file contains extra whitespaces
00243         lname, fname = zip(*(namer.split(',') for namer in names))
00244         lname = (name.strip() for name in lname)
00245         fname = (name.strip() for name in fname)
00246 
00247     if os.path.isfile(db_name):
00248         insert_students(ids, fname, lname, db_name)
00249         register_students_in_class(ids, year, semester, db_name)
00250 
00251 
00252 def insert_students(ids, fname, lname, db_name='./grades.sqlite3'):
00253     """
```

```
00254     Takes students' info from the parameters and insert them into grades DB
00255      ids: pipeline ids
00256      fname: first name
00257      lname: last name
00258      db_name: specific name for grades DB
00259     :return: nothing
```

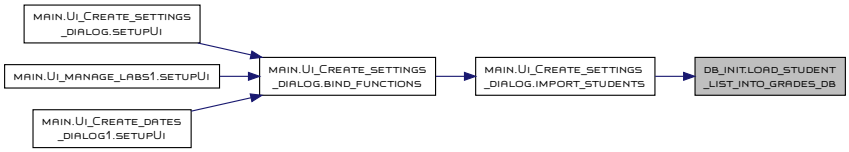
References insert_students(), and register_students_in_class().

Referenced by main.Ui_Create_settings_dialog.import_students().

Here is the call graph for this function:



Here is the caller graph for this function:



6.3.1.28 reconstruct_grades_and_comments()

def db_init.reconstruct_grades_and_comments (db_name = './grades.sqlite3')

Definition at line 556 of file db_init.py.

```
00556 def reconstruct_grades_and_comments(db_name='./grades.sqlite3'):
00557     lab_id, lab_path = get_empty_grades(db_name)
00558     updated_grades = list()
00559     for l_iter in range(len(lab_path)):
00560         lpath = lab_path[l_iter]
00561         submition_t = int(lpath.split('-')[-1])
00562         try:
00563             with open(lpath+'/grade.txt', 'r') as gfile:
00564                 cur_grade = int(gfile.readline().strip())
00565         except Exception as e:
00566             print("Error during grade file reading :", e)
00567             cur_grade = 0
00568         try:
00569             cur_t_graded = int(os.path.getmtime(lpath + '/grade.txt'))
00570         except Exception as e:
00571             print("Error during grade file statistics retrieval: ", e)
00572             cur_t_graded = None
00573 
00574         pass_fail = 'TRUE' if cur_grade else 'FALSE'
00575         try:
00576             with open(lpath+'/responce.txt', 'r') as rfile:
00577                 cur_resp = rfile.readlines()
00578                 if type(cur_resp) == list:
00579                     cur_resp = ' '.join(cur_resp)
00580         except Exception as e:
00581             print("Error during grade file reading", e)
00582             cur_resp = 'NULL'
00583         updated_grades.append((submition_t, cur_grade, cur_t_graded, pass_fail, cur_resp, lab_id[l_iter]))
00584 
00585 
```

```
00586     with lite.connect(db_name) as con:
00587         cur = con.cursor()
00588         cur.executemany('UPDATE grades SET submitted=?, grade=?, graded=?, pass_fail=?, grader_comment=? '
00589                         'WHERE id=?', updated_grades)
00590         con.commit()
00591
00592     with lite.connect(db_name) as con:
00593         cur = con.cursor()
00594         cur.execute('VACUUM;')
00595         con.commit()
00596
00597
```

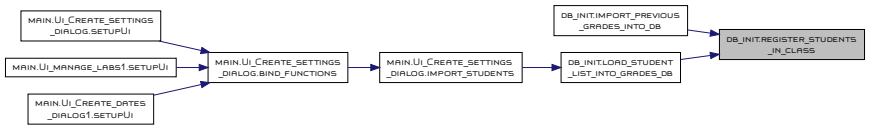
6.3.1.29 register_lab_in_semester() def db_init.register_lab_in_semester (ltype, lab_num, year, semester, due_dates, db_name = './grades.sqlite3')

```
Definition at line 770 of file db_init.py.
00770 def register_lab_in_semester(ltype, lab_num, year, semester, due_dates, db_name='./grades.sqlite3'):
00771     lid = get_lab_id(ltype, int(lab_num))
00772     # TODO: add a check so you do not insert lab twice
00773     if lid is None:
00774         raise Exception('No such lab')
00775     if not os.path.isfile(db_name):
00776         raise Exception("DB not found")
00777     with lite.connect(db_name) as con:
00778         cur = con.cursor()
00779         cur.execute('INSERT OR REPLACE INTO lab_schedule (lab_id, year, semester, due_date_1, due_date_2, due_date_3, due_date_4) VALUES (?, ?,
       ?, ?, ?, ?, ?)', (lid, year, semester, due_dates[0], due_dates[1], due_dates[2], due_dates[3]))
00780         con.commit()
00781
References get_lab_id().
Here is the call graph for this function:
```

6.3.1.30 register_students_in_class() def db_init.register_students_in_class (pipeline_ids, year, semester, db_name = './grades.sqlite3')

```
Definition at line 278 of file db_init.py.
00278     """
00279     len_id = len(pipeline_ids)
00280     names_tupple = list(zip(pipeline_ids, [year] * len_id, [semester] * len_id, [0] * len_id))
00281     with lite.connect(db_name) as con:
00282         cur = con.cursor()
00283         cur.executemany('INSERT OR REPLACE INTO class\
00284                     (pipeline_id, year, semester, cheating_ratio) VALUES (?, ?, ?, ?)', names_tupple)
00285         con.commit()
00286
00287
00288 def get_pipeline_ids(db_name='./grades.sqlite3'):
00289     """
00290
00291      db_name:
00292     :return:
Referenced by import_previous_grades_into_db(), and load_student_list_into_grades_db().
```

Here is the caller graph for this function:



6.3.1.31 save_a_grade_to_db() def db_init.save_a_grade_to_db (grade_id, grade, grader_comment, extra_comment, grader_name, graded = True, pass_fail = True, db_name = './grades.sqlite3')

Definition at line 458 of file db_init.py.

```
00458 def save_a_grade_to_db(grade_id, grade, grader_comment, extra_comment, grader_name, graded=True, pass_fail=True, db_name='./grades.sqlite3'):
00459     pass
00460
00461
00462 # def get_submissions_to_grade(lab_id, att, db_name='./grades.sqlite3'):
00463 #     if not os.path.isfile(db_name):
00464 #         raise Exception("DB not found")
00465 #     with lite.connect(db_name) as con:
00466 #         cur = con.cursor()
00467 #         result = cur.execute("SELECT id, FROM grades where lab=lab_id attempt=att and graded is NULL")
00468 #         try:
00469 #             lab_id, lab_type, lab_num = zip(*result.fetchall())
00470 #         except Exception as e:
00471 #             print(e)
00472 #             return None, None, None
00473 #     return lab_id, lab_type, lab_num
00474
00475
```

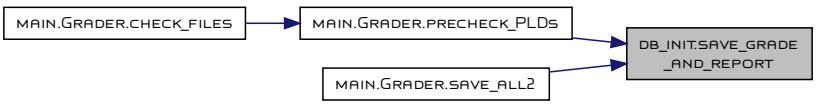
6.3.1.32 save_grade_and_report() def db_init.save_grade_and_report (grade_id, grade, report, user_comment, grader, db_name = './grades.sqlite3')

Definition at line 743 of file db_init.py.

```
00743 def save_grade_and_report(grade_id, grade, report, user_comment, grader, db_name='./grades.sqlite3'):
00744     if not os.path.isfile(db_name):
00745         raise Exception("DB not found")
00746     with lite.connect(db_name) as con:
00747         cur = con.cursor()
00748         cur.execute("UPDATE grades SET graded=strftime('%s','now'), pass_fail=TRUE, grade=?, grader_comment=?, extra_comment=?, grader=? WHERE
      id=?", (grade, report, user_comment, grader, grade_id))
00749         con.commit()
00750
00751
```

Referenced by main.Grader.precheck_PLDs(), and main.Grader.save_all2().

Here is the caller graph for this function:



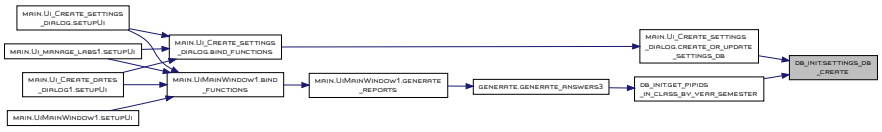
6.3.1.33 settings_db_create() def db_init.settings_db_create (db_name = SETTINGS_DB_NAME, force = False)

Definition at line 18 of file db_init.py.

```
00018     """
00019     if not force and os.path.isfile(db_name):
00020         user_choice = input('Do you really want to drop database ? Type "yes" to continue\n ')
00021         if not user_choice.isalpha() or not user_choice.lower() == 'yes':
00022             return False
00023 
00024     # DB creation logic goes here
00025     with lite.connect(db_name) as con:
00026         cur = con.cursor()
00027         cur.execute('DROP TABLE IF EXISTS PATHS')
00028         cur.execute("CREATE TABLE PATHS "
00029                     "( LOGISIM_HOME VARCHAR NOT NULL,\
00030                        GRADING_PATH VARCHAR NOT NULL,\
00031                        IMPORT_PATH  VARCHAR,\
00032                        GRADES_DB  VARCHAR); ")
00033         cur.execute("CREATE TABLE LOCAL (\
00034                     GRADER_NAME VARCHAR,\
00035                     YEAR        INT,\
00036                     SEMESTER    CHAR (1),\
00037                     USE_STYLE   BOOLEAN,\
00038                      SYNC_COMMAND VARCHAR);")
00039         con.commit()
00040     return True
00041 
00042 
00043 def settings_db_read_settings(db_name=SETTINGS_DB_NAME):
00044     """
00045     Reads settings from the DB with specified name in 'db_name'
00046      db_name: name of DB to query
00047     :return: paths - list of paths to various locations and local - info about grader, grading year, etc.
```

Referenced by main.Ui_Create_settings_dialog.create_or_update_settings_db(), and get_pipids_in_class_by_year_semester().

Here is the caller graph for this function:



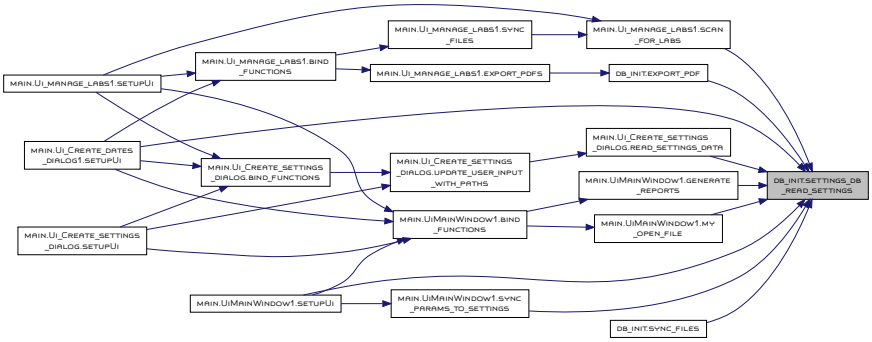
6.3.1.34 settings_db_read_settings() def db_init.settings_db_read_settings (db_name = SETTINGS_DB_NAME)

Definition at line 48 of file db_init.py.

```
00048     """
00049     paths = local = None
00050     if os.path.isfile(db_name):
00051         with lite.connect(db_name) as con:
00052             cur = con.cursor()
00053             result = cur.execute("SELECT LOGISIM_HOME, GRADING_PATH, IMPORT_PATH, GRADES_DB\
00054                                   FROM PATHS")
00055             paths = result.fetchone()
00056             result = cur.execute("SELECT GRADER_NAME, YEAR, SEMESTER, USE_STYLE, SYNC_COMMAND\
00057                                           FROM LOCAL")
00058             local = result.fetchone()
00059 
00060     return paths, local
00061 
00062 
00063 def update_settings(paths, local, db_name=SETTINGS_DB_NAME):
00064     """
00065     Procedure that loads parameters specified in paths and local into settings DB
00066      paths: list of paths to various locations
00067      local: local - info about grader, grading year, etc.
00068      db_name: name of DB to query to update
00069     :return: nothing
```

Referenced by export_pdf(), main.UiMainWindow1.generate_reports(), main.UiMainWindow1.my_open_file(), main.Ui_Create_settings_dialog.read_settings_data(), main.Ui_manage_labs1.scan_for_labs(), main.UiMainWindow1.setupUi(), main.Ui_Create_dates_dialog1.setupUi(), sync_files(), and main.UiMainWindow1.sync_params_to_settings().

Here is the caller graph for this function:



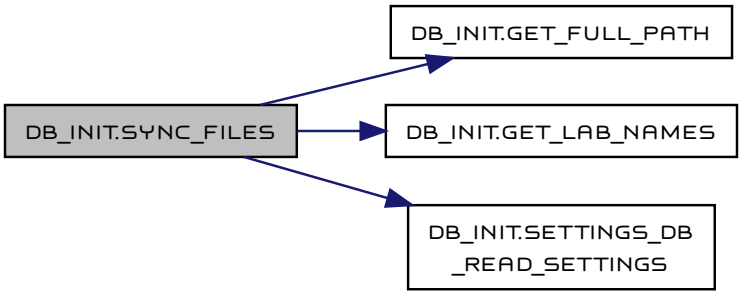
6.3.1.35 sync_files() def db_init.sync_files (self = None)

Definition at line 680 of file db_init.py.

```
00680 def sync_files(self=None):
00681     import subprocess
00682     import os
00683 
00684     paths, local = settings_db_read_settings()
00685     full_path = get_full_path(paths, local) + "/server_sync/"
00686     lab_ids, lab_types, lab_nums = get_lab_names()
00687     lab_names = []
00688     for i in range(len(lab_types)):
00689         lab_names.append(lab_types[i] + '_Lab_' + str(lab_nums[i]))
00690 
00691     if not os.path.isdir(full_path):
00692         os.makedirs(full_path)
00693         for lab_name in lab_names:
00694             os.makedirs(full_path + lab_name)
00695 
00696     proc_arr = []
00697     for lab_name in lab_names:
00698         command = local[4] + ' ' + os.path.expanduser(paths[2] + lab_name) + '/*.zip' + ' ' + full_path + lab_name + '/'
00699         try:
00700             proc_arr.append(subprocess.Popen(os.path.expandvars(command), stdout=subprocess.PIPE, shell=True))
00701             proc_arr[-1].communicate()
00702         except Exception as e:
00703             print('Error in rsync: ', e)
00704         # output, error = process.communicate()
00705         # print(output)
00706         # print(error)
00707 
00708     for proc_elem in proc_arr:
00709         proc_elem.wait()
00710 
00711 
```

References get_full_path(), get_lab_names(), and settings_db_read_settings().

Here is the call graph for this function:



6.3.1.36 update_lab_submissions_paths()

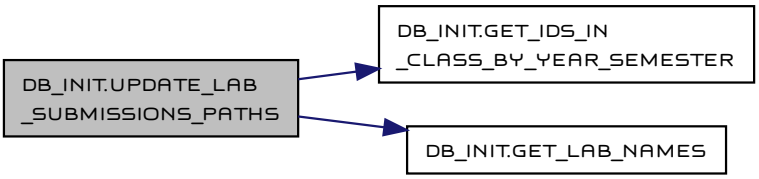
```
def db_init.update_lab_submissions_paths (
                db_name,
                repository_root,
                year,
                semester )
```

Definition at line 502 of file db_init.py.

```
00502 def update_lab_submissions_paths(db_name, repository_root, year, semester):
00503     import fnmatch
00504     import glob
00505     # import_previous_grades_into_db(year, semester, db_name, repository_root+'grades.xlsx')
00506     lab_id, lab_type, lab_num = get_lab_names()
00507     if lab_id is None or lab_type is None or lab_num is None:
00508         raise Exception("Error during lab type/num import: ")
00509     class_dict = get_ids_in_class_by_year_semester(year, semester, db_name)
00510     total_labs = len(lab_type)
00511 
00512     all_dirs = list()
00513     for lab_iter in range(total_labs):
00514         for attempt in range(1, 5):  # class rule - 4 attempts
00515             full_lab_name = repository_root + lab_type[lab_iter] + '_Lab_' + str(lab_num[lab_iter]) + '_' + str(attempt) + '/'
00516             print('Processing ', full_lab_name)
00517             for stud_id in class_dict.keys():
00518                 found_dir = glob.glob(full_lab_name+stud_id+'*')
00519                 if found_dir:
00520                     # since it is initial pass, we do not set pass/fail. It will be set later with grade and comment.
00521                     all_dirs.append((class_dict[stud_id], lab_id[lab_iter], attempt, 'FALSE', found_dir[-1]))
00522 
00523     with lite.connect(db_name) as con:
00524         cur = con.cursor()
00525         cur.executemany('INSERT OR REPLACE INTO grades (class_id, lab, attempt, pass_fail, lab_path)'
00526                         ' VALUES (?, ?, ?, ?, ?)', all_dirs)
00527         con.commit()
00528 
00529 
```

References get_ids_in_class_by_year_semester(), and get_lab_names().

Here is the call graph for this function:



6.3.1.37 update_settings()

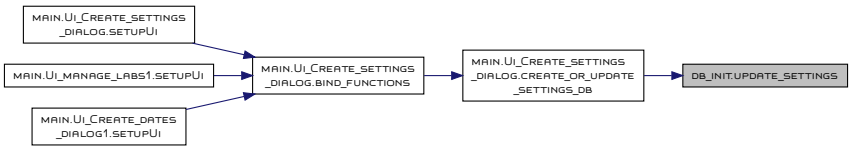
```
def db_init.update_settings (
                paths,
                local,
                db_name =  SETTINGS_DB_NAME )
```

Definition at line 70 of file db_init.py.

```
00070     """
00071     if os.path.isfile(db_name):
00072         with lite.connect(db_name) as con:
00073             cur = con.cursor()
00074             cur.execute('DELETE FROM PATHS;')
00075             cur.execute('INSERT OR REPLACE INTO PATHS (LOGISIM_HOME, GRADING_PATH, IMPORT_PATH, GRADES_DB)'
00076                         ' VALUES (?, ?, ?, ?);', paths)
00077             cur.execute('DELETE FROM LOCAL;')
00078             cur.execute('INSERT OR REPLACE INTO LOCAL (GRADER_NAME, YEAR, SEMESTER, USE_STYLE, SYNC_COMMAND)'
00079                         'VALUES (?, ?, ?, ?, ?);', local)
00080             con.commit()
00081 
00082         with lite.connect(db_name) as con:
00083             cur = con.cursor()
00084             cur.execute('VACUUM;')
00085             con.commit()
00086 
00087 
00088 def grades_db_create(db_name, force=False):
00089     """
00090     Will create database that contains all information about grades
00091      db_name: path and name of the database
00092      force: flag to overwrite existing db
00093     :return: Unknown
```

Referenced by main.Ui_Create_settings_dialog.create_or_update_settings_db().

Here is the caller graph for this function:



6.3.2 Variable Documentation

6.3.2.1 SETTINGS_DB_NAME

```
string db_init.SETTINGS_DB_NAME = 'settings.sqlite3'
```

Definition at line 8 of file db_init.py.

6.4 generate Namespace Reference

Functions

- def convert_to_pdf (html_file, func_type)
- def create_html_pdf_report2 (lab_dict)

  Creates nice html report for submitted labs and converts it to pdf format.
- def create_html_pdf_zero_report (filename, stud_name, top_part, bot_part)
- def create_not_submitted (stud_id, lab_type, lab_num, dir_name)
- def generate_answers3 (lid, att, year, semester, db_name='./grades.sqlite3')
- def time_to_str_with_tz (in_time)

6.4.1 Function Documentation

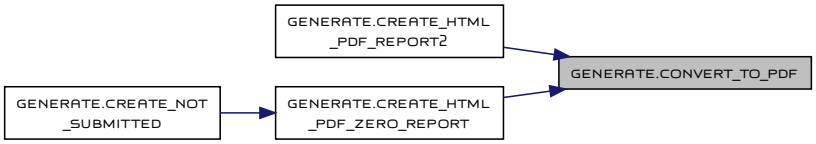
6.4.1.1 convert_to_pdf() def generate.convert_to_pdf (html_file, func_type)

Definition at line 19 of file generate.py.

```
00019     """
00020     if func_type == "wkhtmltopdf":  # old way
00021         from subprocess import call
00022         call(["wkhtmltopdf", "-q", html_file, html_file[:-4] + 'pdf'])
00023     elif func_type == "pdfkit":  # best margins
00024         import pdfkit
00025         options = {
00026             'page-size': 'A4',
00027             'margin-top': '0.0in',
00028             'margin-right': '0.0in',
00029             'margin-bottom': '0.0in',
00030             'margin-left': '0.0in',
00031         }
00032         pdfkit.from_url(html_file, html_file[:-4] + 'pdf', options=options)
00033     elif func_type == 'weasyprint':  # potentially the fastest
00034         # if string is passed as param, but has margins problem
00035         from weasyprint import HTML
00036         with open(html_file, 'r') as html_in_file:
00037             cont = html_in_file.readlines()
00038         str_file = ''.join(cont)
00039         pdf = HTML(string=str_file)
00040         pdf.write_pdf(html_file[:-4] + 'pdf')
00041
00042
00043 # def create_html_pdf_report(joined_path, stud_name, cur_dir, grade, max_points, penalty,
00044 #                            final_score, top_part, bot_part, generated_time):
00045 #     """
00046 #     Creates nice html report for submitted labs and converts it to pdf format.
00047 #     TODO: use latex instead of ugly html.
00048 #      joined_path: working directory
00049 #      stud_name: full student name(first, last)
00050 #      cur_dir: directory with all labs(usually same as joined_path)
00051 #      grade: what grade to assign.
00052 #      max_points: max possible grade for this lab.
00053 #      penalty: usually for resubmission, like 90%, 70%...
00054 #      final_score: final grade = grade * penalty
00055 #      top_part: predefined top part of html document
00056 #      bot_part: predefined bottom part of html document
00057 #      generated_time: some extra statistics for curious students.
00058 #     Returns: nothing, pdf is generated instead.
00059 #     """
00060 #     with open(joined_path + '-returned.html', 'w') as stud_report:
00061 #         stud_report.writelines(top_part)
00062 #         stud_report.write('<p>Grading directory : ' + cur_dir + ' </br>')
00063 #
00064 #         with open(joined_path + '/tech_info.txt', 'r') as tech_file:
00065 #             stud_report.writelines(tech_file.readlines())
00066 #
00067 #         stud_report.write('</p><p><i>Dear ' + stud_name + ', ')
00068 #
00069 #         with open(joined_path + '/responce.txt', 'r') as resp_file:
00070 #             stud_report.writelines(resp_file.readlines())
00071 #
00072 #         stud_report.write("</i></p>\n"
00073 #                           "<p>According to the comment above, next grade was assigned: "
00074 #                           "%d of %d <br/>\n \
00075 #                           Your final grade is %d*%.1f=<b>%d</b> of %d <br/>\n"
00076 #                           % (grade, max_points, grade, penalty, final_score, max_points))
00077 #         stud_report.write('This report was generated {} </p>'.format(generated_time))
00078 #         # TODO add current date/time
```

```
00079 #         stud_report.writelines(bot_part)
00080 #
00081 #     convert_to_pdf(joined_path + '-returned.html', "pdfkit")
00082 #     os.remove(joined_path + '-returned.html')
00083 #
00084
```

Referenced by create_html_pdf_report2(), and create_html_pdf_zero_report().
Here is the caller graph for this function:



6.4.1.2 create_html_pdf_report2() def generate.create_html_pdf_report2 (lab_dict)

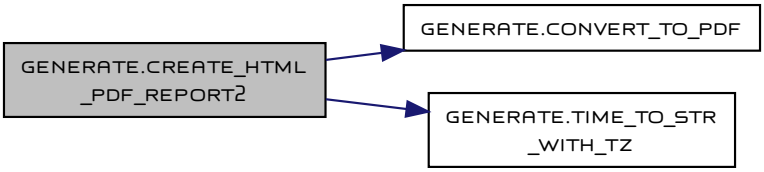
Creates nice html report for submitted labs and converts it to pdf format.
:return: nothing, pdf is generated instead.
Definition at line 90 of file generate.py.

```
00090     """
00091     with open('./answer.top', 'r') as partial_html:
00092         top_part = partial_html.readlines()
00093
00094     with open('./answer.bottom', 'r') as partial_html:
00095         bot_part = partial_html.readlines()
00096
00097     with open(lab_dict['lab_path'] + '-returned.html', 'w') as stud_report:
00098         stud_report.writelines(top_part)
00099
00100         stud_report.write('<p>Grading directory : {} </br>'.format(lab_dict['lab_path'].split('/')[-1]))
00101         stud_report.write('Due date was {} <br/>'.format(time_to_str_with_tz(lab_dict['due_date'])))
00102         stud_report.write('File was submited at {} <br/>'.format(time_to_str_with_tz(lab_dict['submitted'])))
00103         stud_report.write('I imported your file at {} <br/>'.format(time_to_str_with_tz(lab_dict['import_date'])))
00104         if lab_dict['graded'] is not None:
00105             stud_report.write('I graded your lab at {} <br/>'.format(time_to_str_with_tz(lab_dict['graded'])))
00106         else:
00107             stud_report.write('I did not grad your lab or grade timestamp was not set.<br/>')
00108         stud_report.write('Lab type : \'{}\' and it\'s number is \'{}\' <br/>'.format(lab_dict['type'], lab_dict['num']))
00109         stud_report.write('</p><p><i>Dear {} {}, '.format(lab_dict['first_name'], lab_dict['second_name']))
00110         if lab_dict['grader_comment'] is None or len(lab_dict['grader_comment']) < 2:
00111             stud_report.write('There were no comments.')
00112         else:
00113             stud_report.write(lab_dict['grader_comment'])
00114         if lab_dict['extra_comment'] is not None and len(lab_dict['extra_comment']) > 0:
00115             stud_report.write('<br/>\nExtra comment: {}'.format(lab_dict['extra_comment']))
00116
00117         stud_report.write("</i></p>\n"
00118                           "<p>According to the comment above, next grade was assigned: {} of {} <br/>\n"
00119                           " Your final grade is computed as {}*{:.1f}=<b>{}</b> of {} <br/>\n"
00120                           "".format(lab_dict['final_grade'], lab_dict['max_grade'], lab_dict['grade'], lab_dict['percent']/100,
      lab_dict['final_grade'], lab_dict['max_grade']))
00121         if lab_dict['grade'] == 0:
00122             stud_report.write('<br/>Don\'t forget to resubmit it by {} <br/><br/>\n'.format(time_to_str_with_tz(lab_dict['due_date'] +
      604800)))  # one extra week
00123         stud_report.write('This report was generated {} </p>\n'.format(QDateTime.currentDateTime().toString(Qt.DefaultLocaleLongDate)))
00124
00125         stud_report.writelines(bot_part)
00126
00127     convert_to_pdf(lab_dict['lab_path'] + '-returned.html', "pdfkit")
00128     os.remove(lab_dict['lab_path'] + '-returned.html')
00129
00130
00131 def create_html_pdf_zero_report(filename, stud_name, top_part, bot_part):
00132     """
00133     Creates nice html report for nonsubmitted labs and converts it to pdf format.
00134      filename: filename with correct naming(zeroes instead of timestamp)
00135      stud_name: full student name(first, last)
00136      top_part: predefined top part of html document
```

```
00137      bot_part: predefined bottom part of html document
00138     :return:
```

References convert_to_pdf(), and time_to_str_with_tz().

Here is the call graph for this function:



6.4.1.3 create_html_pdf_zero_report()

```
def generate.create_html_pdf_zero_report (
                filename,
                stud_name,
                top_part,
                bot_part )
```

Definition at line 139 of file generate.py.

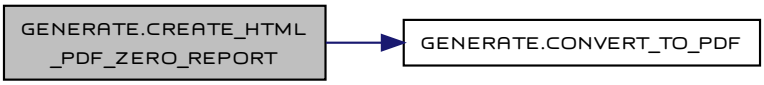
```
00139     """
00140     with open(filename, 'w') as zeroes_file:
00141         zeroes_file.writelines(top_part)
00142         zeroes_file.write(stud_name + ' : You did not submit your lab. :(</p>\n')
00143         zeroes_file.write("<p>According to comments above, next grade was assigned : 0 </p>")
00144         zeroes_file.write("<p>Please submit your file before the next due date.")
00145         zeroes_file.writelines(bot_part)
00146     convert_to_pdf(filename, "pdfkit")
00147     os.remove(filename)
00148
00149
00150 # def generate_answers(resubmit_num, dir_name, lab_type, lab_num, year, semester, grader_name):
00151 #     """
00152 #     general function that figures out max points, filenames, etc
00153 #     and calls generate function with appropriate parameters
00154 #      resubmit_num: resubmission attempt
00155 #      dir_name: working dir
00156 #      lab_type: open or closed lab
00157 #      lab_num: just lab identifier
00158 #      year: used wit semester to identify correct class list
00159 #      semester: used wit year to identify correct class list
00160 #      grader_name: name that will be displayed in the report
00161 #     Returns:
00162 #     """
00163 #     students = {}
00164 #     # select
00165 #
00166 #     ids = get_pipids_in_class_by_year_semester(year, semester, 'grades.sqlite3')
00167 #     with lite.connect('grades.sqlite3') as con:
00168 #         cur = con.cursor()
00169 #
00170 #         for sid in ids.keys():
00171 #             result = cur.execute('SELECT first_name, second_name FROM students WHERE pipeline_id=?', (str(sid),))
00172 #             students[sid] = " ".join(result.fetchall()[0])
00173 #
00174 #     if not students:
00175 #         with open('students_list1.txt', 'r') as stud_list_file:
00176 #             temp_arr = stud_list_file.readlines()
00177 #             for line in temp_arr:
00178 #                 sid, name = line.split('%')
00179 #                 students[sid.strip()] = name.strip()
00180 #         del temp_arr
00181 #
00182 #
00183 #     if lab_type == 'Closed':
00184 #         max_points = 10
00185 #         type_for_name = 'CL'
00186 #     elif lab_type == 'Open':
00187 #         max_points = 20
```

```
00188 #         type_for_name = 'OL'
00189 #     else:
00190 #         raise Exception('Unknown lab type')
00191 #
00192 #     if resubmit_num == 1:
00193 #         penalty = 1.0
00194 #     elif resubmit_num == 2:
00195 #         penalty = 0.9
00196 #     elif resubmit_num == 3:
00197 #         penalty = 0.7
00198 #     elif resubmit_num == 4:
00199 #         penalty = 0.5
00200 #     else:
00201 #         penalty = 0.0
00202 #
00203 #     generated_time = QDateTime.currentDateTime().toString(Qt.DefaultLocaleLongDate)
00204 #
00205 #     print('This is ', type_for_name, ' lab, so max points is ', max_points)
00206 #
00207 #     try:
00208 #         shutil.rmtree(dir_name + 'Answers', ignore_errors=True)
00209 #         os.remove(dir_name + "grades.csv")
00210 #         os.remove(dir_name + "grades_for_" + type_for_name + "lab_num.csv")
00211 #     except Exception as e:
00212 #         print('Exception during dir preparetion : ', e)
00213 #
00214 #     dirs = os.walk(dir_name).__next__()[1]
00215 #
00216 #     with open('./answer.top', 'r') as partial_html:
00217 #         top_part = partial_html.readlines()
00218 #
00219 #     with open('./answer.bottom', 'r') as partial_html:
00220 #         bot_part = partial_html.readlines()
00221 #
00222 #     grades = list()
00223 #     for cur_dir in dirs:
00224 #         student_id = cur_dir.split('-')[0]
00225 #         joined_path = os.path.join(dir_name, cur_dir)
00226 #         with open(joined_path + '/grade.txt', 'r') as grade_file:
00227 #             grade = grade_file.readlines()
00228 #
00229 #         grade = int(grade[0].strip())
00230 #         final_score = grade * penalty
00231 #         grades.append((student_id, final_score))
00232 #         create_html_pdf_report(joined_path, students[student_id], cur_dir, grade,
00233 #                                max_points, penalty, final_score, top_part, bot_part, generated_time)
00234 #
00235 #     submitted = [x.split('-')[0] for x in dirs]
00236 #
00237 #     zeroes = list()
00238 #     for student in students:
00239 #         if student not in submitted:
00240 #             grades.append((student, 0))
00241 #             zeroes.append(student)
00242 #
00243 #     if resubmit_num == 1:
00244 #         for student_id in zeroes:
00245 #             filename = '%s/%s-%s%d-0000000000-returned' % \
00246 #                        (dir_name, student_id, type_for_name, lab_num)
00247 #             create_html_pdf_zero_report(filename+'.html', students[student_id], top_part, bot_part)
00248 #
00249 #     with open(dir_name + '/' + 'grades.csv', 'w') as grades_file:
00250 #         for grade in sorted(grades):
00251 #             grades_file.write("%s, %f \n" % grade)
00252 #
00253 #     os.mkdir(dir_name + '/Answers')
00254 #     files = os.walk(dir_name).__next__()[2]
00255 #     for file in files:
00256 #         if file[-3:] == 'pdf':
00257 #             shutil.move(dir_name + '/' + file, dir_name + '/Answers/' + file)
00258 #
00259 #     print('Done')
00260 #
00261 #
00262 # def generate_answers2(resubmit_num, dir_name, lab_type, lab_num, year, semester, grader_name):
00263 #     """
00264 #     general function that figures out max points, filenames, etc
00265 #     and calls generate function with appropriate parameters
00266 #      resubmit_num: resubmission attempt
00267 #      dir_name: working dir
00268 #      lab_type: open or closed lab
```

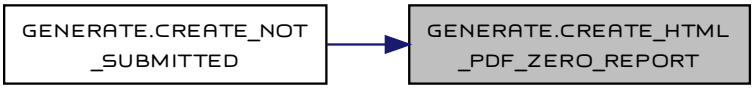
```
00269 #      lab_num: just lab identifier
00270 #      year: used wit semester to identify correct class list
00271 #      semester: used wit year to identify correct class list
00272 #      grader_name: name that will be displayed in the report
00273 #     Returns:
00274 #     """
00275 #     students = {}
00276 #     # select
00277 #     import sqlite3 as lite
00278 #     from db_init import get_ids_in_class_by_year_semester
00279 #     ids = get_ids_in_class_by_year_semester(year, semester, 'grades.sqlite3')
00280 #     with lite.connect('grades.sqlite3') as con:
00281 #         cur = con.cursor()
00282 #
00283 #         for sid in ids.keys():
00284 #             result = cur.execute('SELECT first_name, second_name FROM students WHERE pipeline_id=?', (str(sid),))
00285 #             students[sid] = " ".join(result.fetchall()[0])
00286 #
00287 #     if not students:
00288 #         with open('students_list1.txt', 'r') as stud_list_file:
00289 #             temp_arr = stud_list_file.readlines()
00290 #             for line in temp_arr:
00291 #                 sid, name = line.split('%')
00292 #                 students[sid.strip()] = name.strip()
00293 #         del temp_arr
00294 #
00295 #
00296 #     if lab_type == 'Closed':
00297 #         max_points = 10
00298 #         type_for_name = 'CL'
00299 #     elif lab_type == 'Open':
00300 #         max_points = 20
00301 #         type_for_name = 'OL'
00302 #     else:
00303 #         raise Exception('Unknown lab type')
00304 #
00305 #     if resubmit_num == 1:
00306 #         penalty = 1.0
00307 #     elif resubmit_num == 2:
00308 #         penalty = 0.9
00309 #     elif resubmit_num == 3:
00310 #         penalty = 0.7
00311 #     elif resubmit_num == 4:
00312 #         penalty = 0.5
00313 #     else:
00314 #         penalty = 0.0
00315 #
00316 #     generated_time = QDateTime.currentDateTime().toString(Qt.DefaultLocaleLongDate)
00317 #
00318 #     print('This is ', type_for_name, ' lab, so max points is ', max_points)
00319 #
00320 #     try:
00321 #         shutil.rmtree(dir_name + 'Answers', ignore_errors=True)
00322 #         os.remove(dir_name + "grades.csv")
00323 #         os.remove(dir_name + "grades_for_" + type_for_name + "lab_num.csv")
00324 #     except Exception as e:
00325 #         print('Exception during dir preparetion : ', e)
00326 #
00327 #     dirs = os.walk(dir_name).__next__()[1]
00328 #
00329 #     with open('./answer.top', 'r') as partial_html:
00330 #         top_part = partial_html.readlines()
00331 #
00332 #     with open('./answer.bottom', 'r') as partial_html:
00333 #         bot_part = partial_html.readlines()
00334 #
00335 #     grades = list()
00336 #     for cur_dir in dirs:
00337 #         student_id = cur_dir.split('-')[0]
00338 #         joined_path = os.path.join(dir_name, cur_dir)
00339 #         with open(joined_path + '/grade.txt', 'r') as grade_file:
00340 #             grade = grade_file.readlines()
00341 #
00342 #         grade = int(grade[0].strip())
00343 #         final_score = grade * penalty
00344 #         grades.append((student_id, final_score))
00345 #         create_html_pdf_report(joined_path, students[student_id], cur_dir, grade,
00346 #                                max_points, penalty, final_score, top_part, bot_part, generated_time)
00347 #
00348 #     submitted = [x.split('-')[0] for x in dirs]
00349 #
```

```
00350 #     zeroes = list()
00351 #     for student in students:
00352 #         if student not in submitted:
00353 #             grades.append((student, 0))
00354 #             zeroes.append(student)
00355 #
00356 #     if resubmit_num == 1:
00357 #         for student_id in zeroes:
00358 #             filename = '%s/%s-%s%d-0000000000-returned' % \
00359 #                       (dir_name, student_id, type_for_name, lab_num)
00360 #             create_html_pdf_zero_report(filename+'.html', students[student_id], top_part, bot_part)
00361 #
00362 #     with open(dir_name + '/' + 'grades.csv', 'w') as grades_file:
00363 #         for grade in sorted(grades):
00364 #             grades_file.write("%s, %f \n" % grade)
00365 #
00366 #     os.mkdir(dir_name + '/Answers')
00367 #     files = os.walk(dir_name).__next__()[2]
00368 #     for file in files:
00369 #         if file[-3:] == 'pdf':
00370 #             shutil.move(dir_name + '/' + file, dir_name + '/Answers/' + file)
00371 #
00372 #     print('Done')
00373
00374 #
00375 # def create_a_report(lab_dict):
00376 #
00377 #     create_html_pdf_report2( lab_dict)
00378
00379
```

References convert_to_pdf().
Referenced by create_not_submitted().
Here is the call graph for this function:



Here is the caller graph for this function:



6.4.1.4 create_not_submitted() def generate.create_not_submitted (stud_id, lab_type, lab_num, dir_name)

Definition at line 380 of file generate.py.

```
00380 def create_not_submitted(stud_id, lab_type, lab_num, dir_name):
00381     with open('./answer.top', 'r') as partial_html:
00382         top_part = partial_html.readlines()
00383
00384     with open('./answer.bottom', 'r') as partial_html:
00385         bot_part = partial_html.readlines()
```

```
00386     filename = '%s/%s-%s%d-0000000000-returned' % \
00387                (dir_name, stud_id, lab_type, lab_num)
00388     create_html_pdf_zero_report(filename + '.html', stud_id, top_part, bot_part)
00389
00390
```

References create_html_pdf_zero_report().

Here is the call graph for this function:

6.4.1.5 generate_answers3()

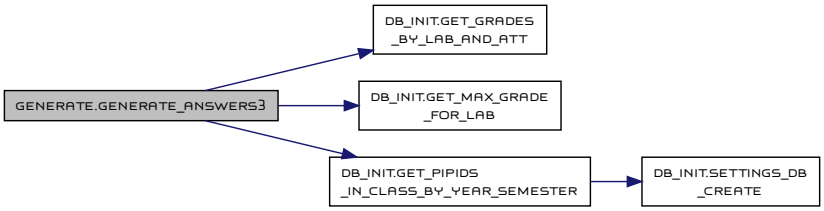
def generate.generate_answers3 (lid, att, year, semester, db_name = './grades.sqlite3')

Definition at line 391 of file generate.py.

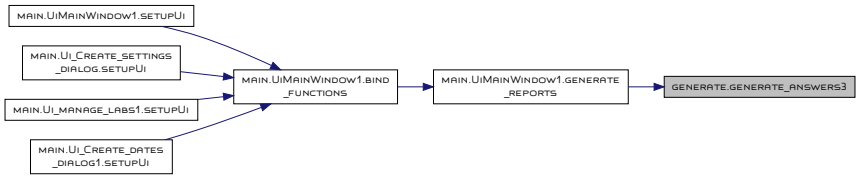
```
00391 def generate_answers3(lid, att, year, semester, db_name='./grades.sqlite3'):
00392     all_ids = get_pipids_in_class_by_year_semester(year, semester)
00393     info_tup, info_desc = get_grades_by_lab_and_att(lid, att)
00394     col_names = [elem[0] for elem in info_desc]
00395     main_list = list()
00396     for tup in info_tup:
00397         a = dict ()
00398         for i, elem in enumerate(tup):
00399             a[col_names[i]] = elem
00400         main_list.append(a)
00401     graded_students = [elem['pipeline_id'] for elem in main_list]
00402     grades = [elem['final_grade'] for elem in main_list]
00403     grade_dict = dict (zip(graded_students, grades))
00404     lab_type = main_list[0]['type']
00405     lab_num = main_list[0]['num']
00406     dir_name = main_list[0]['lab_path']
00407     dir_name = dir_name[:dir_name.rfind('/')]
00408     correctd_lab_type = 'CL' if lab_type == 'Closed' else 'OL'
00409
00410     # for elem in main_list:
00411     #     create_a_report(elem)
00412     #
00413     # for elem in main_list:
00414     #     commit_gen_report(elem['grade_id'])
00415
00416     not_subm_ids = [stud_id for stud_id in all_ids if stud_id not in graded_students]
00417
00418     if len(main_list) + len(not_subm_ids) == 0:
00419         return
00420
00421     ans_dir = os.path.join(dir_name, 'Answers')
00422     if os.path.exists(ans_dir):
00423         shutil.rmtree(ans_dir, ignore_errors=True)
00424     gr_file = os.path.join(dir_name, 'grades.csv')
00425     if os.path.exists(gr_file):
00426         os.remove(gr_file)
00427     gr_long_file = os.path.join(dir_name, "grades_for_{}lab_num.csv".format(correctd_lab_type))
00428     if os.path.exists(gr_long_file):
00429         os.remove(gr_long_file)
00430     files_to_rem = (os.path.join(dir_name, file) for file in (el for el in os.walk(dir_name).__next__()[2] if el[-3:] in ['pdf', 'html']))
00431
00432     with mp.Pool() as pool:
00433         pool.map(os.remove, files_to_rem)
00434         r1 = pool.map_async(create_html_pdf_report2, main_list)
00435         r2 = pool.map_async(commit_gen_report, (elem['grade_id'] for elem in main_list))
00436         if att == 1:
00437             pool.starmap(create_not_submitted, ((stud_id, correctd_lab_type, lab_num, dir_name) for stud_id in not_subm_ids))
00438         r1.wait()
00439         r2.wait()
00440
00441     with open(os.path.join(dir_name, '{}_lab_{}_grades.csv'.format(lab_num, lab_type)), 'w') as grades_file:
00442         grades_file.write("{1} Lab {0}, {1} Lab {0}\n".format(lab_num, lab_type))
```

```
00443         for stud_id in all_ids:
00444             if stud_id not in not_subm_ids:
00445                 grades_file.write("{:s}, {:d}\n".format(stud_id, int(grade_dict[stud_id])))
00446             else:
00447                 grades_file.write("{:s}, {:d}\n".format(stud_id, 0))
00448 
00449 
00450     best_grade_list = get_max_grade_for_lab(lid, year, semester)
00451     with open(os.path.join(dir_name, '{}_lab_{}_grades_best_so_far.csv'.format(lab_num, lab_type)), 'w') as grades_file:
00452         grades_file.write("{1} Lab {0}, {1} Lab {0}\n".format(lab_num, lab_type))
00453         for stud_tup in best_grade_list:
00454             grades_file.write('{}, {}\n'.format(stud_tup[0], stud_tup[1]))
00455 
00456     # for elem in main_list:
00457     #     create_html_pdf_report2(elem)
00458     # for elem in main_list:
00459     #     commit_gen_report(elem['grade_id'])
00460 
00461     # if att == 1:  # we do not form report for second attempt since most people are happy with previous grade
00462         # for stud_id in not_subm_ids:
00463         #     create_not_submitted(stud_id, correctd_lab_type, lab_num, dir_name)
00464 
00465     os.mkdir(os.path.join(dir_name, 'Answers'))
00466     files = os.walk(dir_name).__next__()[2]
00467     for file in files:
00468         if file[-3:] == 'pdf':
00469             shutil.move(os.path.join(dir_name, file), os.path.join(dir_name, 'Answers/{}'.format(file)))
00470 
00471     print('Done')
00472 
00473 
```

References db_init.get_grades_by_lab_and_att(), db_init.get_max_grade_for_lab(), and db_init.get_pipids_in_class_by_year_semester().
Referenced by main.UiMainWindow1.generate_reports().
Here is the call graph for this function:



Here is the caller graph for this function:



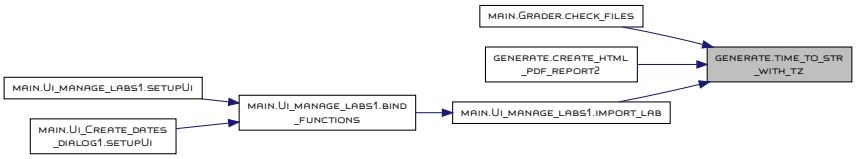
6.4.1.6 time_to_str_with_tz() def generate.time_to_str_with_tz (in_time)

Definition at line 474 of file generate.py.

```
00474 def time_to_str_with_tz(in_time):
00475     return datetime.utcfromtimestamp(in_time).replace(tzinfo=tz.tzutc()).astimezone(tz.tzlocal()).strftime('%m-%d-%Y %H:%M')
00476 # if __name__ == '__main__':
00477 #     # generate_answers(3, 'Open_Lab_3_3', 'Open', 3)
```

Referenced by main.Grader.check_files(), create_html_pdf_report2(), and main.Ui_manage_labs1.import_lab().

Here is the caller graph for this function:



6.5 main Namespace Reference

Classes

- class CircFile
- class Grader
- class SimpleDialog

  Wrapper class for very simple Ok|Cancel dialog.
- class Ui_Create_dates_dialog1
- class Ui_Create_settings_dialog

  Creates window that provides user with convenient way of changing settings that are stored in sqlite3 db.

- class Ui_manage_labs1
- class UiMainWindow1

Functions

- def read_settings (db_name='settings.sqlite3')
- def get_grading_period (lid, cur_only=False)

Variables

- string MAIN_FILE_NAME = ''
- string MAIN_FILE_NAME_OVERRIDE = ''
- string styleData
- app = QtWidgets.QApplication(sys.argv)
- MainWindow = QtWidgets.QMainWindow()
- ui = UiMainWindow1()

6.5.1 Function Documentation

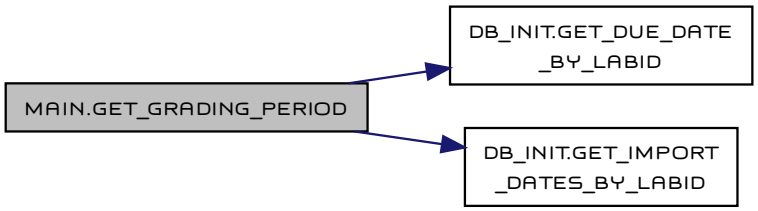
6.5.1.1 get_grading_period() def main.get_grading_period (lid, cur_only = False)

Definition at line 1874 of file main.py.

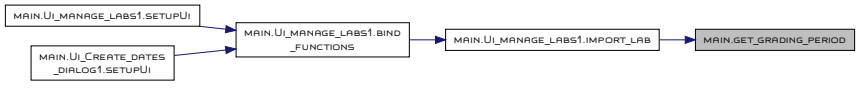
```
01874     import time
01875     # due_timestamps = [int(f.split('_')[2]) for f in due_files]
01876
01877     current_timestamp = int(time.time())
01878     due_timestamps1 = get_due_date_by_labid(lid)
01879     import_timestamps1 = get_import_dates_by_labid(lid)
01880     cur_check = len(due_timestamps1)
01881     for i, ts in enumerate(import_timestamps1):
01882         if ts is None:
01883             cur_check = i
01884             break
01885     i = 0
01886     if cur_check:
01887         while i < len(due_timestamps1) and import_timestamps1[i] is not None and due_timestamps1[i] < current_timestamp and due_timestamps1[i]
       < import_timestamps1[cur_check-1]:
01888             i += 1
01889
01890     if cur_only:  # neede for CLA2-2
01891         i = max(0, i-1)
01892
01893     if i == 0:
01894         from_time = 0
01895         to_time = due_timestamps1[i]
```

```
01896     elif i > len(due_timestamps1)-1:
01897         from_time = due_timestamps1[i-1]
01898         to_time = int(time.time())
01899     else:
01900         from_time = due_timestamps1[i - 1]
01901         to_time = due_timestamps1[i]
01902 
01903     cur_check_num = i+1
01904     # cur_check += 1
01905 
01906 
01907     #
01908     # check_files = [int(f.split('_')[2]) for f in os.listdir(dir) if 'check_' in f]
01909     # if len(check_files) > 0:
01910     #     if len(check_files) >= 4:
01911     #         cur_check_num = 0
01912     #         from_time = 0
01913     #         to_time = 0
01914     #     else:
01915     #         cur_check_num = len(check_files) + 1           # 1 + 1
01916     #         from_time = due_timestamps[cur_check_num - 2]  # 0 => after first due date
01917     #         to_time = due_timestamps[cur_check_num - 1]    # 1 => before second due date
01918     # else:
01919     #     from_time = 0
01920     #     to_time = due_timestamps[0]
01921     #     cur_check_num = 1
01922 
01923     return cur_check_num, from_time, to_time, current_timestamp
01924 
01925 
01926 class Ui_Create_dates_dialog1(Ui_Create_dates_dialog):
01927 
```

References db_init.get_due_date_by_labid(), and db_init.get_import_dates_by_labid().
Referenced by main.Ui_manage_labs1.import_lab().
Here is the call graph for this function:



Here is the caller graph for this function:



6.5.1.2 read_settings() def main.read_settings (db_name = 'settings.sqlite3')

Definition at line 63 of file main.py.

```
00063     """
00064     import os.path
00065 
00066     if os.path.exists(db_name):
00067         with lite.connect(db_name) as con:
00068             cur = con.cursor()
```

```
00069             try:
00070                 cur.execute('SELECT * FROM PATHS')
00071                 result = cur.fetchone()
00072                 for row in result:
00073                     print(row)
00074                 logisim_path = result[0][0]
00075                 working_dir = result[0][1]
00076                 # since import is not implemented - ignore import path: import_path = result[0][2]
00077                 return logisim_path, working_dir
00078             except Exception as e:
00079                 print('Was not able to get results from settings DB: ', e)
00080     return None
00081
00082
```

6.5.2 Variable Documentation

6.5.2.1 app

`main.app = QtWidgets.QApplication(sys.argv)`

Definition at line 2012 of file main.py.

6.5.2.2 MAIN_FILE_NAME

`string main.MAIN_FILE_NAME = ''`

Definition at line 38 of file main.py.

6.5.2.3 MAIN_FILE_NAME_OVERRIDE

`string main.MAIN_FILE_NAME_OVERRIDE = ''`

Definition at line 39 of file main.py.

6.5.2.4 MainWindow

`main.MainWindow = QtWidgets.QMainWindow()`

Definition at line 2013 of file main.py.

6.5.2.5 styleData

`string main.styleData`

Initial value:

```
00001 =  """
00002 /* https://stackoverflow.com/questions/22332106/python-qtgui-qprogressbar-color */
00003 QProgressBar
00004 {
00005     border: 1px solid grey;
00006     border-radius: 5px;
00007     text-align: center;
00008     font-weight: bold;
00009 }
00010 QProgressBar::chunk
00011 {
00012     background-color: #d7801a;
00013     width: 2.15px;
00014     margin: 0.5px;
00015 }
00016 """
```

Definition at line 41 of file main.py.

6.5.2.6 ui

`main.ui = UiMainWindow1()`

Definition at line 2014 of file main.py.

6.6 main_window Namespace Reference

Classes

- class Ui_mainWindow

6.7 manage_labs Namespace Reference

Classes

- class Ui_manage_labs

6.8 mptest_mp Namespace Reference

Functions

- def f (x, y)

Variables

- list b = [elem for elem in range(10)]
- int c = 10
- res = pool.starmap_async(f, ((elem, c) for elem in b))
- int a = 55

6.8.1 Function Documentation

6.8.1.1 f() def mptest_mp.f (x, y)

Definition at line 5 of file mptest_mp.py.

```
00005 def f(x, y):
00006     time.sleep(random.randint(1, 4))
00007     print(x, y)
00008     return x
00009
```

6.8.2 Variable Documentation

6.8.2.1 a int mptest_mp.a = 55

Definition at line 19 of file mptest_mp.py.

6.8.2.2 b list mptest_mp.b = [elem for elem in range(10)]

Definition at line 11 of file mptest_mp.py.

6.8.2.3 c int mptest_mp.c = 10

Definition at line 12 of file mptest_mp.py.

6.8.2.4 res mptest_mp.res = pool.starmap_async(f, ((elem, c) for elem in b))

Definition at line 15 of file mptest_mp.py.

6.9 qt_class_improvements Namespace Reference

Classes

- class BetterLineEdit
- class BetterPlainTextEdit

6.10 settings Namespace Reference

Classes

- class Ui_Settings

6.11 simple_dialog Namespace Reference

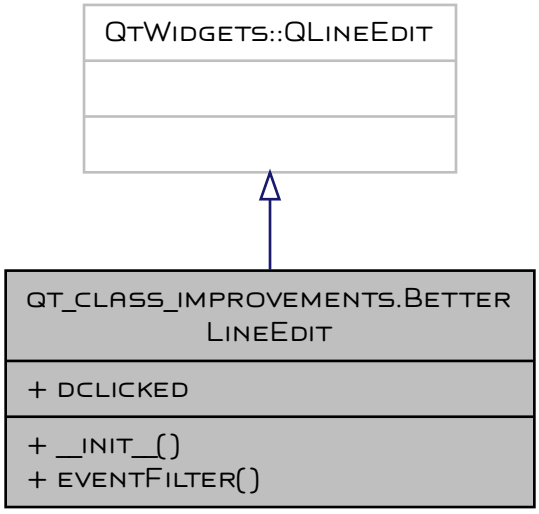
Classes

- class Ui_Dialog

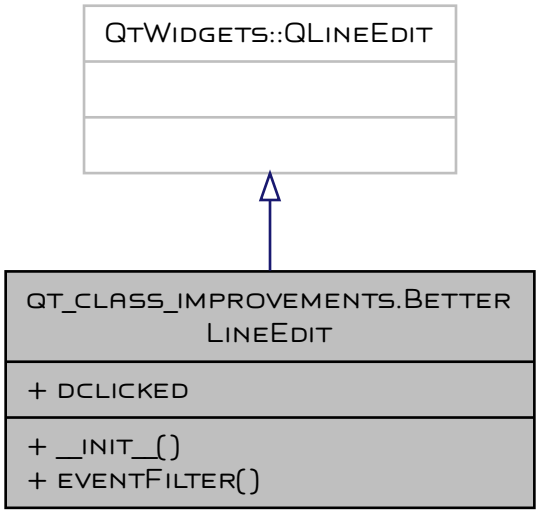
7 Class Documentation

7.1 qt_class_improvements.BetterLineEdit Class Reference

Inheritance diagram for qt_class_improvements.BetterLineEdit:



Collaboration diagram for qt_class_improvements.BetterLineEdit:



Public Member Functions

- def __init__ (self, *args, **kwargs)
- def eventFilter (self, obj, event)

  typical way to add event handler

Static Public Attributes

- dclicked = QtCore.pyqtSignal()

7.1.1 Detailed Description

Definition at line 11 of file qt_class_improvements.py.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 __init__() def qt_class_improvements.BetterLineEdit.__init__ (self, * args, ** kwargs)

Definition at line 14 of file qt_class_improvements.py.

```
00014     def __init__(self, *args, **kwargs):
00015         QtWidgets.QLineEdit.__init__(self, *args, **kwargs)
00016 
00017         self.installEventFilter(self)
00018 
```

7.1.3 Member Function Documentation

7.1.3.1 eventFilter() def qt_class_improvements.BetterLineEdit.eventFilter (self, obj, event)

typical way to add event handler

Definition at line 20 of file qt_class_improvements.py.

```
00020         """ typical way to add event handler """
00021         if event.type() == QtCore.QEvent.MouseButtonDblClick:
00022             self.dclicked.emit()
00023         return False
00024 
00025 
00026 class BetterPlainTextEdit(QtWidgets.QPlainTextEdit):
00027     """
00028     Overloaded QPlainTextEdit to track focus out.
00029     Needed to implement autosaving of user answer.
```

References qt_class_improvements.BetterLineEdit.dclicked.

7.1.4 Member Data Documentation

7.1.4.1 dclicked qt_class_improvements.BetterLineEdit.dclicked = QtCore.pyqtSignal() [static]

Definition at line 12 of file qt_class_improvements.py.

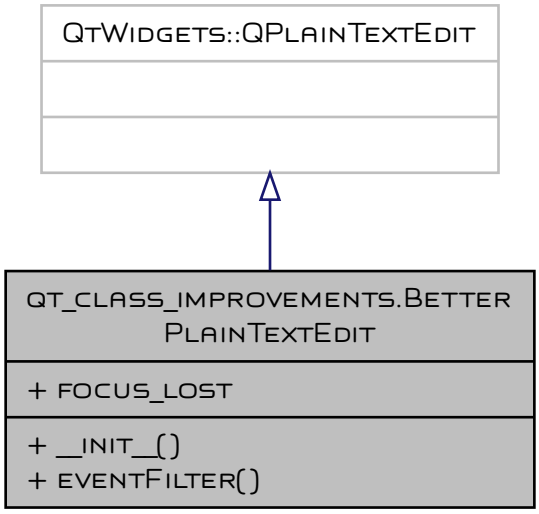
Referenced by qt_class_improvements.BetterLineEdit.eventFilter().

The documentation for this class was generated from the following file:

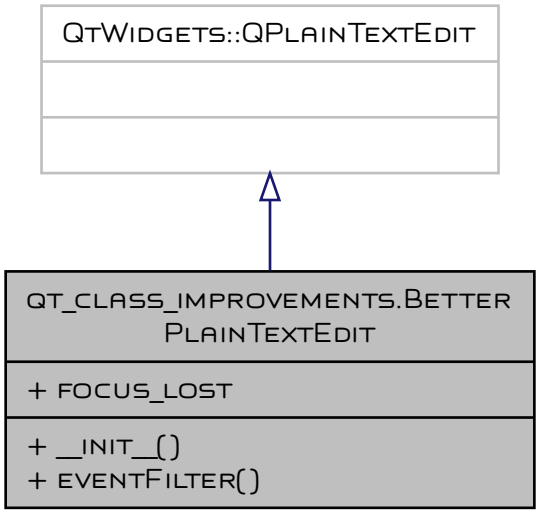
- qt_class_improvements.py

7.2 qt_class_improvements.BetterPlainTextEdit Class Reference

Inheritance diagram for qt_class_improvements.BetterPlainTextEdit:



Collaboration diagram for qt_class_improvements.BetterPlainTextEdit:



Public Member Functions

- def __init__ (self, *args, **kwargs)
- def eventFilter (self, obj, event)

typical way to add event handler

Static Public Attributes

- focus_lost = QtCore.pyqtSignal()

7.2.1 Detailed Description

Definition at line 30 of file qt_class_improvements.py.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 __init__() def qt_class_improvements.BetterPlainTextEdit.__init__ (self, * args, ** kwargs)

Definition at line 33 of file qt_class_improvements.py.

```
00033     def __init__(self, *args, **kwargs):
00034         QtWidgets.QPlainTextEdit.__init__(self, *args, **kwargs)
00035
00036         self.installEventFilter(self)
00037
```

7.2.3 Member Function Documentation

7.2.3.1 eventFilter() def qt_class_improvements.BetterPlainTextEdit.eventFilter (self, obj, event)

typical way to add event handler

Definition at line 39 of file qt_class_improvements.py.

```
00039         """ typical way to add event handler """
00040         if event.type() == QtCore.QEvent.FocusOut:
00041             self.focus_lost.emit()
00042         return False
```

References qt_class_improvements.BetterPlainTextEdit.focus_lost.

7.2.4 Member Data Documentation

7.2.4.1 focus_lost qt_class_improvements.BetterPlainTextEdit.focus_lost = QtCore.pyqtSignal() [static]

Definition at line 31 of file qt_class_improvements.py.

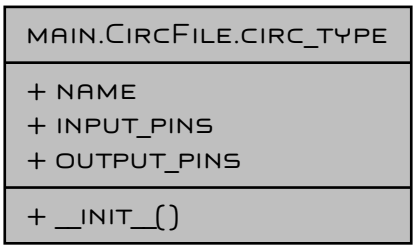
Referenced by qt_class_improvements.BetterPlainTextEdit.eventFilter().

The documentation for this class was generated from the following file:

- qt_class_improvements.py

7.3 main.CircFile.circ_type Class Reference

Collaboration diagram for main.CircFile.circ_type:



Public Member Functions

- def __init__ (self, name)

Public Attributes

- name
- input_pins
- output_pins

7.3.1 Detailed Description

Definition at line 85 of file main.py.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 __init__() def main.CircFile.circ_type.__init__ (self, name)

Definition at line 86 of file main.py.

```
00086         def __init__(self, name):
00087             self.name = name
00088             self.input_pins = list()
00089             self.output_pins = list()
00090
00091     class PinType:
```

7.3.3 Member Data Documentation

7.3.3.1 input_pins main.CircFile.circ_type.input_pins

Definition at line 88 of file main.py.

7.3.3.2 name main.CircFile.circ_type.name

Definition at line 87 of file main.py.

7.3.3.3 output_pins main.CircFile.circ_type.output_pins

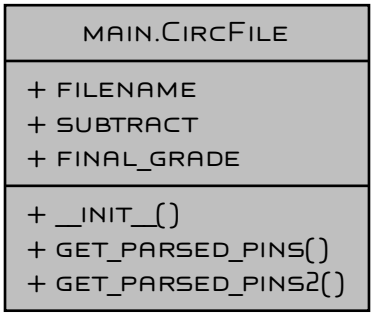
Definition at line 89 of file main.py.

The documentation for this class was generated from the following file:

- main.py

7.4 main.CircFile Class Reference

Collaboration diagram for main.CircFile:



Classes

- class circ_type
- class PinType

Public Member Functions

- def __init__ (self, filename)
- def get_parsed_pins (self)

  :return:
- def get_parsed_pins2 (self, what_to_grade)

Public Attributes

- filename
- subtract
- final_grade

7.4.1 Detailed Description

Definition at line 83 of file main.py.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 __init__() def main.CircFile.__init__ (self, filename)

Definition at line 97 of file main.py.

```
00097     def __init__(self, filename):
00098         self.filename = filename
00099         self.subtract = 0
00100         self.final_grade = 10
00101         self.__all_circuits = list()
00102
```

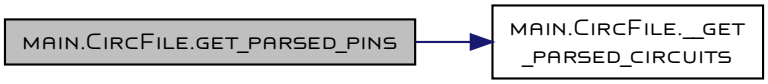
7.4.3 Member Function Documentation

7.4.3.1 get_parsed_pins() def main.CircFile.get_parsed_pins (self)

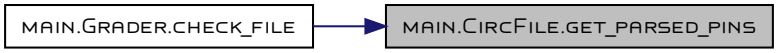
:return:
Definition at line 124 of file main.py.

```
00124         arr = self.__all_circuits
00125         all_pins = list()
00126         for elem in arr:
00127             pins = list()
00128             for child in elem.findall('comp'):
00129                 if child.get('name') == 'Pin':
00130                     pins.append(child)
00131                     # print(child.tag, child.attrib)
00132             all_pins.append(pins)
00133 
00134         clean_data = list()
00135         if all_pins:
00136             for pins in all_pins:  # Although this looks like an error - it is not,
00137                 # there is only one iteration. This code will be extended later
00138                 # as I had in my older scripts to grade all PLDs.
00139                 clean_data = list()
00140                 for pin in pins:
00141                     name = '0'
00142                     io_type = '0'
00143                     facing = ''
00144                     for elem in list(pin):
00145                         if elem.get('name') in ['output', 'input', 'tristate']:
00146                             io_type = elem.get('name')
00147                         elif elem.get('name') == 'label':
00148                             name = elem.get('val')
00149                         elif elem.get('name') == 'facing':
00150                             facing = elem.get('val')
00151                     clean_data.append(self.PinType(name, io_type, facing))
00152         else:
00153             raise Exception('Error in pin parsing(all_pins)')
00154 
00155         output_pins = list()
00156         input_pins = list()
00157         other_pins = list()
00158 
00159         if clean_data:
00160             for pin in clean_data:
00161                 if pin.type == 'output':
00162                     output_pins.append(pin)
00163                 elif pin.type == 'input' or pin.type == 'tristate':
00164                     input_pins.append(pin)
00165                 else:
00166                     other_pins.append(pin)
00167         else:
00168             raise Exception('Error in pin parsing(clean data)')
00169 
00170         return input_pins, output_pins, other_pins
00171 
00172 
00173     def get_parsed_pins2(self, what_to_grade):
00174 
```

References main.CircFile.__all_circuits, and main.CircFile.__get_parsed_circuits().
Referenced by main.Grader.check_file().
Here is the call graph for this function:



Here is the caller graph for this function:



7.4.3.2 get_parsed_pins2() def main.CircFile.get_parsed_pins2 (self, what_to_grade)

Definition at line 175 of file main.py.

```
00175         tree = ET.parse(self.filename)
00176         root = tree.getroot()
00177         arr=list()
00178         for child in root:
00179             # print(child.tag)
00180             if child.tag == 'circuit':
00181                 arr.append(child)
00182             # if child.attrib["name"] == what_to_grade:
00183             #     a = child
00184             #     b =1
00185 
00186         all_circs = list()
00187         good_arr = list()
00188         for node in arr:
00189             if node.get('name').upper() in what_to_grade:
00190                 good_arr.append(node)
00191                 circ_instance = self.circ_type(node.get('name'))
00192                 all_circs.append(circ_instance)
00193                 # print(list(node)[0].items()[0][1])
00194 
00195         all_pins = list()
00196         for elem in good_arr:
00197             pins = list()
00198             for child in elem.findall('comp'):
00199                 if child.get('name') == 'Pin':
00200                     pins.append(child)
00201                     # print(child.tag, child.attrib)
00202             all_pins.append(pins)
00203 
00204 
00205         clean_all_pins = list()
00206         for pins in all_pins:
00207             clean_data = list()
00208             for pin in pins:
00209                 name = '0'
00210                 type = '0'
00211                 for elem in list(pin):
00212                     if elem.get('name') in ['output', 'input', 'tristate']:
00213                         type = elem.get('name')
00214                     elif elem.get('name') == 'label':
00215                         name = elem.get('val')
00216                 clean_data.append(self.PinType(name, type))
00217             clean_all_pins.append(clean_data)
00218         for i in range(len(clean_all_pins)):
00219             for pin in clean_all_pins[i]:
00220                 if pin.type == 'output':
00221                     all_circs[i].output_pins.append(pin.name)
00222                 else:
00223                     all_circs[i].input_pins.append(pin.name)
00224         return all_circs
00225 
00226 
00227 class Grader:
00228     def __init__(self, working_directory, grader='Ivan'):
```

References main.CircFile.filename.

7.4.4 Member Data Documentation

7.4.4.1 filename main.CircFile.filename

Definition at line 98 of file main.py.
Referenced by main.CircFile.get_parsed_pins2().

7.4.4.2 final_grade main.CircFile.final_grade

Definition at line 100 of file main.py.
Referenced by main.Grader.check_circ_exist(), main.Grader.check_file(), main.Grader.check_wrong(), main.Grader.read_resp(), main.Grader.read_resp2(), main.Grader.save_all2(), and main.Grader.save_grade().

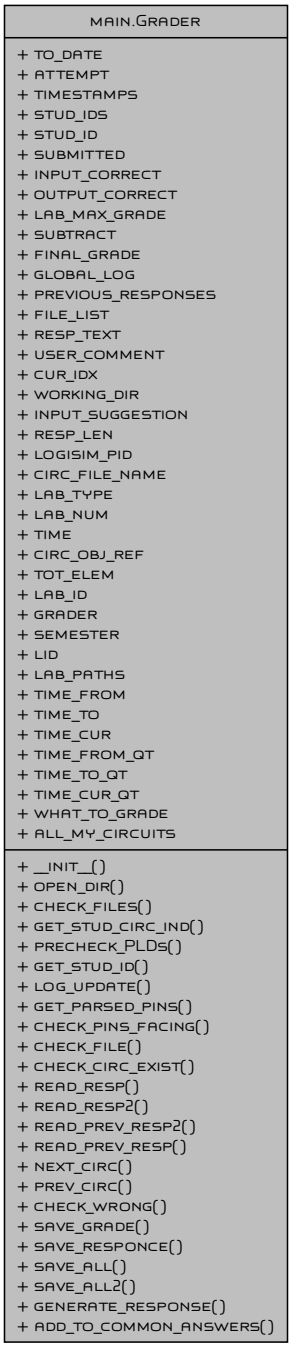
7.4.4.3 subtract main.CircFile.subtract

Definition at line 99 of file main.py.
Referenced by main.Grader.check_file(), and main.Grader.get_parsed_pins().
The documentation for this class was generated from the following file:

- main.py

7.5 main.Grader Class Reference

Collaboration diagram for main.Grader:



Public Member Functions

- def __init__ (self, working_directory, grader='Ivan')
- def open_dir (self)
- def check_files (self)

- def get_stud_circ_ind (self, student_circuits, circ_to_grade)
- def precheck_PLDs (self, stud_ind)
- def get_stud_id (self)
- def log_update (self, log_event)
- def get_parsed_pins (self)
- def check_pins_facing (self, pins, corr_facing)
- def check_file (self)
- def check_circ_exist (self)
- def read_resp (self)
- def read_resp2 (self)
- def read_prev_resp2 (self)
- def read_prev_resp (self)
- def next_circ (self)
- def prev_circ (self)
- def check_wrong (self)
- def save_grade (self)
- def save_responce (self)
- def save_all (self)
- def save_all2 (self)
- def generate_response (self)
- def add_to_common_answers (self, typed)

Public Attributes

- to_date
- attempt
- timestamps
- stud_ids
- stud_id
- submitted
- input_correct
- output_correct
- lab_max_grade
- subtract
- final_grade
- global_log
- previous_responses
- file_list
- resp_text
- user_comment
- cur_idx
- working_dir
- input_suggestion
- resp_len
- logisim_pid
- circ_file_name
- lab_type
- lab_num
- time
- circ_obj_ref
- tot_elem
- lab_id
- grader
- semester
- lid
- lab_paths
- time_from
- time_to
- time_cur
- time_from_qt
- time_to_qt
- time_cur_qt
- what_to_grade
- all_my_circuits

7.5.1 Detailed Description

Definition at line 229 of file main.py.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 __init__() def main.Grader.__init__ (self, working_directory, grader = 'Ivan')

```
Definition at line 230 of file main.py.
00230         self.to_date = 0
00231         self.attempt = 0
00232         self.timestamps = list()
00233         self.stud_ids = list()
00234         self.stud_id = ”
00235         self.submitted = 0
00236         self.input_correct = False
00237         self.output_correct = False
00238         self.lab_max_grade = 0
00239         self.subtract = 0
00240         self.__wrong_clicked = False
00241         self.final_grade = 0
00242         self.__possible_answers_dict = {}
00243         self.global_log = ”
00244         self.previous_responses = ”
00245         self.__message_to_all = ”
00246         self.__graded_idlist = list()
00247         self.file_list = list()
00248         self.resp_text = 'I did not find any errors. Good job!\n'
00249         self.user_comment = ”
00250         self.cur_idx = 0
00251         self.working_dir = working_directory
00252         self.input_suggestion = set(”,)
00253         self.resp_len = 38
00254         self.logisim_pid = -1
00255         self.circ_file_name = MAIN_FILE_NAME
00256         self.lab_type = ”
00257         self.lab_num = 0
00258         self.time = 0
00259         self.circ_obj_ref = None
00260         self.tot_elem = 0
00261         self.lab_id = ”
00262         self.grader = grader
00263
00264     def open_dir(self):
00265         """
References main.Grader.__from_date.
```

7.5.3 Member Function Documentation

7.5.3.1 add_to_common_answers() def main.Grader.add_to_common_answers (self, typed)

```
Definition at line 712 of file main.py.
00712
00713
00714 class UiMainWindow1(Ui_mainWindow):
00715     """
References main.Grader.input_suggestion.
```

7.5.3.2 check_circ_exist() def main.Grader.check_circ_exist (self)

```
Definition at line 519 of file main.py.
00519             self.resp_text = 'File was not found'
00520             file_found = os.listdir(self.file_list[self.cur_idx])
00521             potential_files = list()
00522             for file in file_found:
00523                 if file not in ['grade.txt', 'penalty.txt', 'responce.txt', 'tech_info.txt', ]:
00524                     potential_files.append(file)
00525             if potential_files:
00526                 self.resp_text += '\nNext files|folders were found:\n'
00527             for file in potential_files:
00528                 if os.path.isdir(self.file_list[self.cur_idx] + '/' + file):
00529                     self.resp_text += file + ' - directory.\n'
00530                 else:
00531                     self.resp_text += file + ' - regular file.\n'
00532             self.resp_len = len(self.resp_text)
00533             self.final_grade = 0
00534             return False
00535         return True
00536
00537     def read_resp(self):
```

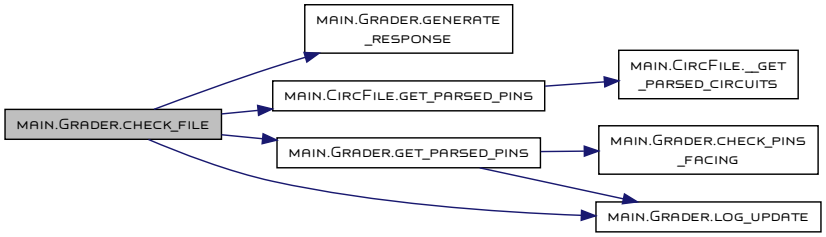
```
00538         """
```
References main.Grader.circ_file_name, main.Grader.cur_idx, main.Grader.file_list, main.CircFile.final_grade, main.Grader.final_grade, main.Grader.resp_len, and main.Grader.resp_text.

7.5.3.3 check_file() def main.Grader.check_file (self)

Definition at line 498 of file main.py.

```
00498
00499         circ_obj = CircFile(file)
00500         self.circ_obj_ref = circ_obj
00501         self.subtract = 0
00502         try:
00503             self.get_parsed_pins()
00504
00505             self.log_update('Pins successfully parsed.')
00506             self.final_grade = self.lab_max_grade - self.subtract
00507             self.generate_response()
00508         except Exception as e:
00509             print(e)
00510             self.log_update(sys.exc_info()[0])
00511
00512     def check_circ_exist(self):
00513         """
```
References main.Grader.circ_obj_ref, main.Grader.cur_idx, main.Grader.file_list, main.CircFile.final_grade, main.Grader.final_grade, main.Grader.generate_response(), main.CircFile.get_parsed_pins(), main.Grader.get_parsed_pins(), main.Grader.lab_max_grade, main.Grader.log_update(), main.CircFile.subtract, and main.Grader.subtract.

Here is the call graph for this function:

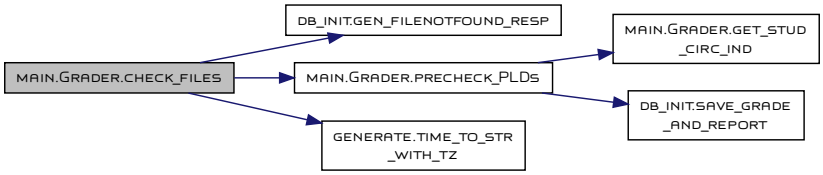


7.5.3.4 check_files() def main.Grader.check_files (self)

Definition at line 348 of file main.py.

```
00348         good_ids = list()
00349         good_sids = list()
00350         good_tss = list()
00351
00352         for i, stud_path in enumerate(self.lab_paths):
00353             cur_path = os.path.join(stud_path, self.circ_file_name)
00354             if os.path.exists(cur_path):
00355                 paths_with_files_list.append(stud_path)
00356                 good_ids.append(self.grade_ids[i])
00357                 good_sids.append(self.stud_ids[i])
00358                 good_tss.append(self.timestamps[i])
00359                 if self.lab_num > 8 and self.lab_type == 'Closed':
00360                     self.precheck_PLDs(i)
00361             else:
00362                 if self.attempt > 1:
00363                     next_date = time_to_str_with_tz(self.time_to + self.time_to - self.time_from)
00364                 else:
00365                     next_date = time_to_str_with_tz(self.time_to + 604800)  # 604800 - one week in unix time, this line needs corrections for
      case when you skip a week
00366                 gen_filenotfound_resp(self.grade_ids[i], stud_path, self.circ_file_name, self.grader, self.attempt, next_date)
00367         # self.grade_ids = good_ids
00368         # self.stud_ids = good_sids
00369         # self.timestamps = good_tss
00370         return good_tss, good_sids, good_ids, paths_with_files_list
00371
00372     def get_stud_circ_ind(self, student_circuits, circ_to_grade):
00373         for stud_circ in student_circuits:
```

References main.Grader.attempt, main.Grader.circ_file_name, db_init.gen_filenotfound_resp(), main.Grader.grader, main.Grader.lab_num, main.Grader.lab_paths, main.Grader.lab_type, main.Grader.precheck_PLDs(), main.Grader.stud_ids, main.Grader.time_from, main.Grader.time_to, generate.time_to_str_with_tz(), and main.Grader.timestamps.
Here is the call graph for this function:



7.5.3.5 check_pins_facing()

```
def main.Grader.check_pins_facing (
                self,
                pins,
                corr_facing )
```

Definition at line 487 of file main.py.

```
00487             if pin.facing != corr_facing and pin.facing != '':
00488                 return False
00489         return True
00490
00491     def check_file(self):
00492         """
```

Referenced by main.Grader.get_parsed_pins().
Here is the caller graph for this function:

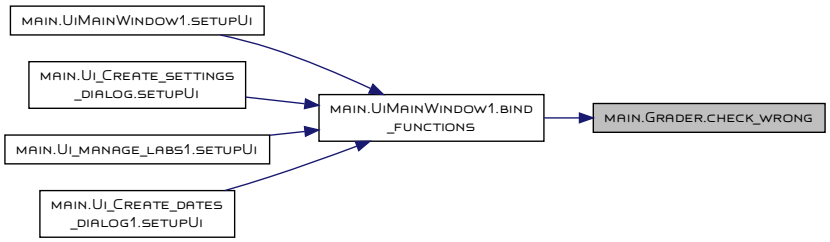
7.5.3.6 check_wrong()

```
def main.Grader.check_wrong (
                self )
```

Definition at line 643 of file main.py.

```
00643         self.resp_text = 'your lab was marked as wrong. You should fix errors listed below and resubmit it.'
00644         self.resp_len = len(self.resp_text)
00645
00646     def save_grade(self):
00647         """
```

References main.CircFile.final_grade, main.Grader.final_grade, main.Grader.resp_len, and main.Grader.resp_text.
Referenced by main.UiMainWindow1.bind_functions().
Here is the caller graph for this function:



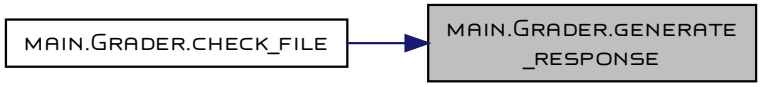
7.5.3.7 generate_response() def main.Grader.generate_response (self)

```
Definition at line 693 of file main.py.
00693         self.user_comment = ''
00694         if self.input_correct and self.output_correct:
00695             self.resp_text = 'I did not find any errors. Good job!'
00696         else:
00697             if not self.input_correct:
00698                 self.resp_text += 'Your input pins have wrong orientation.\n'
00699 
00700             if not self.output_correct:
00701                 self.resp_text += 'Your output pins have wrong orientation.\n'
00702         self.resp_len = len(self.resp_text)
00703 
00704     def add_to_common_answers(self, typed):
00705         """
```

References main.Grader.input_correct, main.Grader.output_correct, main.Grader.resp_len, main.Grader.resp_text, and main.Grader.user_comment.

Referenced by main.Grader.check_file().

Here is the caller graph for this function:



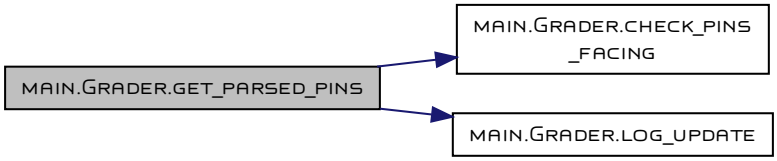
7.5.3.8 get_parsed_pins() def main.Grader.get_parsed_pins (self)

```
Definition at line 459 of file main.py.
00459             input_pins, output_pins, other_pins = self.circ_obj_ref.get_parsed_pins()
00460             if other_pins:
00461                 self.log_update('I was not able to recognize ' + str(len(other_pins)) + " pins.")
00462             self.input_correct = True
00463             self.output_correct = True
00464             if not self.check_pins_facing(pins=input_pins, corr_facing='east'):
00465                 self.subtract += 1
00466                 self.input_correct = False
00467             if not self.check_pins_facing(pins=output_pins, corr_facing='west'):
00468                 self.subtract += 1
00469                 self.output_correct = False
00470         except Exception as e:  # TODO check for FileNotFoundError and assign ZERO
00471             print(e)
00472             # self.log_update(sys.exc_info()[0])
00473             # print(sys.exc_info()[0])
00474             raise
00475         # self.log_update('Done checking: ' + self.filename)
00476 
00477 
00478     # noinspection PyMethodMayBeStatic
00479     def check_pins_facing(self, pins, corr_facing):
```

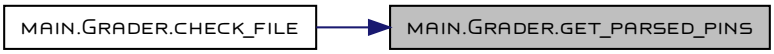
References main.Grader.check_pins_facing(), main.Grader.circ_obj_ref, main.Grader.input_correct, main.Grader.log_update(), main.Grader.output_correct, main.CircFile.subtract, and main.Grader.subtract.

Referenced by main.Grader.check_file().

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.9 get_stud_circ_ind() def main.Grader.get_stud_circ_ind (self, student_circuits, circ_to_grade)

Definition at line 374 of file main.py.

```
00374             if stud_circ.name.upper() == circ_to_grade.upper():
00375                 return student_circuits.index(stud_circ)
00376         for stud_circ in student_circuits:
00377             print(stud_circ.name.upper())
00378         return -1
00379
00380     def precheck_PLDs(self, stud_ind):
00381         file = os.path.join(self.lab_paths[stud_ind], self.circ_file_name)
```

Referenced by main.Grader.precheck_PLDs().

Here is the caller graph for this function:

7.5.3.10 get_stud_id() def main.Grader.get_stud_id (self)

Definition at line 443 of file main.py.

```
00443
00444     def log_update(self, log_event):
00445         """
```

References main.Grader.stud_id.

7.5.3.11 log_update() def main.Grader.log_update (self, log_event)

Definition at line 452 of file main.py.

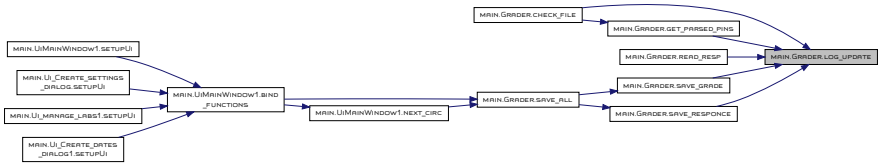
```
00452
```

```
00453     def get_parsed_pins(self):
00454         """
```

References main.Grader.global_log, and main.Grader.stud_id.

Referenced by main.Grader.check_file(), main.Grader.get_parsed_pins(), main.Grader.read_resp(), main.Grader.save_grade(), and main.Grader.save_responce().

Here is the caller graph for this function:



7.5.3.12 next_circ()

```
def main.Grader.next_circ (
        self )
```

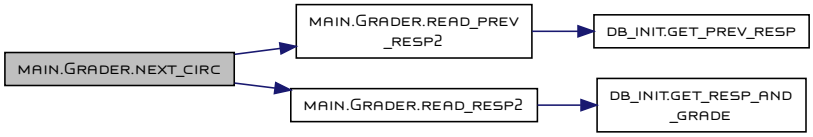
Definition at line 601 of file main.py.

```
00601         # self.check_file(self.cur_idx)
00602         self.user_comment = ''
00603         graded = self.read_resp2()
00604         # if graded:
00605         self.read_prev_resp2()
00606         # if self.check_circ_exist():
00607         #     self.read_resp()
00608         self.stud_id = self.stud_ids[self.cur_idx]
00609         # try:
00610         #     self.read_prev_resp()
00611         # except Exception as e:
00612         #     print('Error during attempt to read prev resp when opening next circuit: ', e)
00613         #     # TODO add handler
00614         return self.cur_idx
00615 
00616     def prev_circ(self):
00617         """
```

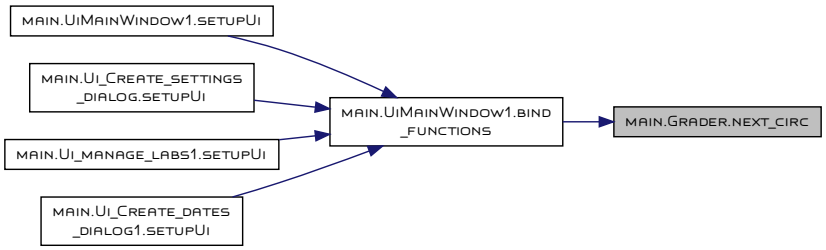
References main.Grader.cur_idx, main.Grader.read_prev_resp2(), main.Grader.read_resp2(), main.Grader.stud_id, main.Grader.stud_ids, and main.Grader.user_comment.

Referenced by main.UiMainWindow1.bind_functions().

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.13 open_dir() def main.Grader.open_dir (self)

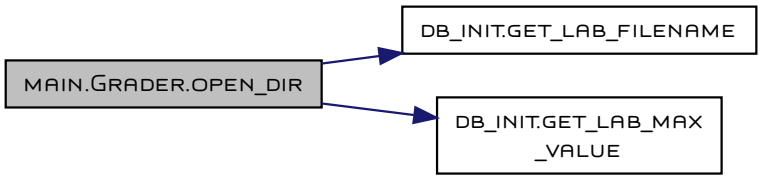
Definition at line 270 of file main.py.

```
00270         # if len(self.working_directory) < 3:
00271         #     wdir = './'
00272         # else:
00273         #     wdir = self.working_directory
00274 
00275 
00276         root, dirs, files = os.walk(self.working_dir).__next__()
00277         files.sort()
00278         # check_file = files[0]  # not used at this time
00279         # if len(files) < 1:
00280         #     raise Exception("No due files ? Extra files in working directory ?")
00281         # due_file = files[1] # TODO: change this to a better design. - Already changed
00282 
00283         self.lab_type = self.working_dir.split('/')[-2].split('_')[0]
00284         self.lab_num = int(self.working_dir.split('/')[-2].split('_')[2])
00285         self.attempt = int(self.working_dir.split('/')[-2].split('_')[3])
00286 
00287         if self.lab_type == 'Closed':
00288             self.lab_id = 'CLA{}'.format(self.lab_num)
00289             # self.lab_max_grade = 10
00290         else:  # Open
00291             # self.lab_max_grade = 20
00292             self.lab_id = 'OLA{}'.format(self.lab_num)
00293 
00294         self.lab_max_grade = get_lab_max_value(self.lab_id)
00295 
00296         # self.time = int(due_file[6:])
00297 
00298         # dirs.sort()  # sort list of submitted labs
00299         # if dirs[0] == 'Answers':
00300         #     dirs.pop(0)
00301 
00302         self.circ_file_name = get_lab_filename(self.lab_id)[0]
00303         self.year, self.semester = self.working_dir.split('/')[-3].split('_')
00304         self.lid = get_labid_in_schedule(get_lab_id(self.lab_type, self.lab_num), self.year, self.semester)
00305         self.timestamps, self.stud_ids, self.grade_ids, self.lab_paths = get_empty_grades_by_lid(self.lid, self.attempt)
00306 
00307         atime = get_grading_period(self.lid, cur_only=True)
00308         self.time_from = atime[1]
00309         self.time_to = atime[2]
00310         self.time_cur = atime[3]
00311 
00312         self.time_from_qt = QDateTime.fromSecsSinceEpoch(self.time_from)
00313         self.time_to_qt = QDateTime.fromSecsSinceEpoch(self.time_to)
00314         self.time_cur_qt = QDateTime.fromSecsSinceEpoch(self.time_cur)
00315 
00316         if self.lab_num > 8 and self.lab_type == 'Closed':
00317             if self.lab_num == 9:
00318                 self.what_to_grade = ['PC_BUS', 'AR_LD', 'PC_LD', 'PC_INC', 'DR_LD', 'DR_BUS']
00319             elif self.lab_num == 10:
00320                 self.what_to_grade = ["R_LD", "R_BUS", "S_LD", "ACC_CLR", "ACC_LD", "ACC_BUS", "ALU_SEL"]
00321             elif self.lab_num == 11:
00322                 self.what_to_grade = ["Z_LD", "OUTR_LD", "RAM_RW", "RAM_EN", "IR_LD", "SC_CLR"]
00323             circ = CircFile('/home/vanya/Documents/3130_labs/2018_2/PLDs.circ')
```

```
00324             self.all_my_circuits = circ.get_parsed_pins2(self.what_to_grade)
00325
00326         if self.lab_paths is not None and len(self.lab_paths) > 0:
00327             self.timestamps, self.stud_ids, self.grade_ids, self.lab_paths = self.check_files()
00328
00329         if self.lab_paths is None or len(self.lab_paths) == 0:  # if there are no ungraded labs - display all labs
00330             self.timestamps, self.stud_ids, self.grade_ids, self.lab_paths = get_all_grades_by_lid(self.lid, self.attempt)
00331
00332         # self.grades = [self.lab_max_grade]*len(self.grade_ids)
00333         # self.stud_ids = dirs
00334         # self.stud_ids = list()
00335         # self.timestamps = list()
00336         # # directory_list = list()
00337         # for name in dirs:
00338         #     self.file_list.append(os.path.join(root, name))
00339         #     temp_arr = name.split('-')
00340         #     self.stud_ids.append(temp_arr[0])
00341         #     self.timestamps.append(int(temp_arr[2]))
00342
00343         # for file in self.file_list:
00344         #     print(file)
00345
00346     def check_files(self):
00347         paths_with_files_list = list()
```

References main.Grader.attempt, main.Grader.circ_file_name, db_init.get_lab_filename(), db_init.get_lab_max_value(), main.Grader.lab_id, main.Grader.lab_max_grade, main.Grader.lab_num, main.Grader.lab_type, and main.Grader.working_dir.

Here is the call graph for this function:



7.5.3.14 precheck_PLDs() def main.Grader.precheck_PLDs (self, stud_ind)

Definition at line 382 of file main.py.

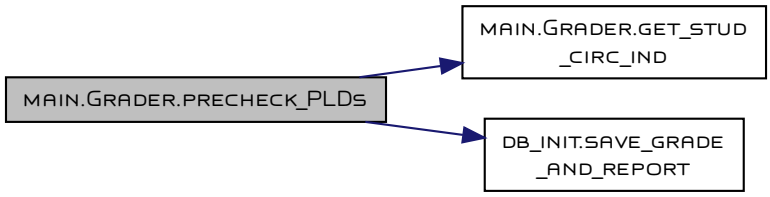
```
00382
00383         student_circuits = CircFile(file).get_parsed_pins2(self.what_to_grade)
00384         errors = 0
00385
00386         out_str = '<br> Next part was generated by automatic grader that I wrote several years ago.' \
00387                   'If you are not agree with something or suspect an error - please send me a message.<br>With this grading approach you cat
      get nonzero grade ' \
00388                   'even if not everything was correct.<br>'
00389         for circ_to_grade in self.what_to_grade:
00390             for good_circ in self.all_my_circuits:
00391                 if good_circ.name.upper() == circ_to_grade.upper():
00392                     cur_ind = self.get_stud_circ_ind(student_circuits, circ_to_grade)
00393                     out_str += '<br>'
00394                     if cur_ind == -1:
00395                         out_str += '<font color="red">{}  NOT FOUND!<br> </font>'.format(circ_to_grade)
00396                         errors += 1
00397                     else:
00398                         check_pins = student_circuits[cur_ind].input_pins
00399                         for i in range(len(check_pins)):
00400                             if check_pins[i][0].lower() != 'c':
00401                                 # print(check_pins[i])
00402                                 if len(check_pins[i][1:]) > 0:
00403                                     try:
00404                                         pos = None
00405                                         for ch in check_pins[i]:
00406                                             if not ch.isalpha():
00407                                                 pos = check_pins[i].index(ch)
```

```
00408                                             break
00409                                         num = int(check_pins[i][pos:])
00410                                     except Exception as e:
00411                                         print(e)
00412                                         continue
00413                                     check_pins[i] = check_pins[i][0:1] + str(num)
00414                             student_circuits_sorted = sorted(check_pins)
00415                             good_circ_sorted = sorted(good_circ.input_pins)
00416                             sm = difflib.SequenceMatcher(None, student_circuits_sorted, good_circ_sorted)
00417                             res_ratio = sm.ratio()
00418 
00419                             if res_ratio > 0.99:
00420                                 out_str += '<font color="green"> {} : PERFECT MATCH!<br> </font>'.format(circ_to_grade)
00421                             elif res_ratio > 0.15:
00422                                 out_str += '{} :Great news : you match ratio is {:.1%} (>75%)<br>{} : <b>FOUND</b> {}  <br>{} : <b>EXPECTED</b> {}
      <br>'\
00423                                     .format(circ_to_grade, res_ratio, circ_to_grade, ' '.join(student_circuits_sorted), circ_to_grade, '
      '.join(good_circ_sorted))
00424                                 errors += 1
00425                             else:
00426                                 out_str += '<font color="red">{} Bad news : you match ratio is only {:.1f}% - this means that you have to ' \
00427                                            'significantly change your circuit. <br>  Please send me a message if you need some advice.<br> ' \
00428                                            '</font>'.format(circ_to_grade, res_ratio)
00429                                 errors += 1
00430 
00431         final_grade = math.ceil(10 * (len(self.what_to_grade) - errors) / len(self.what_to_grade))
00432         # out_str += '<br> Bad grade confidence: ' + conf + ' (this is for Ivan)<br>' + '<br> Next part will be typed manually: <br>'
00433         save_grade_and_report(self.grade_ids[stud_ind], final_grade, out_str, None, self.grader)
00434         return final_grade, out_str
00435 
00436 
00437     def get_stud_id(self):
00438         """
```

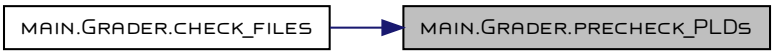
References main.Grader.all_my_circuits, main.Grader.circ_file_name, main.Grader.get_stud_circ_ind(), main.Grader.grader, main.Grader.lab_paths, db_init.save_grade_and_report(), and main.Grader.what_to_grade.

Referenced by main.Grader.check_files().

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.15 prev_circ() def main.Grader.prev_circ (self)

Definition at line 622 of file main.py.

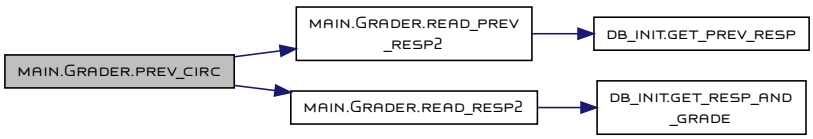
```
00622         # self.check_file(self.cur_idx)
00623         self.user_comment = ''
```

```
00624         graded = self.read_resp2()
00625         if graded:
00626             self.read_prev_resp2()
00627         # if self.check_circ_exist():
00628         #     self.read_resp()
00629         self.stud_id = self.stud_ids[self.cur_idx]
00630         # try:
00631         #     self.read_prev_resp()
00632         # except Exception as e:
00633         #     print('Error during attempt to read prev resp when opening prev circuit: ', e)
00634         #     # TODO add handler
00635         return self.cur_idx
00636 
00637     def check_wrong(self):
00638         """
```

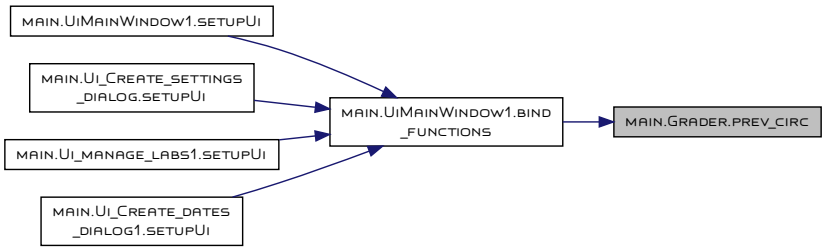
References main.Grader.cur_idx, main.Grader.read_prev_resp2(), main.Grader.read_resp2(), main.Grader.stud_id, main.Grader.stud_ids, and main.Grader.user_comment.

Referenced by main.UiMainWindow1.bind_functions().

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.16 read_prev_resp() def main.Grader.read_prev_resp (self)

Definition at line 581 of file main.py.

```
00581             self.previous_responses = ''  # TODO find same name in folder name
00582             prev_att = int(self.working_dir[-2:-1])
00583             for i in range(prev_att-1, 0, -1):
00584                 prev_working_dir = self.working_dir[:-2] + str(i) + '/'
00585                 for file in os.listdir(prev_working_dir):
00586                     if file.__contains__(self.stud_id):
00587                         # print(file)
00588                         try:
00589                             with open(prev_working_dir + file + '/responce.txt', 'r') as resp_file:
00590                                 self.previous_responses += str(i) + 'th submission :\n\t' \
00591                                                            + '\n'.join(resp_file.readlines())
00592                         except Exception as e:
00593                             print('Error in read prev responce: ', e)
00594 
00595     def next_circ(self):
00596         """
```

References main.Grader.attempt, main.Grader.previous_responses, main.Grader.stud_id, and main.Grader.working_dir.

7.5.3.17 read_prev_resp2() def main.Grader.read_prev_resp2 (self)

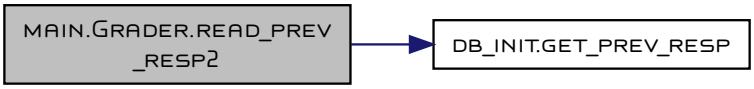
Definition at line 573 of file main.py.

```
00573
00574     def read_prev_resp(self):
00575         """
```

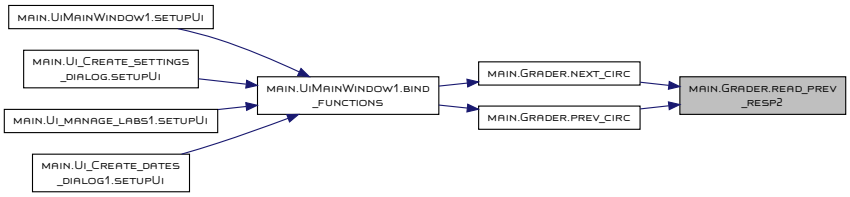
References main.Grader.cur_idx, db_init.get_prev_resp(), main.Grader.lid, main.Grader.previous_responses, and main.Grader.stud_ids.

Referenced by main.Grader.next_circ(), and main.Grader.prev_circ().

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.18 read_resp() def main.Grader.read_resp (self)

Definition at line 544 of file main.py.

```
00544         try:
00545             with open(os.path.join(self.file_list[self.cur_idx], 'responce.txt'), 'r') as resp_file:
00546                 a = resp_file.readlines()
00547                 self.resp_text = ''.join(a)
00548                 self.resp_len = len(self.resp_text)
00549         except Exception as e:
00550             print(e)
00551             self.log_update(sys.exc_info()[0])
00552
00553         try:
00554             with open(os.path.join(self.file_list[self.cur_idx], 'grade.txt'), 'r') as grade_file:
00555                 self.final_grade = int(grade_file.readline())
00556         except Exception as e:
00557             print(e)
00558             self.log_update(sys.exc_info()[0])
00559
00560         # self.read_prev_resp()
00561
00562     def read_resp2(self):
00563         self.final_grade, self.resp_text, self.user_comment, graded = get_resp_and_grade(self.grade_ids[self.cur_idx])
```

References main.Grader.cur_idx, main.Grader.file_list, main.CircFile.final_grade, main.Grader.final_grade, main.Grader.log_update(), main.Grader.resp_len, main.Grader.resp_text, main.Grader.submitted, and main.Grader.timestamps.

Here is the call graph for this function:

7.5.3.19 read_resp2() def main.Grader.read_resp2 (self)

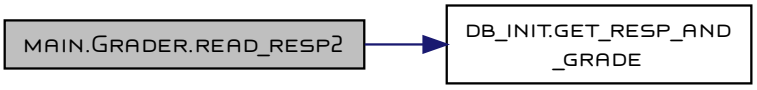
Definition at line 564 of file main.py.

```
00564         if graded is None:
00565             self.final_grade = self.lab_max_grade
00566             self.resp_text = 'I did not find any errors. Good job!'
00567         # self.resp_text = '' if self.resp_text is None else self.resp_text
00568         self.resp_len = len(self.resp_text)
00569         return graded
00570 
00571     def read_prev_resp2(self):
00572         self.previous_responses = get_prev_resp(self.grade_ids[self.cur_idx], self.stud_ids[self.cur_idx], self.lid)
```

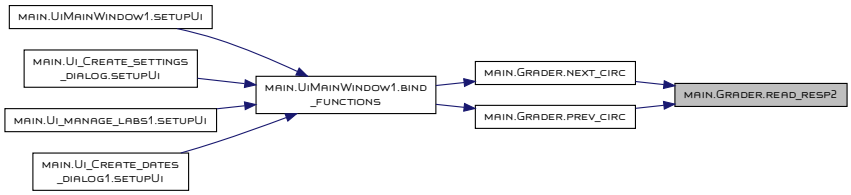
References main.Grader.cur_idx, main.CircFile.final_grade, main.Grader.final_grade, db_init.get_resp_and_grade(), main.Grader.lab_max_grade, main.Grader.resp_len, main.Grader.resp_text, and main.Grader.user_comment.

Referenced by main.Grader.next_circ(), and main.Grader.prev_circ().

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.20 save_all() def main.Grader.save_all (self)

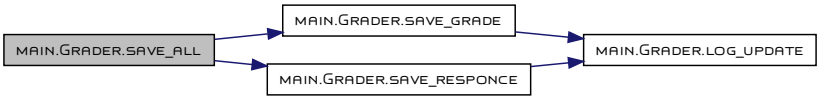
Definition at line 677 of file main.py.

```
00677         self.save_responce()
00678 
00679 
00680     def save_all2(self):
00681         """
```

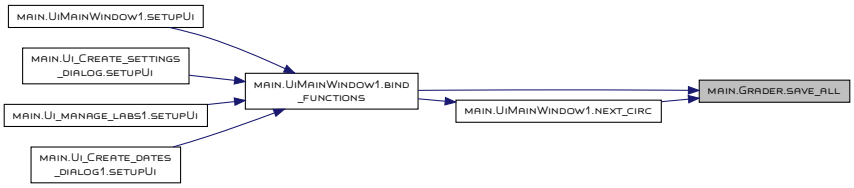
References main.Grader.save_grade(), and main.Grader.save_responce().

Referenced by main.UiMainWindow1.bind_functions(), and main.UiMainWindow1.next_circ().

Here is the call graph for this function:



Here is the caller graph for this function:



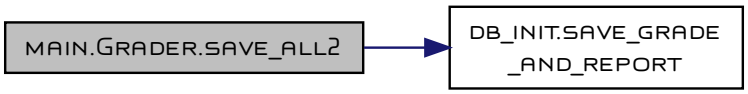
7.5.3.21 save_all2() def main.Grader.save_all2 (self)

Definition at line 686 of file main.py.

```
00686
00687     def generate_response(self):
00688         """
```

References main.Grader.cur_idx, main.CircFile.final_grade, main.Grader.final_grade, main.Grader.grader, main.Grader.resp_text, db_init.save_grade_and_report(), and main.Grader.user_comment.

Here is the call graph for this function:



7.5.3.22 save_grade() def main.Grader.save_grade (self)

Definition at line 652 of file main.py.

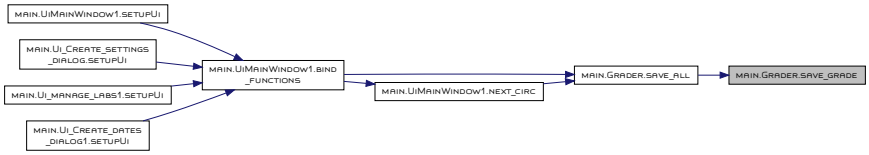
```
00652         with open(file, 'w') as grade_file:
00653             grade_file.write(str(self.final_grade))
00654
00655         self.log_update('Grade saved')
00656
00657     def save_responce(self):
00658         """
```

References main.Grader.cur_idx, main.CircFile.final_grade, main.Grader.final_grade, main.Grader.lab_paths, and main.Grader.log_update().

Referenced by main.Grader.save_all().

Here is the call graph for this function:

Here is the caller graph for this function:



7.5.3.23 save_responce() def main.Grader.save_responce (self)

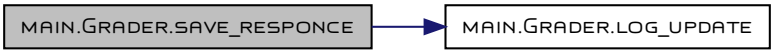
Definition at line 664 of file main.py.

```
00664         with open(file, 'w') as resp_file:
00665             resp_file.write(self.resp_text)
00666             if self.user_comment:
00667                 resp_file.write('\nAdditional comment: ' + self.user_comment + '\n')
00668         self.log_update('Responce saved')
00669 
00670     def save_all(self):
00671         """
```

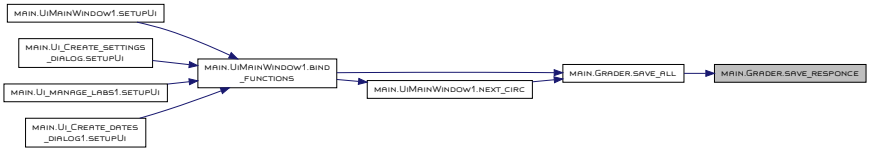
References main.Grader.cur_idx, main.Grader.lab_paths, main.Grader.log_update(), main.Grader.resp_text, and main.Grader.user_comment.

Referenced by main.Grader.save_all().

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.4 Member Data Documentation

7.5.4.1 all_my_circuits main.Grader.all_my_circuits

Definition at line 326 of file main.py.
Referenced by main.Grader.precheck_PLDs().

7.5.4.2 attempt main.Grader.attempt

Definition at line 233 of file main.py.
Referenced by main.Grader.check_files(), main.Grader.open_dir(), and main.Grader.read_prev_resp().

7.5.4.3 circ_file_name main.Grader.circ_file_name

Definition at line 257 of file main.py.
Referenced by main.Grader.check_circ_exist(), main.Grader.check_files(), main.Grader.open_dir(), and main.Grader.precheck_PLDs().

7.5.4.4 circ_obj_ref main.Grader.circ_obj_ref

Definition at line 261 of file main.py.
Referenced by main.Grader.check_file(), and main.Grader.get_parsed_pins().

7.5.4.5 cur_idx main.Grader.cur_idx

Definition at line 252 of file main.py.
Referenced by main.Grader.check_circ_exist(), main.Grader.check_file(), main.Grader.next_circ(), main.Grader.prev_circ(), main.Grader.read_prev_resp2(), main.Grader.read_resp(), main.Grader.read_resp2(), main.Grader.save_all2(), main.Grader.save_grade(), and main.Grader.save_responce().

7.5.4.6 file_list main.Grader.file_list

Definition at line 249 of file main.py.
Referenced by main.Grader.check_circ_exist(), main.Grader.check_file(), and main.Grader.read_resp().

7.5.4.7 final_grade main.Grader.final_grade

Definition at line 243 of file main.py.
Referenced by main.Grader.check_circ_exist(), main.Grader.check_file(), main.Grader.check_wrong(), main.Grader.read_resp(), main.Grader.read_resp2(), main.Grader.save_all2(), and main.Grader.save_grade().

7.5.4.8 global_log main.Grader.global_log

Definition at line 245 of file main.py.
Referenced by main.Grader.log_update().

7.5.4.9 grader main.Grader.grader

Definition at line 264 of file main.py.
Referenced by main.Grader.check_files(), main.Grader.precheck_PLDs(), and main.Grader.save_all2().

7.5.4.10 input_correct main.Grader.input_correct

Definition at line 238 of file main.py.
Referenced by main.Grader.generate_response(), and main.Grader.get_parsed_pins().

7.5.4.11 input_suggestion main.Grader.input_suggestion

Definition at line 254 of file main.py.
Referenced by main.Grader.add_to_common_answers().

7.5.4.12 lab_id main.Grader.lab_id

Definition at line 263 of file main.py.
Referenced by main.Grader.open_dir().

7.5.4.13 lab_max_grade main.Grader.lab_max_grade

Definition at line 240 of file main.py.
Referenced by main.Grader.check_file(), main.Grader.open_dir(), and main.Grader.read_resp2().

7.5.4.14 lab_num main.Grader.lab_num

Definition at line 259 of file main.py.
Referenced by main.Grader.check_files(), and main.Grader.open_dir().

7.5.4.15 lab_paths main.Grader.lab_paths

Definition at line 307 of file main.py.
Referenced by main.Grader.check_files(), main.Grader.precheck_PLDs(), main.Grader.save_grade(), and main.Grader.save_responce().

7.5.4.16 lab_type main.Grader.lab_type

Definition at line 258 of file main.py.
Referenced by main.Grader.check_files(), and main.Grader.open_dir().

7.5.4.17 lid main.Grader.lid

Definition at line 306 of file main.py.
Referenced by main.Grader.read_prev_resp2().

7.5.4.18 logisim_pid main.Grader.logisim_pid

Definition at line 256 of file main.py.

7.5.4.19 output_correct main.Grader.output_correct

Definition at line 239 of file main.py.
Referenced by main.Grader.generate_response(), and main.Grader.get_parsed_pins().

7.5.4.20 previous_responses main.Grader.previous_responses

Definition at line 246 of file main.py.
Referenced by main.Grader.read_prev_resp(), and main.Grader.read_prev_resp2().

7.5.4.21 resp_len main.Grader.resp_len

Definition at line 255 of file main.py.
Referenced by main.Grader.check_circ_exist(), main.Grader.check_wrong(), main.Grader.generate_response(), main.Grader.read_resp(), and main.Grader.read_resp2().

7.5.4.22 resp_text main.Grader.resp_text

Definition at line 250 of file main.py.
Referenced by main.Grader.check_circ_exist(), main.Grader.check_wrong(), main.Grader.generate_response(), main.Grader.read_resp(), main.Grader.read_resp2(), main.Grader.save_all2(), and main.Grader.save_responce().

7.5.4.23 semester main.Grader.semester

Definition at line 305 of file main.py.

7.5.4.24 stud_id main.Grader.stud_id

Definition at line 236 of file main.py.
Referenced by main.Grader.get_stud_id(), main.Grader.log_update(), main.Grader.next_circ(), main.Grader.prev_circ(), and main.Grader.read_prev_resp().

7.5.4.25 stud_ids main.Grader.stud_ids

Definition at line 235 of file main.py.
Referenced by main.Grader.check_files(), main.Grader.next_circ(), main.Grader.prev_circ(), and main.Grader.read_prev_resp2().

7.5.4.26 submitted main.Grader.submitted

Definition at line 237 of file main.py.
Referenced by main.Grader.read_resp().

7.5.4.27 subtract main.Grader.subtract

Definition at line 241 of file main.py.
Referenced by main.Grader.check_file(), and main.Grader.get_parsed_pins().

7.5.4.28 time main.Grader.time

Definition at line 260 of file main.py.

7.5.4.29 time_cur main.Grader.time_cur

Definition at line 312 of file main.py.

7.5.4.30 time_cur_qt main.Grader.time_cur_qt

Definition at line 316 of file main.py.

7.5.4.31 time_from main.Grader.time_from

Definition at line 310 of file main.py.
Referenced by main.Grader.check_files().

7.5.4.32 time_from_qt main.Grader.time_from_qt
Definition at line 314 of file main.py.

7.5.4.33 time_to main.Grader.time_to
Definition at line 311 of file main.py.
Referenced by main.Grader.check_files().

7.5.4.34 time_to_qt main.Grader.time_to_qt
Definition at line 315 of file main.py.

7.5.4.35 timestamps main.Grader.timestamps
Definition at line 234 of file main.py.
Referenced by main.Grader.check_files(), and main.Grader.read_resp().

7.5.4.36 to_date main.Grader.to_date
Definition at line 232 of file main.py.

7.5.4.37 tot_elem main.Grader.tot_elem
Definition at line 262 of file main.py.

7.5.4.38 user_comment main.Grader.user_comment
Definition at line 251 of file main.py.
Referenced by main.Grader.generate_response(), main.Grader.next_circ(), main.Grader.prev_circ(), main.Grader.read_resp2(), main.Grader.save_all2(), and main.Grader.save_responce().

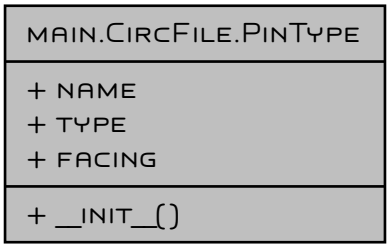
7.5.4.39 what_to_grade main.Grader.what_to_grade
Definition at line 320 of file main.py.
Referenced by main.Grader.precheck_PLDs().

7.5.4.40 working_dir main.Grader.working_dir
Definition at line 253 of file main.py.
Referenced by main.Grader.open_dir(), and main.Grader.read_prev_resp().
The documentation for this class was generated from the following file:

- main.py

7.6 main.CircFile.PinType Class Reference

Collaboration diagram for main.CircFile.PinType:



Public Member Functions

- def __init__ (self, name, iotype, facing=None)

Public Attributes

- name
- type
- facing

7.6.1 Detailed Description

Definition at line 91 of file main.py.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 __init__() def main.CircFile.PinType.__init__ (self, name, iotype, facing = None)

Definition at line 92 of file main.py.

```
00092         def __init__(self, name, iotype, facing=None):
00093             self.name = name
00094             self.type = iotype
00095             self.facing = facing
00096
00097     def __init__(self, filename):
```

7.6.3 Member Data Documentation

7.6.3.1 facing main.CircFile.PinType.facing

Definition at line 95 of file main.py.

7.6.3.2 name main.CircFile.PinType.name

Definition at line 93 of file main.py.

7.6.3.3 type main.CircFile.PinType.type

Definition at line 94 of file main.py.

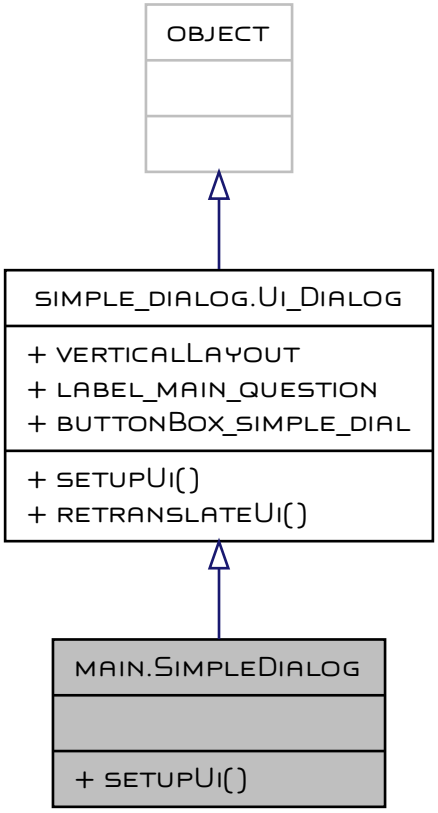
The documentation for this class was generated from the following file:

- main.py

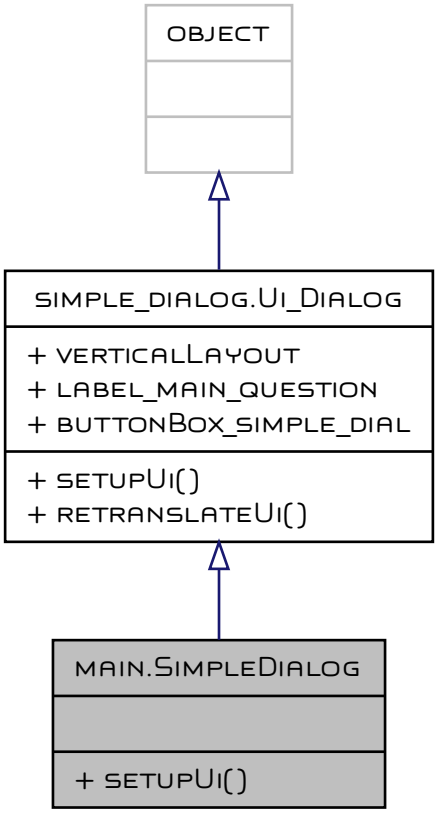
7.7 main.SimpleDialog Class Reference

Wrapper class for very simple Ok|Cancel dialog.

Inheritance diagram for main.SimpleDialog:



Collaboration diagram for main.SimpleDialog:



Public Member Functions

- def setupUi (self, Dialog, phrase)

Additional Inherited Members

7.7.1 Detailed Description

Wrapper class for very simple Ok|Cancel dialog.
Definition at line 1564 of file main.py.

7.7.2 Member Function Documentation

7.7.2.1 setupUi() def main.SimpleDialog.setupUi (self, Dialog, phrase)

Definition at line 1569 of file main.py.

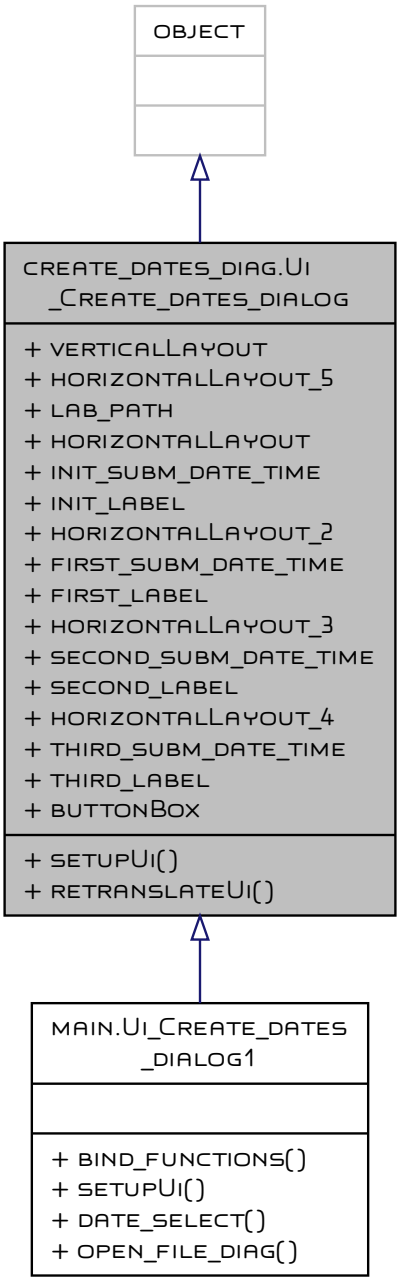
```
01569         self.label_main_question.setText(phrase)
01570
01571
01572 class Ui_manage_labs1(Ui_manage_labs):
01573     srv_sync_path = None
```

References simple_dialog.Ui_Dialog.label_main_question.
The documentation for this class was generated from the following file:

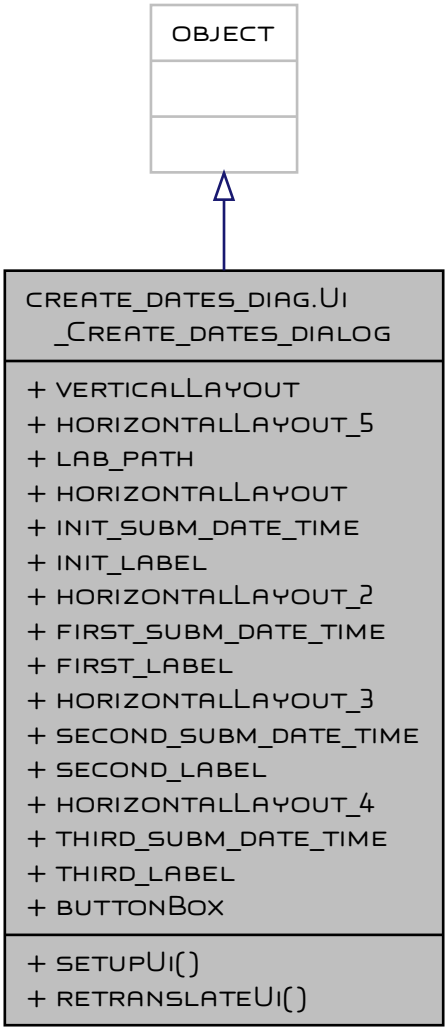
- main.py

7.8 create_dates_diag.Ui_Create_dates_dialog Class Reference

Inheritance diagram for create_dates_diag.Ui_Create_dates_dialog:



Collaboration diagram for create_dates_diag.Ui_Create_dates_dialog:



Public Member Functions

- def setupUi (self, Create_dates_dialog)
- def retranslateUi (self, Create_dates_dialog)

Public Attributes

- verticalLayout
- horizontalLayout_5
- lab_path
- horizontalLayout
- init_subm_date_time
- init_label
- horizontalLayout_2
- first_subm_date_time
- first_label

- horizontalLayout_3
- second_subm_date_time
- second_label
- horizontalLayout_4
- third_subm_date_time
- third_label
- buttonBox

7.8.1 Detailed Description

Definition at line 11 of file create_dates_diag.py.

7.8.2 Member Function Documentation

7.8.2.1 retranslateUi() def create_dates_diag.Ui_Create_dates_dialog.retranslateUi (self, Create_dates_dialog)

Definition at line 96 of file create_dates_diag.py.

```
00096     def retranslateUi(self, Create_dates_dialog):
00097         _translate = QtCore.QCoreApplication.translate
00098         Create_dates_dialog.setWindowTitle(_translate("Create_dates_dialog", "Dialog"))
00099         self.lab_path.setToolTip(_translate("Create_dates_dialog", "Tripple for file dialog"))
00100         self.lab_path.setPlaceholderText(_translate("Create_dates_dialog", "DoubleClick to select path"))
00101         self.init_label.setText(_translate("Create_dates_dialog", "Submission date"))
00102         self.first_label.setText(_translate("Create_dates_dialog", "1st resubmission"))
00103         self.second_label.setText(_translate("Create_dates_dialog", "2nd resubmission"))
00104         self.third_label.setText(_translate("Create_dates_dialog", "3rd resubmission"))
00105
00106 from qt_class_improvements import BetterLineEdit
```

References create_dates_diag.Ui_Create_dates_dialog.first_label, create_dates_diag.Ui_Create_dates_dialog.init_label, create_dates_diag.Ui_Create_dates_dialog.lab_path, create_dates_diag.Ui_Create_dates_dialog.second_label, and create_dates_diag.Ui_Create_dates_dialog.third_label.

7.8.2.2 setupUi() def create_dates_diag.Ui_Create_dates_dialog.setupUi (self, Create_dates_dialog)

Definition at line 12 of file create_dates_diag.py.

```
00012     def setupUi(self, Create_dates_dialog):
00013         Create_dates_dialog.setObjectName("Create_dates_dialog")
00014         Create_dates_dialog.resize(589, 250)
00015         Create_dates_dialog.setMinimumSize(QtCore.QSize(500, 250))
00016         Create_dates_dialog.setMaximumSize(QtCore.QSize(1000, 300))
00017         icon = QtGui.QIcon()
00018         icon.addPixmap(QtGui.QPixmap("os_linux_1.ico"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
00019         Create_dates_dialog.setWindowIcon(icon)
00020         self.verticalLayout = QtWidgets.QVBoxLayout(Create_dates_dialog)
00021         self.verticalLayout.setObjectName("verticalLayout")
00022         self.horizontalLayout_5 = QtWidgets.QHBoxLayout()
00023         self.horizontalLayout_5.setObjectName("horizontalLayout_5")
00024         self.lab_path = BetterLineEdit(Create_dates_dialog)
00025         self.lab_path.setFocusPolicy(QtCore.Qt.StrongFocus)
00026         self.lab_path.setStatusTip("")
00027         self.lab_path.setWhatsThis("")
00028         self.lab_path.setAccessibleName("")
00029         self.lab_path.setAccessibleDescription("")
00030         self.lab_path.setInputMask("")
00031         self.lab_path.setReadOnly(False)
00032         self.lab_path.setCursorMoveStyle(QtCore.Qt.LogicalMoveStyle)
00033         self.lab_path.setClearButtonEnabled(False)
00034         self.lab_path.setObjectName("lab_path")
00035         self.horizontalLayout_5.addWidget(self.lab_path)
00036         self.verticalLayout.addLayout(self.horizontalLayout_5)
00037         self.horizontalLayout = QtWidgets.QHBoxLayout()
00038         self.horizontalLayout.setObjectName("horizontalLayout")
00039         self.init_subm_date_time = QtWidgets.QDateTimeEdit(Create_dates_dialog)
00040         self.init_subm_date_time.setMaximumSize(QtCore.QSize(150, 40))
00041         self.init_subm_date_time.setCalendarPopup(True)
00042         self.init_subm_date_time.setObjectName("init_subm_date_time")
00043         self.horizontalLayout.addWidget(self.init_subm_date_time)
00044         self.init_label = QtWidgets.QLabel(Create_dates_dialog)
00045         self.init_label.setObjectName("init_label")
00046         self.horizontalLayout.addWidget(self.init_label)
00047         self.verticalLayout.addLayout(self.horizontalLayout)
00048         self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
00049         self.horizontalLayout_2.setObjectName("horizontalLayout_2")
00050         self.first_subm_date_time = QtWidgets.QDateTimeEdit(Create_dates_dialog)
00051         self.first_subm_date_time.setMaximumSize(QtCore.QSize(150, 35))
```

```
00052         self.first_subm_date_time.setCalendarPopup(True)
00053         self.first_subm_date_time.setObjectName("first_subm_date_time")
00054         self.horizontalLayout_2.addWidget(self.first_subm_date_time)
00055         self.first_label = QtWidgets.QLabel(Create_dates_dialog)
00056         self.first_label.setObjectName("first_label")
00057         self.horizontalLayout_2.addWidget(self.first_label)
00058         self.verticalLayout.addLayout(self.horizontalLayout_2)
00059         self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
00060         self.horizontalLayout_3.setObjectName("horizontalLayout_3")
00061         self.second_subm_date_time = QtWidgets.QDateTimeEdit(Create_dates_dialog)
00062         self.second_subm_date_time.setMaximumSize(QtCore.QSize(150, 35))
00063         self.second_subm_date_time.setCalendarPopup(True)
00064         self.second_subm_date_time.setObjectName("second_subm_date_time")
00065         self.horizontalLayout_3.addWidget(self.second_subm_date_time)
00066         self.second_label = QtWidgets.QLabel(Create_dates_dialog)
00067         self.second_label.setObjectName("second_label")
00068         self.horizontalLayout_3.addWidget(self.second_label)
00069         self.verticalLayout.addLayout(self.horizontalLayout_3)
00070         self.horizontalLayout_4 = QtWidgets.QHBoxLayout()
00071         self.horizontalLayout_4.setObjectName("horizontalLayout_4")
00072         self.third_subm_date_time = QtWidgets.QDateTimeEdit(Create_dates_dialog)
00073         self.third_subm_date_time.setMaximumSize(QtCore.QSize(150, 35))
00074         self.third_subm_date_time.setCalendarPopup(True)
00075         self.third_subm_date_time.setObjectName("third_subm_date_time")
00076         self.horizontalLayout_4.addWidget(self.third_subm_date_time)
00077         self.third_label = QtWidgets.QLabel(Create_dates_dialog)
00078         self.third_label.setObjectName("third_label")
00079         self.horizontalLayout_4.addWidget(self.third_label)
00080         self.verticalLayout.addLayout(self.horizontalLayout_4)
00081         self.buttonBox = QtWidgets.QDialogButtonBox(Create_dates_dialog)
00082         self.buttonBox.setMaximumSize(QtCore.QSize(16777215, 40))
00083         self.buttonBox.setOrientation(QtCore.Qt.Horizontal)
00084         self.buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Abort|QtWidgets.QDialogButtonBox.SaveAll)
00085         self.buttonBox.setObjectName("buttonBox")
00086         self.verticalLayout.addWidget(self.buttonBox)
00087 
00088         self.retranslateUi(Create_dates_dialog)
00089         self.buttonBox.accepted.connect(Create_dates_dialog.accept)
00090         self.buttonBox.rejected.connect(Create_dates_dialog.reject)
00091         QtCore.QMetaObject.connectSlotsByName(Create_dates_dialog)
00092         Create_dates_dialog.setTabOrder(self.init_subm_date_time, self.first_subm_date_time)
00093         Create_dates_dialog.setTabOrder(self.first_subm_date_time, self.second_subm_date_time)
00094         Create_dates_dialog.setTabOrder(self.second_subm_date_time, self.third_subm_date_time)
00095 
```

7.8.3 Member Data Documentation

7.8.3.1 buttonBox create_dates_diag.Ui_Create_dates_dialog.buttonBox

Definition at line 81 of file create_dates_diag.py.
Referenced by main.Ui_Create_settings_dialog.bind_functions(), main.Ui_Create_settings_dialog.create_or_update_settings_db(), main.Ui_Create_settings_dialog.read_settings_data(), main.Ui_Create_settings_dialog.set_apply_restet_active(), main.Ui_Create_settings_dialog.set_default_user_input_with_paths(), main.Ui_Create_settings_dialog.setupUi(), and main.Ui_Create_settings_dialog.update_user_input_with_paths().

7.8.3.2 first_label create_dates_diag.Ui_Create_dates_dialog.first_label

Definition at line 55 of file create_dates_diag.py.
Referenced by create_dates_diag.Ui_Create_dates_dialog.retranslateUi().

7.8.3.3 first_subm_date_time create_dates_diag.Ui_Create_dates_dialog.first_subm_date_time

Definition at line 50 of file create_dates_diag.py.
Referenced by main.Ui_Create_dates_dialog1.date_select().

7.8.3.4 horizontalLayout create_dates_diag.Ui_Create_dates_dialog.horizontalLayout

Definition at line 37 of file create_dates_diag.py.

7.8.3.5 horizontalLayout_2 create_dates_diag.Ui_Create_dates_dialog.horizontalLayout_2

Definition at line 48 of file create_dates_diag.py.

7.8.3.6 horizontalLayout_3 create_dates_diag.Ui_Create_dates_dialog.horizontalLayout_3

Definition at line 59 of file create_dates_diag.py.

7.8.3.7 horizontalLayout_4

create_dates_diag.Ui_Create_dates_dialog.horizontalLayout_4
Definition at line 70 of file create_dates_diag.py.

7.8.3.8 horizontalLayout_5

create_dates_diag.Ui_Create_dates_dialog.horizontalLayout_5
Definition at line 22 of file create_dates_diag.py.

7.8.3.9 init_label

create_dates_diag.Ui_Create_dates_dialog.init_label
Definition at line 44 of file create_dates_diag.py.
Referenced by create_dates_diag.Ui_Create_dates_dialog.retranslateUi().

7.8.3.10 init_subm_date_time

create_dates_diag.Ui_Create_dates_dialog.init_subm_date_time
Definition at line 39 of file create_dates_diag.py.
Referenced by main.Ui_Create_dates_dialog1.bind_functions(), main.Ui_Create_dates_dialog1.date_select(), and main.Ui_Create_dates_dialog1.setupUi().

7.8.3.11 lab_path

create_dates_diag.Ui_Create_dates_dialog.lab_path
Definition at line 24 of file create_dates_diag.py.
Referenced by main.Ui_Create_dates_dialog1.bind_functions(), main.Ui_Create_dates_dialog1.open_file_diag(), create_dates_diag.Ui_Create_dates_dialog.retranslateUi(), and main.Ui_Create_dates_dialog1.setupUi().

7.8.3.12 second_label

create_dates_diag.Ui_Create_dates_dialog.second_label
Definition at line 66 of file create_dates_diag.py.
Referenced by create_dates_diag.Ui_Create_dates_dialog.retranslateUi().

7.8.3.13 second_subm_date_time

create_dates_diag.Ui_Create_dates_dialog.second_subm_date_time
Definition at line 61 of file create_dates_diag.py.
Referenced by main.Ui_Create_dates_dialog1.date_select().

7.8.3.14 third_label

create_dates_diag.Ui_Create_dates_dialog.third_label
Definition at line 77 of file create_dates_diag.py.
Referenced by create_dates_diag.Ui_Create_dates_dialog.retranslateUi().

7.8.3.15 third_subm_date_time

create_dates_diag.Ui_Create_dates_dialog.third_subm_date_time
Definition at line 72 of file create_dates_diag.py.
Referenced by main.Ui_Create_dates_dialog1.date_select().

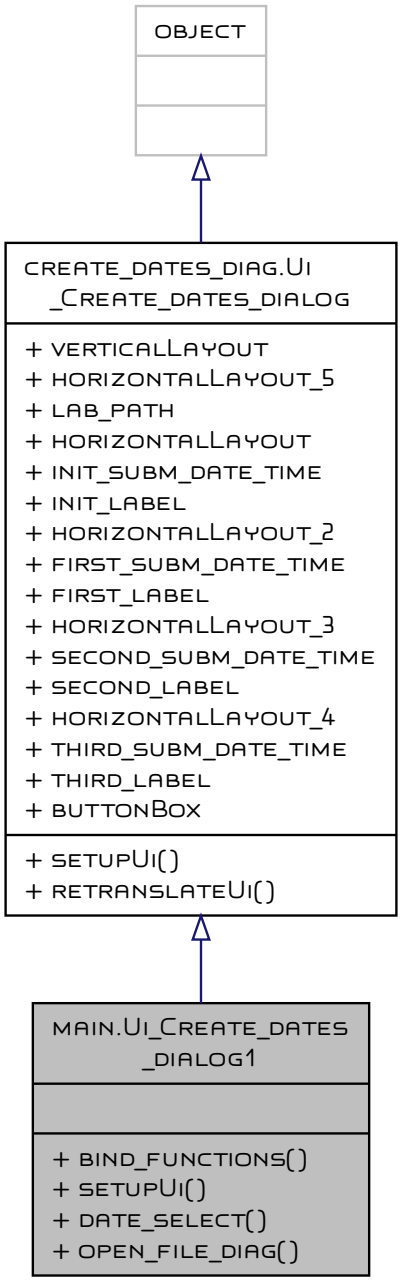
7.8.3.16 verticalLayout

create_dates_diag.Ui_Create_dates_dialog.verticalLayout
Definition at line 20 of file create_dates_diag.py.
The documentation for this class was generated from the following file:

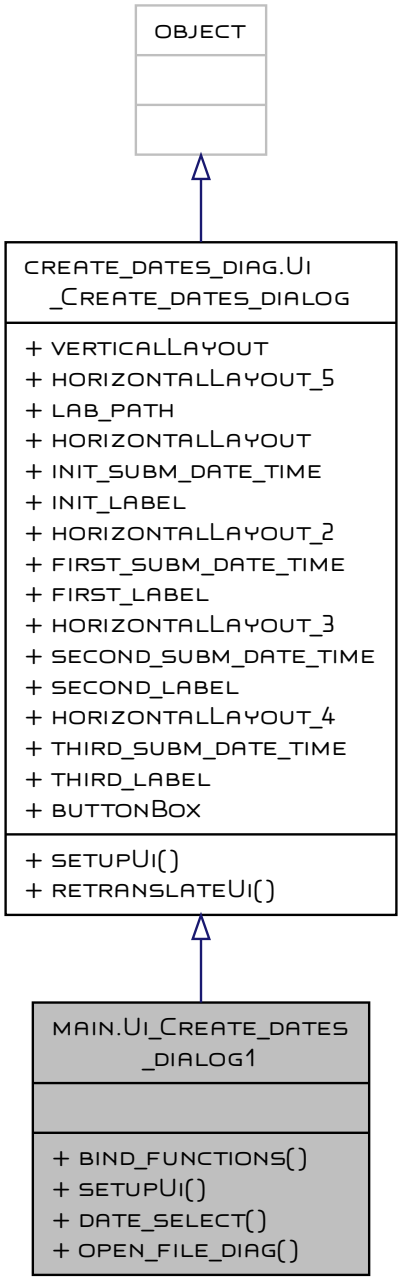
- create_dates_diag.py

7.9 main.Ui_Create_dates_dialog1 Class Reference

Inheritance diagram for main.Ui_Create_dates_dialog1:



Collaboration diagram for main.Ui_Create_dates_dialog1:



Public Member Functions

- def bind_functions (self)
- def setupUi (self, Create_dates_dialog, selected_lab='')
- def date_select (self)
- def open_file_diag (self)

Additional Inherited Members

7.9.1 Detailed Description

Definition at line 1928 of file main.py.

7.9.2 Member Function Documentation

7.9.2.1 bind_functions() def main.Ui_Create_dates_dialog1.bind_functions (self)

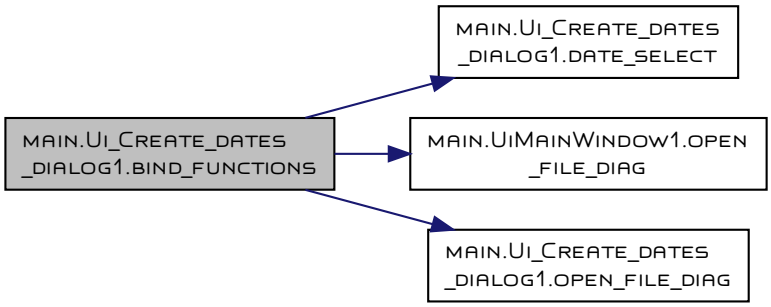
Definition at line 1934 of file main.py.

```
01934         # self.select_file_path.clicked.connect(self.open_file_diag)
01935         # self.lineEdit.left_clicked[int].connect(self.dummy_d)
01936         self.lab_path.dclicked.connect(self.open_file_diag)
01937 
01938     # noinspection PyMethodMayBeStatic
01939     def __dummy_d(self, nb):
```

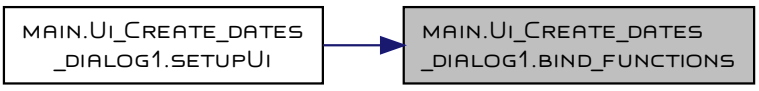
References main.Ui_Create_dates_dialog1.date_select(), create_dates_diag.Ui_Create_dates_dialog.init_subm_date_time, create_dates_diag.Ui_Create_dates_dialog.lab_path, main.UiMainWindow1.open_file_diag(), and main.Ui_Create_dates_dialog1.open_file_diag().

Referenced by main.Ui_Create_dates_dialog1.setupUi().

Here is the call graph for this function:



Here is the caller graph for this function:



7.9.2.2 date_select() def main.Ui_Create_dates_dialog1.date_select (self)

Definition at line 1987 of file main.py.

```
01987         """
01988         Creates File browser in a new window.
01989         Returns: nothing.
01990         """
01991         obtained_dir = QFileDialog.getExistingDirectory(caption='Select where to create due files',
```

References create_dates_diag.Ui_Create_dates_dialog.first_subm_date_time, create_dates_diag.Ui_Create_dates_dialog.init_subm_date_time, create_dates_diag.Ui_Create_dates_dialog.second_subm_date_time, and create_dates_diag.Ui_Create_dates_dialog.third_subm_date_time.

Referenced by main.Ui_Create_dates_dialog1.bind_functions().

Here is the caller graph for this function:

7.9.2.3 open_file_diag()

```
def main.Ui_Create_dates_dialog1.open_file_diag (
                self )
```

Definition at line 1996 of file main.py.

```
01996
01997 if __name__ == "__main__":
01998     # generate_final_grades('./grades.sqlite3', 2018, 1)
01999     # reconstruct_grades_and_comments()
02000     # update_lab_submissions_paths('./grades.sqlite3', '/home/vanya/Documents/3130_labs/2018/', 2018, 1)
02001     # import_previous_grades_into_db('./grades.sqlite3', 'grades.xls', 2018, 1)
02002     # load_student_list_into_grades_db('./grades.sqlite3', 2018, '1')
```

References create_dates_diag.Ui_Create_dates_dialog.lab_path.

Referenced by main.Ui_Create_dates_dialog1.bind_functions().

Here is the caller graph for this function:

7.9.2.4 setupUi()

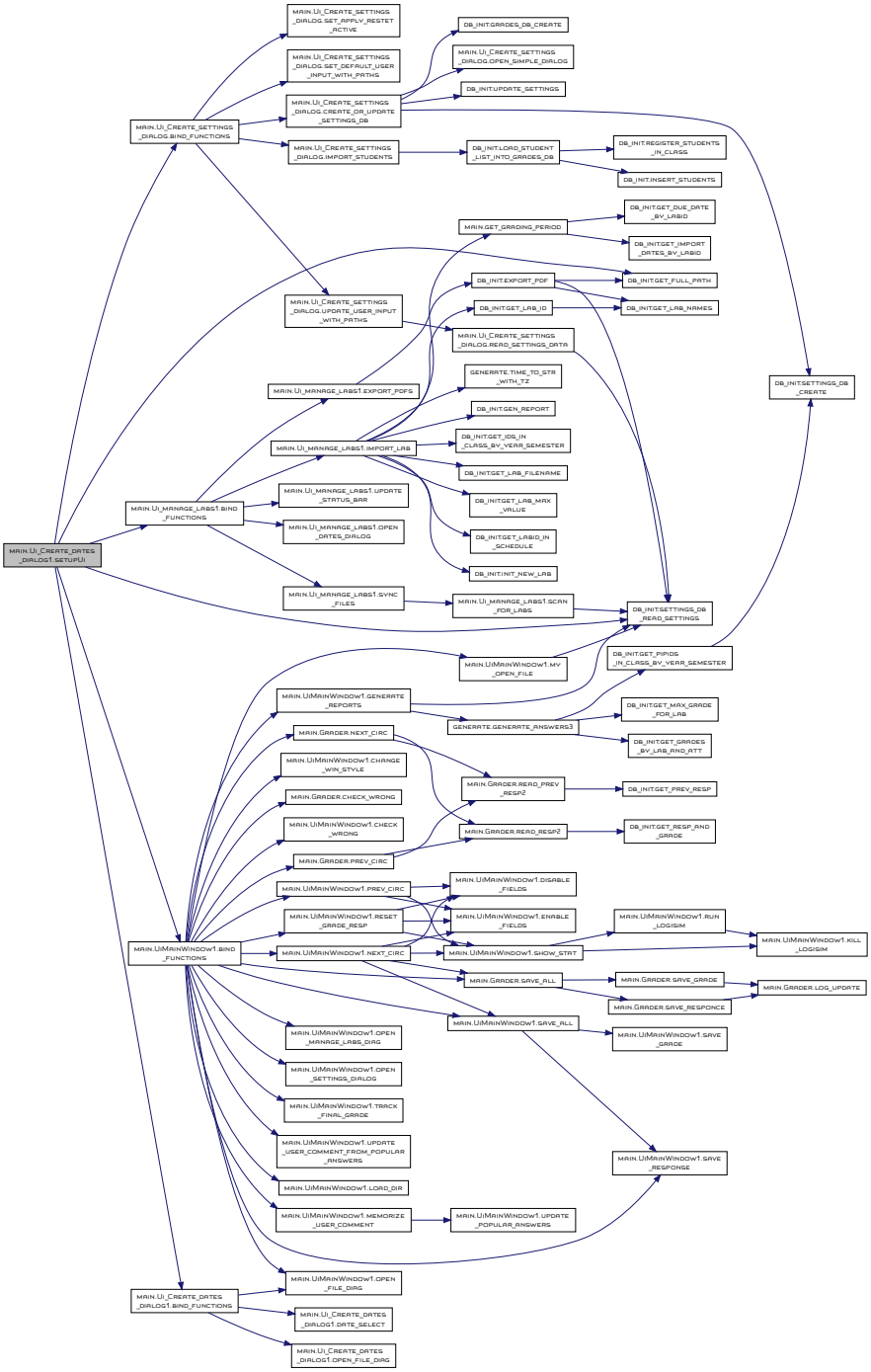
```
def main.Ui_Create_dates_dialog1.setupUi (
                self,
                Create_dates_dialog,
                selected_lab =  '' )
```

Definition at line 1971 of file main.py.

```
01971         try:
01972             good_path += selected_lab + '/'
01973         except Exception as e:
01974             print('Exception when tried to append selected folder from Manage labs. ', e)
01975         self.lab_path.setText(good_path)
01976
01977     def date_select(self):
01978         """
01979         Automatically set next due dates.
01980         Returns: nothing.
01981         """
01982         self.first_subm_date_time.setDateTime(self.init_subm_date_time.dateTime().addDays(7))
```

References main.UiMainWindow1.bind_functions(), main.Ui_Create_settings_dialog.bind_functions(), main.Ui_manage_labs1.bind_functions(), main.Ui_Create_dates_dialog1.bind_functions(), db_init.get_full_path(), create_dates_diag.Ui_Create_dates_dialog.init_subm_date_time, create_dates_diag.Ui_Create_dates_dialog.lab_path, and db_init.settings_db_read_settings().

Here is the call graph for this function:



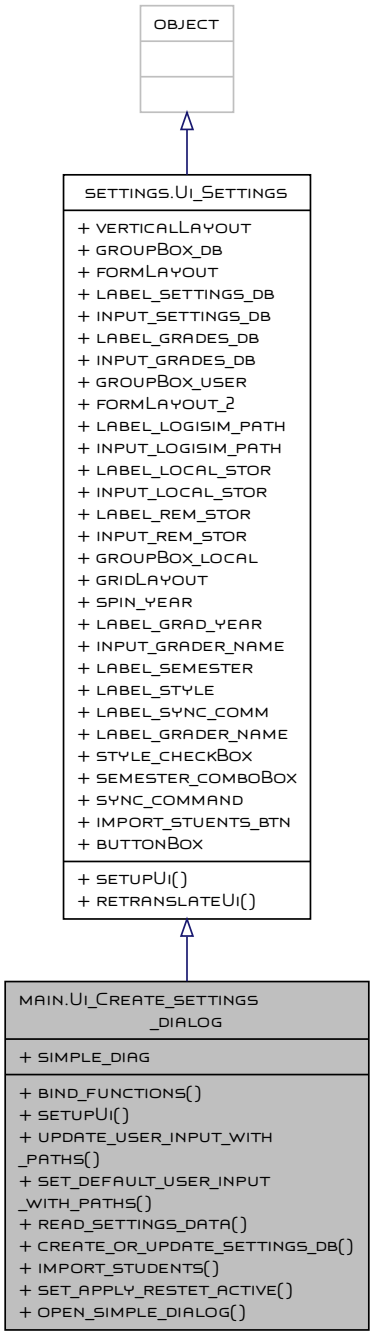
The documentation for this class was generated from the following file:

- main.py

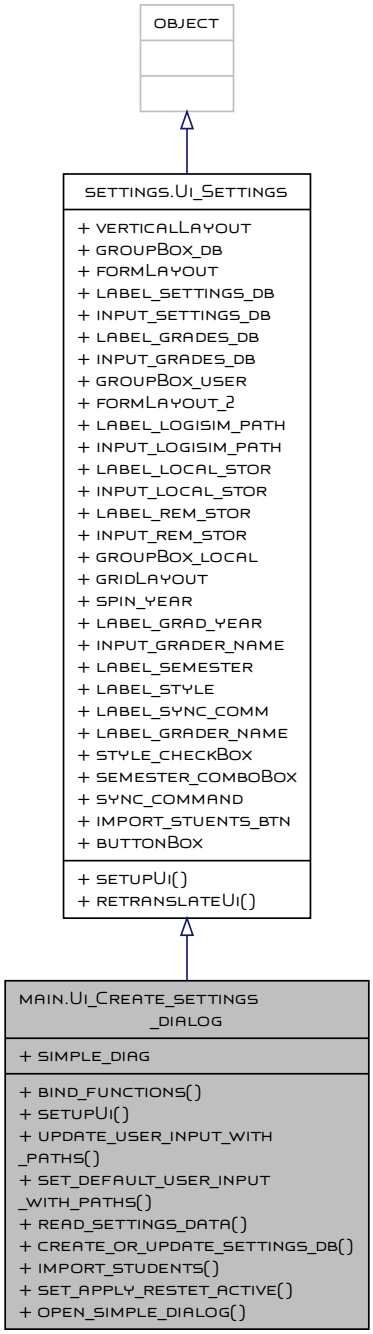
7.10 main.Ui_Create_settings_dialog Class Reference

Creates window that provides user with convenient way of changing settings that are stored in sqlite3 db.

Inheritance diagram for main.Ui_Create_settings_dialog:



Collaboration diagram for main.Ui_Create_settings_dialog:



Public Member Functions

- def bind_functions (self)
- def setupUi (self, Settings)
- def update_user_input_with_paths (self)

Reads settings parameters from DB and sets appropriate fields with obtained values.

- def set_default_user_input_with_paths (self)
- def read_settings_data (self)
- def create_or_update_settings_db (self)
- def import_students (self)
- def set_apply_restet_active (self)
- def open_simple_dialog (self, phrase)

Public Attributes

- simple_diag

7.10.1 Detailed Description

Creates window that provides user with convenient way of changing settings that are stored in sqlite3 db.
Definition at line 1341 of file main.py.

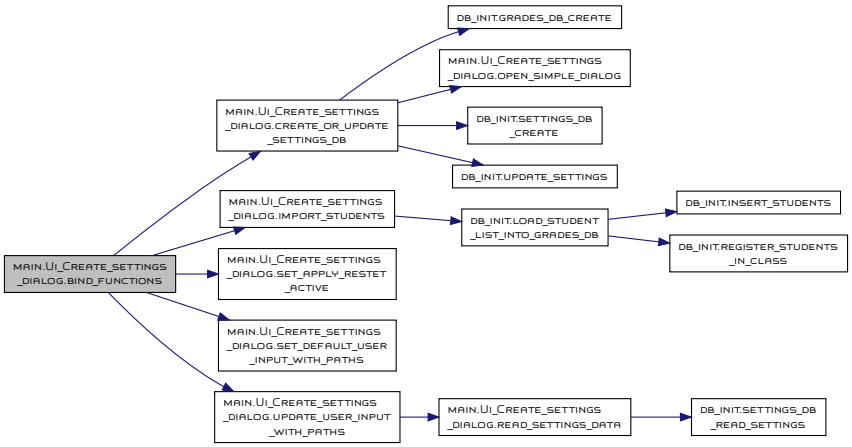
7.10.2 Member Function Documentation

7.10.2.1 bind_functions() def main.Ui_Create_settings_dialog.bind_functions (self)

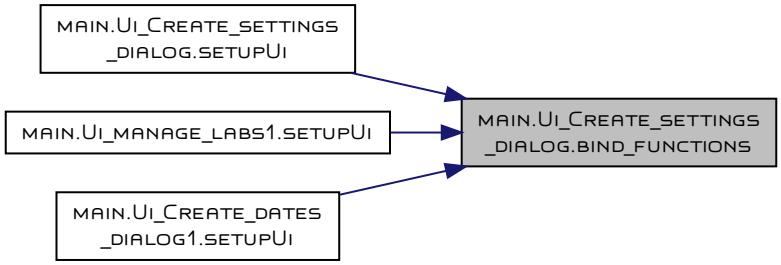
Definition at line 1347 of file main.py.

```
01347         self.buttonBox.button(self.buttonBox.RestoreDefaults).clicked.connect(self.set_default_user_input_with_paths)
01348         self.buttonBox.button(self.buttonBox.Apply).clicked.connect(self.create_or_update_settings_db)
01349         self.buttonBox.button(self.buttonBox.Ok).clicked.connect(self.create_or_update_settings_db)
01350 
01351         self.import_stuents_btn.clicked.connect(self.import_students)
01352 
01353         # TODO: make 'personal' events and update only fields that have been changed
01354         self.input_logisim_path.textChanged.connect(self.set_apply_restet_active)
01355         self.input_local_stor.textChanged.connect(self.set_apply_restet_active)
01356         self.input_rem_stor.textChanged.connect(self.set_apply_restet_active)
01357         self.input_grader_name.textChanged.connect(self.set_apply_restet_active)
01358         self.spin_year.valueChanged.connect(self.set_apply_restet_active)
01359         self.semester_comboBox.currentIndexChanged.connect(self.set_apply_restet_active)
01360         self.style_checkBox.stateChanged.connect(self.set_apply_restet_active)
01361         self.sync_command.textChanged.connect(self.set_apply_restet_active)
01362         self.input_grades_db.textChanged.connect(self.set_apply_restet_active)
01363 
01364     def setupUi(self, Settings):
01365         """
```

References dates_window.Ui_dates_window.buttonBox, create_dates_diag.Ui_Create_dates_dialog.buttonBox, settings.Ui_Settings.buttonBox, main.Ui_Create_settings_dialog.create_or_update_settings_db(), main.Ui_Create_settings_dialog.import_students(), settings.Ui_Settings.import_stuents_btn, settings.Ui_Settings.input_grader_name, settings.Ui_Settings.input_grades_db, settings.Ui_Settings.input_local_stor, settings.Ui_Settings.input_logisim_path, settings.Ui_Settings.input_rem_stor, settings.Ui_Settings.semester_comboBox, main.Ui_Create_settings_dialog.set_apply_restet_active(), main.Ui_Create_settings_dialog.set_default_user_input_with_paths(), settings.Ui_Settings.spin_year, settings.Ui_Settings.style_checkBox, settings.Ui_Settings.sync_command, and main.Ui_Create_settings_dialog.update_user_input_with_paths().
Referenced by main.Ui_Create_settings_dialog.setupUi(), main.Ui_manage_labs1.setupUi(), and main.Ui_Create_dates_dialog1.setupUi().
Here is the call graph for this function:



Here is the caller graph for this function:



7.10.2.2 create_or_update_settings_db()

def main.Ui_Create_settings_dialog.create_or_update_settings_db (self)

Definition at line 1439 of file main.py.

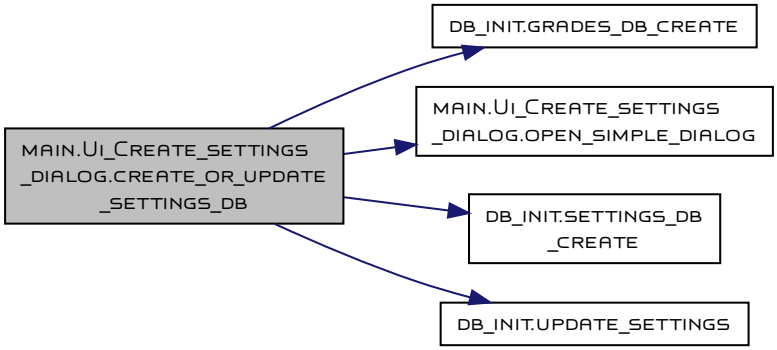
```
01439         settings_location = str(Path(os.path.expandvars(os.path.expanduser('./settings.sqlite3'))).absolute())
01440         if not os.path.isfile(settings_location):
01441             if self.open_simple_dialog("Do you want to create settings database ?"):
01442                 if not settings_db_create(force=True):
01443                     raise Exception('Was not able to create SETTINGS db.')
01444         if len(self.input_local_stor.text()) > 0:
01445             if self.input_local_stor.text()[-1] != '/':
01446                 self.input_local_stor.setText(self.input_local_stor.text() + '/')
01447         if len(self.input_rem_stor.text()) > 0:
01448             if self.input_rem_stor.text()[-1] != '/':
01449                 self.input_rem_stor.setText(self.input_rem_stor.text() + '/')
01450         if len(self.input_logisim_path.text()) > 0:
01451             if self.input_logisim_path.text()[-1] != '/':
01452                 self.input_logisim_path.setText(self.input_logisim_path.text() + '/')
01453 
01454 
01455         paths = (self.input_logisim_path.text(), self.input_local_stor.text(), self.input_rem_stor.text(),
01456                  self.input_grades_db.text())
01457         if os.path.isfile(settings_location):
01458             local = (self.input_grader_name.text(), int(self.spin_year.text()),
01459                     self.semester_comboBox.currentIndex(), self.style_checkBox.checkState(), self.sync_command.text())
01460             if len(self.input_local_stor.text()) > 0:
01461                 local_stor = str(Path(os.path.expanduser(os.path.expandvars(self.input_local_stor.text()))).absolute())
01462                 if local_stor[-1] != '/':
01463                     local_stor += '/'
01464                 if not os.path.isdir(local_stor):
01465                     os.mkdir(local_stor)
01466                 local_grading_path = local_stor + self.spin_year.text() + '_' +\
01467                                      str(self.semester_comboBox.currentIndex())
01468                 if not os.path.isdir(local_grading_path):
01469                     os.mkdir(local_grading_path)
01470             update_settings(paths, local)
01471 
01472 
01473         grades_location = str(Path(os.path.expandvars(os.path.expanduser(self.input_grades_db.text()))).absolute())
01474         if len(self.input_grades_db.text()) > 1 and not os.path.isfile(grades_location):
01475             if self.open_simple_dialog("Do you want to create GRADES database ?"):
01476                 print('Before grades creation.')
01477                 if not grades_db_create(grades_location, force=True):
01478                     raise Exception('Was not able to create GRADES db.')
01479 
01480         if os.path.isfile(settings_location) and os.path.isfile(grades_location):
01481             self.buttonBox.button(self.buttonBox.Apply).setDisabled(True)
01482             self.buttonBox.button(self.buttonBox.Apply).repaint()
01483             self.buttonBox.button(self.buttonBox.Reset).setDisabled(True)
01484             self.buttonBox.button(self.buttonBox.Reset).repaint()
01485             if not self.groupBox_user.isEnabled():
01486                 self.groupBox_user.setEnabled(True)
01487             if not self.input_logisim_path.isEnabled():
01488                 self.input_logisim_path.setEnabled(True)
01489                 self.label_logisim_path.setEnabled(True)
```

```
01490             if not self.input_local_stor.isEnabled():
01491                 self.input_local_stor.setEnabled(True)
01492                 self.label_local_stor.setEnabled(True)
01493             if not self.input_rem_stor.isEnabled():
01494                 self.input_rem_stor.setEnabled(True)
01495                 self.label_rem_stor.setEnabled(True)
01496             if not self.spin_year.isEnabled():
01497                 self.spin_year.setEnabled(True)
01498             if not self.semester_comboBox.isEnabled():
01499                 self.semester_comboBox.setEnabled(True)
01500             if not self.style_checkBox.isEnabled():
01501                 self.style_checkBox.setEnabled(True)
01502             if not self.input_grader_name.isEnabled():
01503                 self.input_grader_name.setEnabled(True)
01504             if not self.sync_command.isEnabled():
01505                 self.sync_command.setEnabled(True)
01506 
01507         # if len(self.input_local_stor.text()) > 1:
01508         #     full_path = Path(self.input_local_stor.text()).absolute()
01509         #     if not os.path.exists(full_path) or not os.path.isdir(full_path):
01510         #         os.makedirs(full_path)
01511 
01512     def import_students(self):
01513         """
```

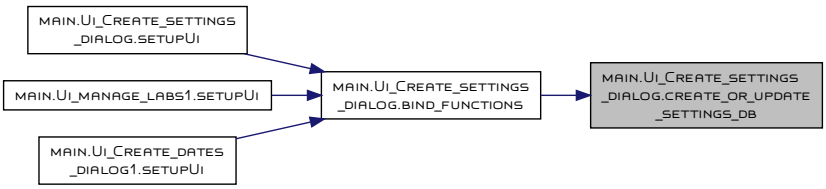
References dates_window.Ui_dates_window.buttonBox, create_dates_diag.Ui_Create_dates_dialog.buttonBox, settings.Ui_Settings.buttonBox, db_init.grades_db_create(), settings.Ui_Settings.groupBox_user, settings.Ui_Settings.input_grader_name, settings.Ui_Settings.input_grades_db, settings.Ui_Settings.input_local_stor, settings.Ui_Settings.input_logisim_path, settings.Ui_Settings.input_rem_stor, settings.Ui_Settings.label_local_stor, settings.Ui_Settings.label_logisim_path, settings.Ui_Settings.label_rem_stor, main.Ui_Create_settings_dialog.open_simple_dialog(), settings.Ui_Settings.semester_comboBox, db_init.settings_db_create(), settings.Ui_Settings.spin_year, settings.Ui_Settings.style_checkBox, settings.Ui_Settings.sync_command, and db_init.update_settings().

Referenced by main.Ui_Create_settings_dialog.bind_functions().

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.2.3 import_students()

def main.Ui_Create_settings_dialog.import_students (self)

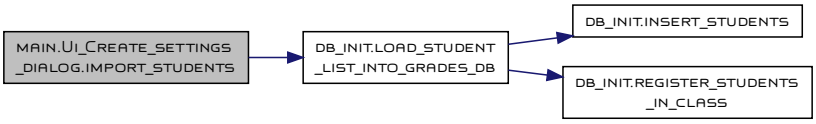
Definition at line 1518 of file main.py.

```
01518         stud_file = QFileDialog.getOpenFileName(caption="Select file with students' info", directory='.', filter="Text files (*.txt)")
01519         if len(stud_file[0]) > 3:
01520             load_student_list_into_grades_db(self.input_grades_db.text(), self.spin_year.value(), self.semester_comboBox.currentIndex(),
      filename=stud_file[0])
01521
01522
01523         self.import_stuents_btn.setEnabled(True)
01524
01525     def set_apply_restet_active(self):
01526         """
```

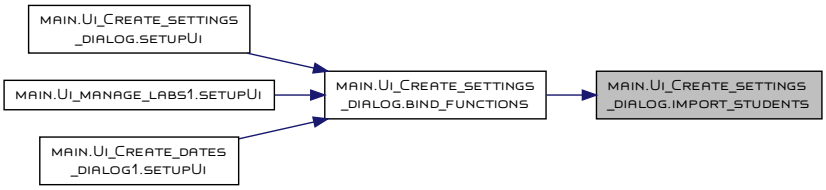
References settings.Ui_Settings.import_stuents_btn, settings.Ui_Settings.input_grades_db, db_init.load_student_list_into_grades_db(), settings.Ui_Settings.semester_comboBox, and settings.Ui_Settings.spin_year.

Referenced by main.Ui_Create_settings_dialog.bind_functions().

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.2.4 open_simple_dialog()

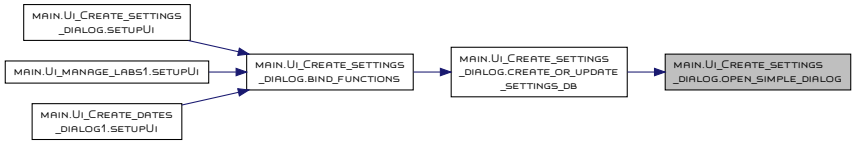
def main.Ui_Create_settings_dialog.open_simple_dialog (self, phrase)

Definition at line 1543 of file main.py.

```
01543         dui = SimpleDialog()
01544         dui.setupUi(self.simple_diag, phrase)
01545
01546         self.buttonBox.setDisabled(True)
01547         self.buttonBox.repaint()
01548
01549         self.simple_diag.setWindowTitle('Settings confirmation')
01550         self.simple_diag.show()
01551
01552         result = self.simple_diag.exec_()
01553
01554         self.buttonBox.setEnabled(True)
01555
01556         return result
01557
01558
01559 class SimpleDialog(Ui_Dialog):
01560     """
```

Referenced by main.Ui_Create_settings_dialog.create_or_update_settings_db().

Here is the caller graph for this function:



7.10.2.5 read_settings_data() def main.Ui_Create_settings_dialog.read_settings_data (self)

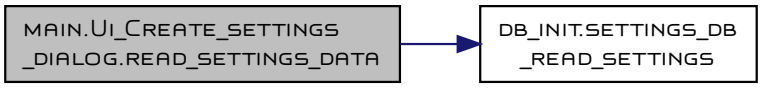
Definition at line 1428 of file main.py.

```
01428         return settings_db_read_settings()
01429
01430     def create_or_update_settings_db(self):
01431         """
```

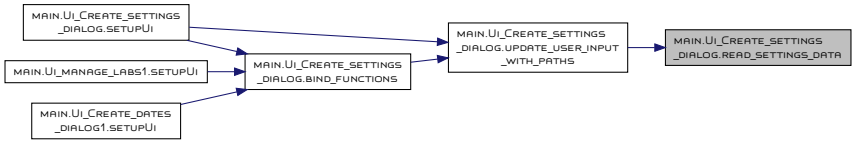
References dates_window.Ui_dates_window.buttonBox, create_dates_diag.Ui_Create_dates_dialog.buttonBox, settings.Ui_Settings.buttonBox, and db_init.settings_db_read_settings().

Referenced by main.Ui_Create_settings_dialog.update_user_input_with_paths().

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.2.6 set_apply_restet_active() def main.Ui_Create_settings_dialog.set_apply_restet_active (self)

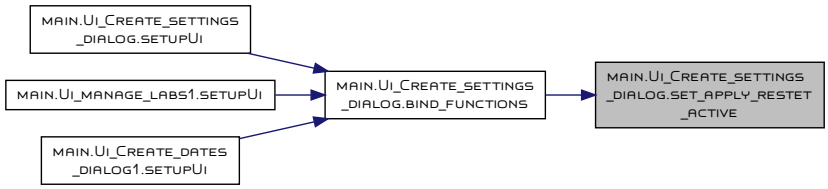
Definition at line 1531 of file main.py.

```
01531         self.buttonBox.button(self.buttonBox.Apply).setEnabled(True)
01532
01533     def __dummy(self):
01534         print('dummy exec')
```

References dates_window.Ui_dates_window.buttonBox, create_dates_diag.Ui_Create_dates_dialog.buttonBox, and settings.Ui_Settings.buttonBox.

Referenced by main.Ui_Create_settings_dialog.bind_functions().

Here is the caller graph for this function:



7.10.2.7 set_default_user_input_with_paths()

```
def main.Ui_Create_settings_dialog.set_default_user_input_with_paths (
        self )
```

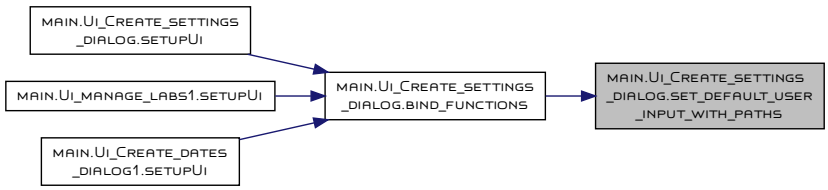
Definition at line 1415 of file main.py.

```
01415         self.input_local_stor.setText(" /Documents/3130_labs/")
01416         self.input_grades_db.setText(" /Documents/3130_labs/grades.sqlite3")
01417         self.input_rem_stor.setText("")  # impossible to predict
01418         self.groupBox_user.setEnabled(True)
01419         self.buttonBox.button(self.buttonBox.Reset).setEnabled(True)
01420         self.buttonBox.button(self.buttonBox.Apply).setEnabled(True)
01421
01422     def read_settings_data(self):
01423         """
```

References dates_window.Ui_dates_window.buttonBox, create_dates_diag.Ui_Create_dates_dialog.buttonBox, settings.Ui_Settings.buttonBox, settings.Ui_Settings.groupBox_user, settings.Ui_Settings.input_grades_db, settings.Ui_Settings.input_local_stor, settings.Ui_Settings.input_logisim_path, and settings.Ui_Settings.input_rem_stor.

Referenced by main.Ui_Create_settings_dialog.bind_functions().

Here is the caller graph for this function:



7.10.2.8 setupUi()

```
def main.Ui_Create_settings_dialog.setupUi (
        self,
        Settings )
```

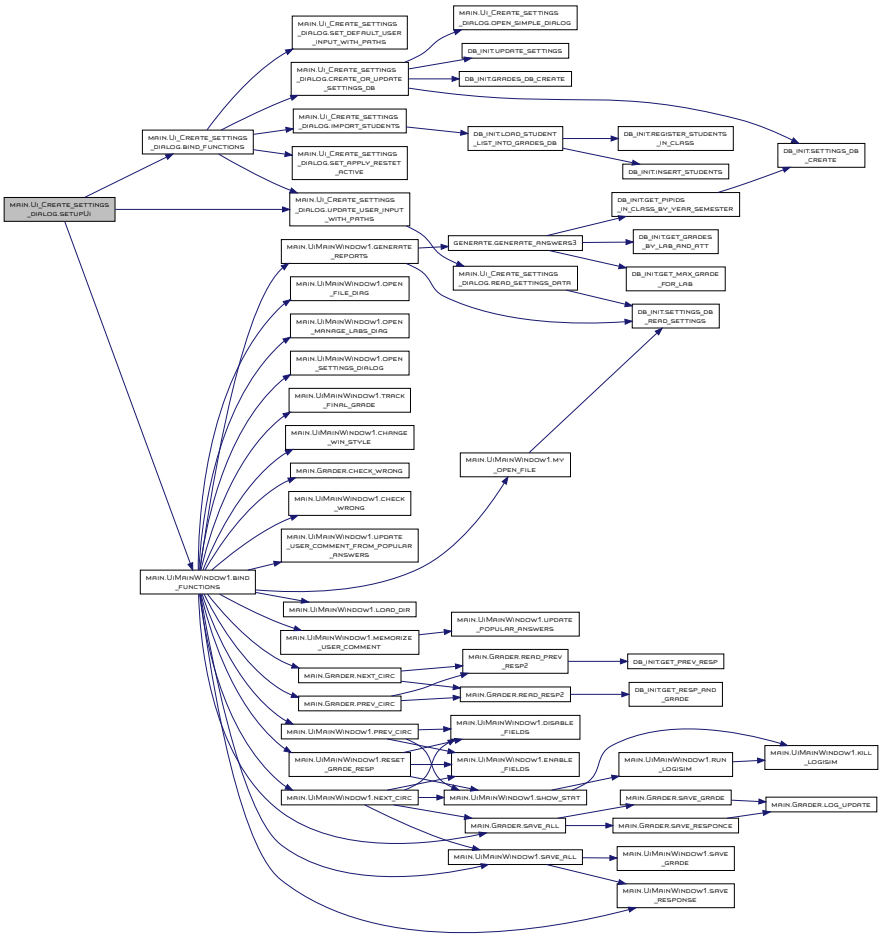
Reimplemented from settings.Ui_Settings.

Definition at line 1370 of file main.py.

```
01370         self.buttonBox.button(self.buttonBox.Reset).setDisabled(True)
01371         self.buttonBox.button(self.buttonBox.Apply).setDisabled(True)
01372         self.bind_functions()
01373         self.update_user_input_with_paths()
01374
01375     def update_user_input_with_paths(self):
01376         """
```

References main.UiMainWindow1.bind_functions(), main.Ui_Create_settings_dialog.bind_functions(), dates_window.Ui_dates_window.buttonBox, create_dates_diag.Ui_Create_dates_dialog.buttonBox, settings.Ui_Settings.buttonBox, and main.Ui_Create_settings_dialog.update_user_input_with_paths().

Here is the call graph for this function:



7.10.2.9 update_user_input_with_paths() def main.Ui_Create_settings_dialog.update_user_input_with_paths (self)

Reads settings parameters from DB and sets appropriate fields with obtained values.

Warning

dependa on a number of settings obtained from read_settings_data :return: Nothing

Definition at line 1382 of file main.py.

```
01382         if paths and len(paths) >= 4:
01383             self.input_logisim_path.setText(paths[0])
01384             self.input_local_stor.setText(paths[1])
01385             self.input_rem_stor.setText(paths[2])
01386             self.input_grades_db.setText(paths[3])
01387             self.groupBox_user.setEnabled(True)
01388 
01389         if local and len(local) >= 4:
01390             self.input_grader_name.setText(local[0])
01391             self.spin_year.setValue(local[1])
01392             self.semester_comboBox.setCurrentIndex(int(local[2]))
01393             self.style_checkBox.setChecked(bool(local[3]))
01394             self.sync_command.setText(local[4])
01395 
01396         if (paths and len(paths) >= 4 ) and (local and len(local) >= 4):
01397             self.spin_year.setEnabled(True)
01398             self.semester_comboBox.setEnabled(True)
01399             self.style_checkBox.setEnabled(True)
01400             self.input_grader_name.setEnabled(True)
```

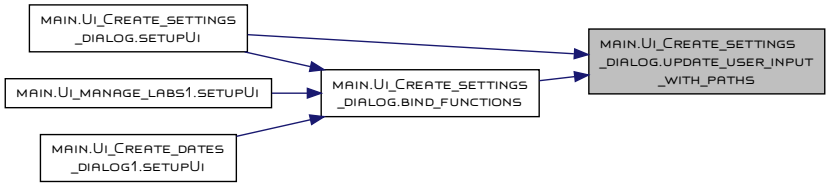
```
01401             self.sync_command.setEnabled(True)
01402         # if (local and len(local) > 5) or len(paths):
01403         #         print('Obtained more settings than expected. Please check Ui_Create_settings_dialog.')
01404 
01405         self.buttonBox.button(self.buttonBox.Reset).setDisabled(True)
01406         self.buttonBox.button(self.buttonBox.Apply).setDisabled(True)
01407 
01408     def set_default_user_input_with_paths(self):
01409         """
```

References dates_window.Ui_dates_window.buttonBox, create_dates_diag.Ui_Create_dates_dialog.buttonBox, settings.Ui_Settings.buttonBox, settings.Ui_Settings.groupBox_user, settings.Ui_Settings.input_grader_name, settings.Ui_Settings.input_grades_db, settings.Ui_Settings.input_local_stor, settings.Ui_Settings.input_logisim_path, settings.Ui_Settings.input_rem_stor, main.Ui_Create_settings_dialog.read_settings_data(), settings.Ui_Settings.semester_comboBox, settings.Ui_Settings.spin_year, settings.Ui_Settings.style_checkBox, and settings.Ui_Settings.sync_command.

Referenced by main.Ui_Create_settings_dialog.bind_functions(), and main.Ui_Create_settings_dialog.setupUi().

Here is the call graph for this function:

Here is the caller graph for this function:



7.10.3 Member Data Documentation

7.10.3.1 simple_diag main.Ui_Create_settings_dialog.simple_diag

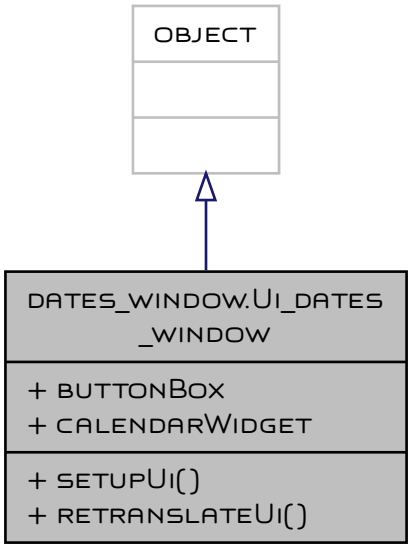
Definition at line 1544 of file main.py.

The documentation for this class was generated from the following file:

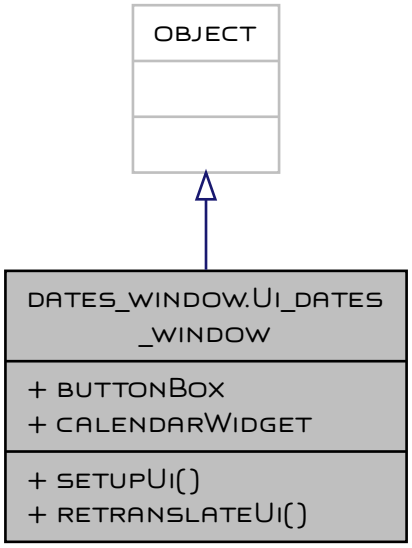
- main.py

7.11 dates_window.Ui_dates_window Class Reference

Inheritance diagram for dates_window.Ui_dates_window:



Collaboration diagram for dates_window.Ui_dates_window:



Public Member Functions

- def setupUi (self, dates_window)
- def retranslateUi (self, dates_window)

Public Attributes

- buttonBox
- calendarWidget

7.11.1 Detailed Description

Definition at line 11 of file dates_window.py.

7.11.2 Member Function Documentation

7.11.2.1 retranslateUi() def dates_window.Ui_dates_window.retranslateUi (self, dates_window)

Definition at line 29 of file dates_window.py.

```
00029     def retranslateUi(self, dates_window):
00030         _translate = QtCore.QCoreApplication.translate
00031         dates_window.setWindowTitle(_translate("dates_window", "Check dates"))
00032         self.calendarWidget.setAccessibleName(_translate("dates_window", "cal_diag"))
00033
```

References dates_window.Ui_dates_window.calendarWidget.

7.11.2.2 setupUi() def dates_window.Ui_dates_window.setupUi (self, dates_window)

Definition at line 12 of file dates_window.py.

```
00012     def setupUi(self, dates_window):
00013         dates_window.setObjectName("dates_window")
00014         dates_window.resize(251, 314)
00015         self.buttonBox = QtWidgets.QDialogButtonBox(dates_window)
00016         self.buttonBox.setGeometry(QtCore.QRect(40, 260, 191, 32))
00017         self.buttonBox.setOrientation(QtCore.Qt.Horizontal)
00018         self.buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Ok)
00019         self.buttonBox.setObjectName("buttonBox")
00020         self.calendarWidget = QtWidgets.QCalendarWidget(dates_window)
00021         self.calendarWidget.setGeometry(QtCore.QRect(10, 10, 224, 232))
00022         self.calendarWidget.setObjectName("calendarWidget")
00023
00024         self.retranslateUi(dates_window)
00025         self.buttonBox.accepted.connect(dates_window.accept)
00026         self.buttonBox.rejected.connect(dates_window.reject)
00027         QtCore.QMetaObject.connectSlotsByName(dates_window)
00028
```

7.11.3 Member Data Documentation

7.11.3.1 buttonBox dates_window.Ui_dates_window.buttonBox

Definition at line 15 of file dates_window.py.

Referenced by main.Ui_Create_settings_dialog.bind_functions(), main.Ui_Create_settings_dialog.create_or_update_settings_db(), main.Ui_Create_settings_dialog.read_settings_data(), main.Ui_Create_settings_dialog.set_apply_restet_active(), main.Ui_Create_settings_dialog.set_default_user_input_with_paths(), main.Ui_Create_settings_dialog.setupUi(), and main.Ui_Create_settings_dialog.update_user_input_with_paths().

7.11.3.2 calendarWidget dates_window.Ui_dates_window.calendarWidget

Definition at line 20 of file dates_window.py.

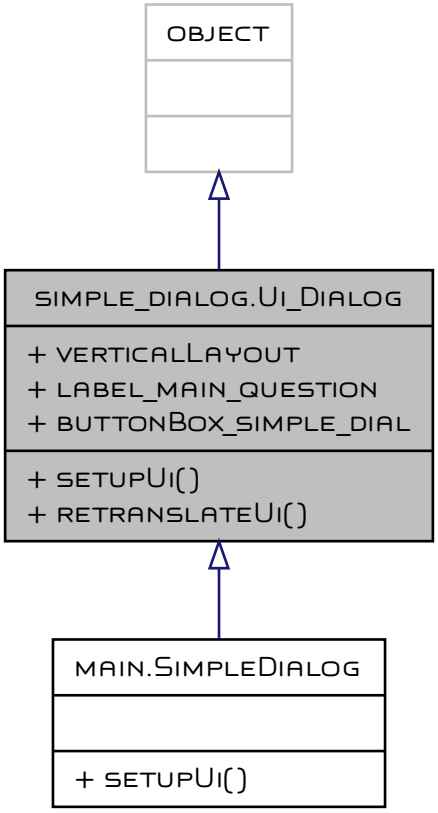
Referenced by dates_window.Ui_dates_window.retranslateUi().

The documentation for this class was generated from the following file:

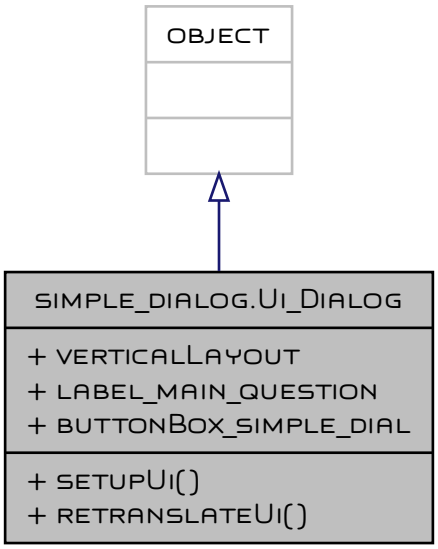
- dates_window.py

7.12 simple_dialog.Ui_Dialog Class Reference

Inheritance diagram for simple_dialog.Ui_Dialog:



Collaboration diagram for simple_dialog.Ui_Dialog:



Public Member Functions

- def setupUi (self, Dialog)
- def retranslateUi (self, Dialog)

Public Attributes

- verticalLayout
- label_main_question
- buttonBox_simple_dial

7.12.1 Detailed Description

Definition at line 11 of file simple_dialog.py.

7.12.2 Member Function Documentation

7.12.2.1 retranslateUi() def simple_dialog.Ui_Dialog.retranslateUi (self, Dialog)

Definition at line 46 of file simple_dialog.py.

```
00046     def retranslateUi(self, Dialog):
00047         _translate = QtCore.QCoreApplication.translate
00048         Dialog.setWindowTitle(_translate("Dialog", "Create database ?"))
00049         self.label_main_question.setText(_translate("Dialog", "Database will be created.  Confirm.."))
00050
```

References simple_dialog.Ui_Dialog.label_main_question.

7.12.2.2 setupUi() def simple_dialog.Ui_Dialog.setupUi (self, Dialog)

Definition at line 12 of file simple_dialog.py.

```
00012     def setupUi(self, Dialog):
00013         Dialog.setObjectName("Dialog")
00014         Dialog.resize(328, 76)
00015         icon = QtGui.QIcon()
```

```
00016         icon.addPixmap(QtGui.QPixmap("os_linux_1.ico"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
00017         Dialog.setWindowIcon(icon)
00018         Dialog.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00019         self.verticalLayout = QtWidgets.QVBoxLayout(Dialog)
00020         self.verticalLayout.setObjectName("verticalLayout")
00021         self.label_main_question = QtWidgets.QLabel(Dialog)
00022         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
00023         sizePolicy.setHorizontalStretch(0)
00024         sizePolicy.setVerticalStretch(0)
00025         sizePolicy.setHeightForWidth(self.label_main_question.sizePolicy().hasHeightForWidth())
00026         self.label_main_question.setSizePolicy(sizePolicy)
00027         self.label_main_question.setAlignment(QtCore.Qt.AlignCenter)
00028         self.label_main_question.setObjectName("label_main_question")
00029         self.verticalLayout.addWidget(self.label_main_question)
00030         self.buttonBox_simple_dial = QtWidgets.QDialogButtonBox(Dialog)
00031         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)
00032         sizePolicy.setHorizontalStretch(0)
00033         sizePolicy.setVerticalStretch(0)
00034         sizePolicy.setHeightForWidth(self.buttonBox_simple_dial.sizePolicy().hasHeightForWidth())
00035         self.buttonBox_simple_dial.setSizePolicy(sizePolicy)
00036         self.buttonBox_simple_dial.setOrientation(QtCore.Qt.Horizontal)
00037         self.buttonBox_simple_dial.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Ok)
00038         self.buttonBox_simple_dial.setObjectName("buttonBox_simple_dial")
00039         self.verticalLayout.addWidget(self.buttonBox_simple_dial)
00040
00041         self.retranslateUi(Dialog)
00042         self.buttonBox_simple_dial.accepted.connect(Dialog.accept)
00043         self.buttonBox_simple_dial.rejected.connect(Dialog.reject)
00044         QtCore.QMetaObject.connectSlotsByName(Dialog)
00045
```

7.12.3 Member Data Documentation

7.12.3.1 buttonBox_simple_dial simple_dialog.Ui_Dialog.buttonBox_simple_dial

Definition at line 30 of file simple_dialog.py.

7.12.3.2 label_main_question simple_dialog.Ui_Dialog.label_main_question

Definition at line 21 of file simple_dialog.py.
Referenced by simple_dialog.Ui_Dialog.retranslateUi(), and main.SimpleDialog.setupUi().

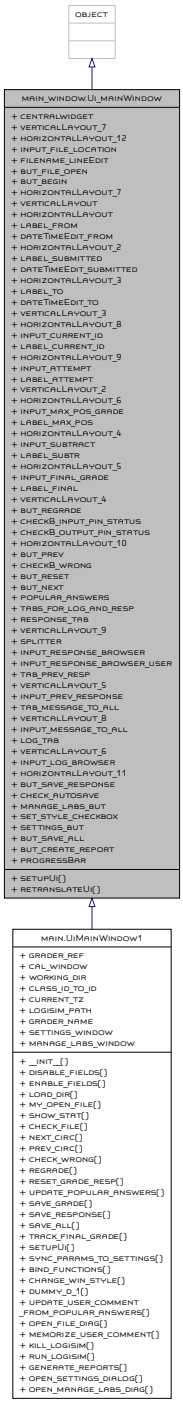
7.12.3.3 verticalLayout simple_dialog.Ui_Dialog.verticalLayout

Definition at line 19 of file simple_dialog.py.
The documentation for this class was generated from the following file:

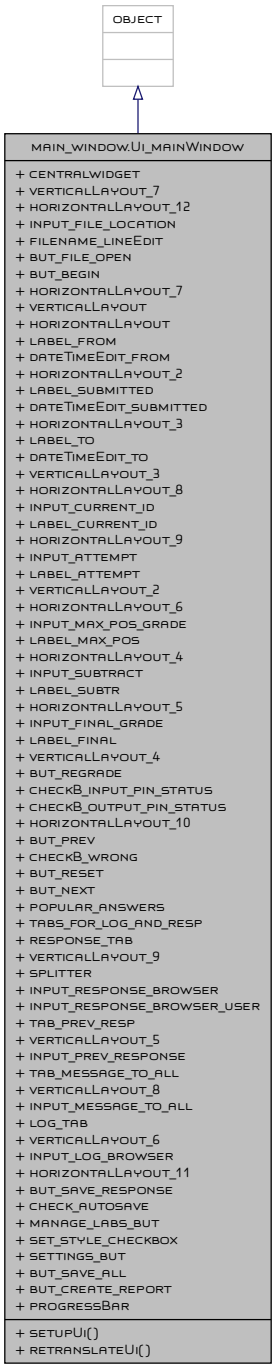
- simple_dialog.py

7.13 main_window.Ui_mainWindow Class Reference

Inheritance diagram for main_window.Ui_mainWindow:



Collaboration diagram for main_window.Ui_mainWindow:



Public Member Functions

- def setupUi (self, mainWindow)
- def retranslateUi (self, mainWindow)

Public Attributes

- centralwidget
- verticalLayout_7
- horizontalLayout_12
- input_file_location
- filename_lineEdit
- but_file_open
- but_begin
- horizontalLayout_7
- verticalLayout
- horizontalLayout
- label_from
- dateTimeEdit_from
- horizontalLayout_2
- label_submitted
- dateTimeEdit_submitted
- horizontalLayout_3
- label_to
- dateTimeEdit_to
- verticalLayout_3
- horizontalLayout_8
- input_current_id
- label_current_id
- horizontalLayout_9
- input_attempt
- label_attempt
- verticalLayout_2
- horizontalLayout_6
- input_max_pos_grade
- label_max_pos
- horizontalLayout_4
- input_subtract
- label_subtr
- horizontalLayout_5
- input_final_grade
- label_final
- verticalLayout_4
- but_regrade
- checkB_input_pin_status
- checkB_output_pin_status
- horizontalLayout_10
- but_prev
- checkB_wrong
- but_reset
- but_next
- popular_answers
- tabs_for_log_and_resp
- response_tab
- verticalLayout_9
- splitter
- input_response_browser
- input_response_browser_user
- tab_prev_resp
- verticalLayout_5
- input_prev_response
- tab_message_to_all
- verticalLayout_8
- input_message_to_all
- log_tab
- verticalLayout_6
- input_log_browser
- horizontalLayout_11
- but_save_response
- check_autosave
- manage_labs_but
- set_style_checkbox
- settings_but
- but_save_all
- but_create_report
- progressBar

7.13.1 Detailed Description

Definition at line 11 of file main_window.py.

7.13.2 Member Function Documentation

7.13.2.1 retranslateUi() def main_window.Ui_mainWindow.retranslateUi (self, mainWindow)

Definition at line 351 of file main_window.py.

```
00351     def retranslateUi(self, mainWindow):
00352         _translate = QtCore.QCoreApplication.translate
00353         mainWindow.setWindowTitle(_translate("mainWindow", "CSCI3130 grader"))
00354         self.input_file_location.setPlaceholderText(_translate("mainWindow", "Double click for path selection or paste|type path here"))
00355         self.but_file_open.setText(_translate("mainWindow", "Open"))
00356         self.but_begin.setText(_translate("mainWindow", "Begin"))
00357         self.label_from.setText(_translate("mainWindow", "From"))
00358         self.label_submitted.setText(_translate("mainWindow", "Submitted"))
00359         self.label_to.setText(_translate("mainWindow", "To"))
00360         self.label_current_id.setText(_translate("mainWindow", "current id"))
00361         self.label_attempt.setText(_translate("mainWindow", "attempt"))
00362         self.label_max_pos.setText(_translate("mainWindow", "lab max grade"))
00363         self.label_subtr.setText(_translate("mainWindow", "subtract"))
00364         self.label_final.setText(_translate("mainWindow", "final grade"))
00365         self.but_regrade.setText(_translate("mainWindow", "GRADE"))
00366         self.checkB_input_pin_status.setText(_translate("mainWindow", "Input direction"))
00367         self.checkB_output_pin_status.setText(_translate("mainWindow", "Output direction"))
00368         self.but_prev.setText(_translate("mainWindow", "prev"))
00369         self.checkB_wrong.setText(_translate("mainWindow", "WRONG"))
00370         self.but_reset.setText(_translate("mainWindow", "Reset"))
00371         self.but_next.setText(_translate("mainWindow", "next"))
00372         self.input_response_browser.setPlaceholderText(_translate("mainWindow", "Auto answer"))
00373         self.input_response_browser_user.setPlaceholderText(_translate("mainWindow", "User comment"))
00374         self.tabs_for_log_and_resp.setTabText(self.tabs_for_log_and_resp.indexOf(self.response_tab), _translate("mainWindow", "Response"))
00375         self.tabs_for_log_and_resp.setTabText(self.tabs_for_log_and_resp.indexOf(self.tab_prev_resp), _translate("mainWindow", "Previous
      Response"))
00376         self.tabs_for_log_and_resp.setTabText(self.tabs_for_log_and_resp.indexOf(self.tab_message_to_all), _translate("mainWindow", "Message to
      all"))
00377         self.tabs_for_log_and_resp.setTabText(self.tabs_for_log_and_resp.indexOf(self.log_tab), _translate("mainWindow", "Log"))
00378         self.but_save_response.setText(_translate("mainWindow", "save responce"))
00379         self.check_autosave.setText(_translate("mainWindow", "autosave"))
00380         self.manage_labs_but.setText(_translate("mainWindow", "Manage labs"))
00381         self.set_style_checkbox.setText(_translate("mainWindow", "style"))
00382         self.settings_but.setText(_translate("mainWindow", "Settings"))
00383         self.but_save_all.setText(_translate("mainWindow", "save all"))
00384         self.but_create_report.setText(_translate("mainWindow", "Create reports"))
00385         self.progressBar.setFormat(_translate("mainWindow", "%v/%m (%p%)"))
00386
00387 from qt_class_improvements import BetterLineEdit, BetterPlainTextEdit
```

References main_window.Ui_mainWindow.but_begin, main_window.Ui_mainWindow.but_create_report, main_window.Ui_mainWindow.but_file_open, main_window.Ui_mainWindow.but_next, main_window.Ui_mainWindow.but_prev, main_window.Ui_mainWindow.but_regrade, main_window.Ui_mainWindow.but_reset, main_window.Ui_mainWindow.but_save_all, main_window.Ui_mainWindow.but_save_response, main_window.Ui_mainWindow.check_autosave, main_window.Ui_mainWindow.checkB_input_pin_status, main_window.Ui_mainWindow.checkB_output_pin_status, main_window.Ui_mainWindow.checkB_wrong, main_window.Ui_mainWindow.input_file_location, main_window.Ui_mainWindow.input_response_browser, main_window.Ui_mainWindow.input_response_browser_user, main_window.Ui_mainWindow.label_attempt, main_window.Ui_mainWindow.label_current_id, main_window.Ui_mainWindow.label_final, main_window.Ui_mainWindow.label_from, main_window.Ui_mainWindow.label_max_pos, main_window.Ui_mainWindow.label_submitted, main_window.Ui_mainWindow.label_subtr, main_window.Ui_mainWindow.label_to, main_window.Ui_mainWindow.log_tab, main_window.Ui_mainWindow.manage_labs_but, main_window.Ui_mainWindow.progressBar, main_window.Ui_mainWindow.response_tab, main_window.Ui_mainWindow.set_style_checkbox, main_window.Ui_mainWindow.settings_but, main_window.Ui_mainWindow.tab_message_to_all, main_window.Ui_mainWindow.tab_prev_resp, and main_window.Ui_mainWindow.tabs_for_log_and_resp.

7.13.2.2 setupUi() def main_window.Ui_mainWindow.setupUi (self, mainWindow)

Reimplemented in main.UiMainWindow1.

Definition at line 12 of file main_window.py.

```
00012     def setupUi(self, mainWindow):
00013         mainWindow.setObjectName("mainWindow")
00014         mainWindow.setEnabled(True)
00015         mainWindow.resize(888, 584)
00016         icon = QtGui.QIcon()
00017         icon.addPixmap(QtGui.QPixmap("os_linux_1.ico"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
00018         mainWindow.setWindowIcon(icon)
00019         mainWindow.setAccessibleName("")
00020         self.centralwidget = QtWidgets.QWidget(mainWindow)
00021         self.centralwidget.setObjectName("centralwidget")
00022         self.verticalLayout_7 = QtWidgets.QVBoxLayout(self.centralwidget)
00023         self.verticalLayout_7.setObjectName("verticalLayout_7")
```

```
00024         self.horizontalLayout_12 = QtWidgets.QHBoxLayout()
00025         self.horizontalLayout_12.setObjectName("horizontalLayout_12")
00026         self.input_file_location = BetterLineEdit(self.centralwidget)
00027         self.input_file_location.setEnabled(False)
00028         self.input_file_location.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00029         self.input_file_location.setText("")
00030         self.input_file_location.setObjectName("input_file_location")
00031         self.horizontalLayout_12.addWidget(self.input_file_location)
00032         self.filename_lineEdit = QtWidgets.QLineEdit(self.centralwidget)
00033         self.filename_lineEdit.setMaximumSize(QtCore.QSize(90, 16777215))
00034         self.filename_lineEdit.setReadOnly(True)
00035         self.filename_lineEdit.setObjectName("filename_lineEdit")
00036         self.horizontalLayout_12.addWidget(self.filename_lineEdit)
00037         self.but_file_open = QtWidgets.QPushButton(self.centralwidget)
00038         self.but_file_open.setEnabled(False)
00039         self.but_file_open.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00040         self.but_file_open.setObjectName("but_file_open")
00041         self.horizontalLayout_12.addWidget(self.but_file_open)
00042         self.but_begin = QtWidgets.QPushButton(self.centralwidget)
00043         self.but_begin.setEnabled(False)
00044         self.but_begin.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00045         self.but_begin.setCheckable(False)
00046         self.but_begin.setAutoDefault(False)
00047         self.but_begin.setDefault(False)
00048         self.but_begin.setFlat(False)
00049         self.but_begin.setObjectName("but_begin")
00050         self.horizontalLayout_12.addWidget(self.but_begin)
00051         self.verticalLayout_7.addLayout(self.horizontalLayout_12)
00052         self.horizontalLayout_7 = QtWidgets.QHBoxLayout()
00053         self.horizontalLayout_7.setSpacing(6)
00054         self.horizontalLayout_7.setObjectName("horizontalLayout_7")
00055         self.verticalLayout = QtWidgets.QVBoxLayout()
00056         self.verticalLayout.setObjectName("verticalLayout")
00057         self.horizontalLayout = QtWidgets.QHBoxLayout()
00058         self.horizontalLayout.setObjectName("horizontalLayout")
00059         self.label_from = QtWidgets.QLabel(self.centralwidget)
00060         self.label_from.setObjectName("label_from")
00061         self.horizontalLayout.addWidget(self.label_from)
00062         self.dateTimeEdit_from = QtWidgets.QDateTimeEdit(self.centralwidget)
00063         self.dateTimeEdit_from.setEnabled(True)
00064         self.dateTimeEdit_from.setWrapping(False)
00065         self.dateTimeEdit_from.setReadOnly(True)
00066         self.dateTimeEdit_from.setAccelerated(False)
00067         self.dateTimeEdit_from.setCalendarPopup(True)
00068         self.dateTimeEdit_from.setObjectName("dateTimeEdit_from")
00069         self.horizontalLayout.addWidget(self.dateTimeEdit_from)
00070         self.verticalLayout.addLayout(self.horizontalLayout)
00071         self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
00072         self.horizontalLayout_2.setObjectName("horizontalLayout_2")
00073         self.label_submitted = QtWidgets.QLabel(self.centralwidget)
00074         self.label_submitted.setObjectName("label_submitted")
00075         self.horizontalLayout_2.addWidget(self.label_submitted)
00076         self.dateTimeEdit_submitted = QtWidgets.QDateTimeEdit(self.centralwidget)
00077         self.dateTimeEdit_submitted.setEnabled(True)
00078         self.dateTimeEdit_submitted.setWrapping(False)
00079         self.dateTimeEdit_submitted.setFrame(True)
00080         self.dateTimeEdit_submitted.setReadOnly(True)
00081         self.dateTimeEdit_submitted.setKeyboardTracking(False)
00082         self.dateTimeEdit_submitted.setCalendarPopup(True)
00083         self.dateTimeEdit_submitted.setObjectName("dateTimeEdit_submitted")
00084         self.horizontalLayout_2.addWidget(self.dateTimeEdit_submitted)
00085         self.verticalLayout.addLayout(self.horizontalLayout_2)
00086         self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
00087         self.horizontalLayout_3.setObjectName("horizontalLayout_3")
00088         self.label_to = QtWidgets.QLabel(self.centralwidget)
00089         self.label_to.setObjectName("label_to")
00090         self.horizontalLayout_3.addWidget(self.label_to)
00091         self.dateTimeEdit_to = QtWidgets.QDateTimeEdit(self.centralwidget)
00092         self.dateTimeEdit_to.setEnabled(True)
00093         self.dateTimeEdit_to.setReadOnly(True)
00094         self.dateTimeEdit_to.setCalendarPopup(True)
00095         self.dateTimeEdit_to.setObjectName("dateTimeEdit_to")
00096         self.horizontalLayout_3.addWidget(self.dateTimeEdit_to)
00097         self.verticalLayout.addLayout(self.horizontalLayout_3)
00098         self.horizontalLayout_7.addLayout(self.verticalLayout)
00099         self.verticalLayout_3 = QtWidgets.QVBoxLayout()
00100         self.verticalLayout_3.setObjectName("verticalLayout_3")
00101         self.horizontalLayout_8 = QtWidgets.QHBoxLayout()
00102         self.horizontalLayout_8.setObjectName("horizontalLayout_8")
00103         self.input_current_id = QtWidgets.QLineEdit(self.centralwidget)
00104         self.input_current_id.setEnabled(False)
```

```
00105         self.input_current_id.setMaximumSize(QtCore.QSize(60, 40))
00106         self.input_current_id.setReadOnly(True)
00107         self.input_current_id.setObjectName("input_current_id")
00108         self.horizontalLayout_8.addWidget(self.input_current_id)
00109         self.label_current_id = QtWidgets.QLabel(self.centralwidget)
00110         self.label_current_id.setObjectName("label_current_id")
00111         self.horizontalLayout_8.addWidget(self.label_current_id)
00112         self.verticalLayout_3.addLayout(self.horizontalLayout_8)
00113         self.horizontalLayout_9 = QtWidgets.QHBoxLayout()
00114         self.horizontalLayout_9.setObjectName("horizontalLayout_9")
00115         self.input_attempt = QtWidgets.QLineEdit(self.centralwidget)
00116         self.input_attempt.setEnabled(False)
00117         self.input_attempt.setMaximumSize(QtCore.QSize(40, 40))
00118         self.input_attempt.setReadOnly(True)
00119         self.input_attempt.setObjectName("input_attempt")
00120         self.horizontalLayout_9.addWidget(self.input_attempt)
00121         spacerItem = QtWidgets.QSpacerItem(20, 20, QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Minimum)
00122         self.horizontalLayout_9.addItem(spacerItem)
00123         self.label_attempt = QtWidgets.QLabel(self.centralwidget)
00124         self.label_attempt.setObjectName("label_attempt")
00125         self.horizontalLayout_9.addWidget(self.label_attempt)
00126         self.verticalLayout_3.addLayout(self.horizontalLayout_9)
00127         spacerItem1 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)
00128         self.verticalLayout_3.addItem(spacerItem1)
00129         self.horizontalLayout_7.addLayout(self.verticalLayout_3)
00130         self.verticalLayout_2 = QtWidgets.QVBoxLayout()
00131         self.verticalLayout_2.setObjectName("verticalLayout_2")
00132         self.horizontalLayout_6 = QtWidgets.QHBoxLayout()
00133         self.horizontalLayout_6.setObjectName("horizontalLayout_6")
00134         self.input_max_pos_grade = QtWidgets.QLineEdit(self.centralwidget)
00135         self.input_max_pos_grade.setEnabled(False)
00136         self.input_max_pos_grade.setMaximumSize(QtCore.QSize(40, 40))
00137         self.input_max_pos_grade.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00138         self.input_max_pos_grade.setText("")
00139         self.input_max_pos_grade.setReadOnly(True)
00140         self.input_max_pos_grade.setObjectName("input_max_pos_grade")
00141         self.horizontalLayout_6.addWidget(self.input_max_pos_grade)
00142         self.label_max_pos = QtWidgets.QLabel(self.centralwidget)
00143         self.label_max_pos.setEnabled(True)
00144         self.label_max_pos.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00145         self.label_max_pos.setObjectName("label_max_pos")
00146         self.horizontalLayout_6.addWidget(self.label_max_pos)
00147         self.verticalLayout_2.addLayout(self.horizontalLayout_6)
00148         self.horizontalLayout_4 = QtWidgets.QHBoxLayout()
00149         self.horizontalLayout_4.setObjectName("horizontalLayout_4")
00150         self.input_subtract = QtWidgets.QLineEdit(self.centralwidget)
00151         self.input_subtract.setEnabled(False)
00152         self.input_subtract.setMaximumSize(QtCore.QSize(40, 40))
00153         self.input_subtract.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00154         self.input_subtract.setReadOnly(True)
00155         self.input_subtract.setObjectName("input_subtract")
00156         self.horizontalLayout_4.addWidget(self.input_subtract)
00157         self.label_subtr = QtWidgets.QLabel(self.centralwidget)
00158         self.label_subtr.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00159         self.label_subtr.setObjectName("label_subtr")
00160         self.horizontalLayout_4.addWidget(self.label_subtr)
00161         self.verticalLayout_2.addLayout(self.horizontalLayout_4)
00162         self.horizontalLayout_5 = QtWidgets.QHBoxLayout()
00163         self.horizontalLayout_5.setObjectName("horizontalLayout_5")
00164         self.input_final_grade = QtWidgets.QLineEdit(self.centralwidget)
00165         self.input_final_grade.setEnabled(False)
00166         self.input_final_grade.setMaximumSize(QtCore.QSize(40, 40))
00167         self.input_final_grade.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00168         self.input_final_grade.setText("")
00169         self.input_final_grade.setReadOnly(True)
00170         self.input_final_grade.setObjectName("input_final_grade")
00171         self.horizontalLayout_5.addWidget(self.input_final_grade)
00172         self.label_final = QtWidgets.QLabel(self.centralwidget)
00173         self.label_final.setEnabled(True)
00174         self.label_final.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00175         self.label_final.setObjectName("label_final")
00176         self.horizontalLayout_5.addWidget(self.label_final)
00177         self.verticalLayout_2.addLayout(self.horizontalLayout_5)
00178         self.horizontalLayout_7.addLayout(self.verticalLayout_2)
00179         self.verticalLayout_4 = QtWidgets.QVBoxLayout()
00180         self.verticalLayout_4.setObjectName("verticalLayout_4")
00181         self.but_regrade = QtWidgets.QPushButton(self.centralwidget)
00182         self.but_regrade.setEnabled(False)
00183         self.but_regrade.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00184         self.but_regrade.setObjectName("but_regrade")
00185         self.verticalLayout_4.addWidget(self.but_regrade)
```

```
00186         self.checkB_input_pin_status = QtWidgets.QCheckBox(self.centralwidget)
00187         self.checkB_input_pin_status.setEnabled(False)
00188         self.checkB_input_pin_status.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00189         self.checkB_input_pin_status.setObjectName("checkB_input_pin_status")
00190         self.verticalLayout_4.addWidget(self.checkB_input_pin_status)
00191         self.checkB_output_pin_status = QtWidgets.QCheckBox(self.centralwidget)
00192         self.checkB_output_pin_status.setEnabled(False)
00193         self.checkB_output_pin_status.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00194         self.checkB_output_pin_status.setObjectName("checkB_output_pin_status")
00195         self.verticalLayout_4.addWidget(self.checkB_output_pin_status)
00196         self.horizontalLayout_7.addLayout(self.verticalLayout_4)
00197         self.verticalLayout_7.addLayout(self.horizontalLayout_7)
00198         self.horizontalLayout_10 = QtWidgets.QHBoxLayout()
00199         self.horizontalLayout_10.setSpacing(65)
00200         self.horizontalLayout_10.setObjectName("horizontalLayout_10")
00201         self.but_prev = QtWidgets.QPushButton(self.centralwidget)
00202         self.but_prev.setEnabled(False)
00203         self.but_prev.setMinimumSize(QtCore.QSize(60, 30))
00204         self.but_prev.setMaximumSize(QtCore.QSize(200, 16777215))
00205         self.but_prev.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00206         self.but_prev.setObjectName("but_prev")
00207         self.horizontalLayout_10.addWidget(self.but_prev)
00208         self.checkB_wrong = QtWidgets.QCheckBox(self.centralwidget)
00209         self.checkB_wrong.setEnabled(False)
00210         self.checkB_wrong.setMinimumSize(QtCore.QSize(80, 20))
00211         self.checkB_wrong.setMaximumSize(QtCore.QSize(75, 16777215))
00212         self.checkB_wrong.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00213         self.checkB_wrong.setObjectName("checkB_wrong")
00214         self.horizontalLayout_10.addWidget(self.checkB_wrong)
00215         self.but_reset = QtWidgets.QPushButton(self.centralwidget)
00216         self.but_reset.setEnabled(False)
00217         self.but_reset.setMinimumSize(QtCore.QSize(60, 20))
00218         self.but_reset.setMaximumSize(QtCore.QSize(90, 16777215))
00219         self.but_reset.setObjectName("but_reset")
00220         self.horizontalLayout_10.addWidget(self.but_reset)
00221         self.but_next = QtWidgets.QPushButton(self.centralwidget)
00222         self.but_next.setEnabled(False)
00223         self.but_next.setMinimumSize(QtCore.QSize(60, 30))
00224         self.but_next.setMaximumSize(QtCore.QSize(200, 16777215))
00225         self.but_next.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00226         self.but_next.setObjectName("but_next")
00227         self.horizontalLayout_10.addWidget(self.but_next)
00228         self.verticalLayout_7.addLayout(self.horizontalLayout_10)
00229         self.popular_answers = QtWidgets.QComboBox(self.centralwidget)
00230         self.popular_answers.setEnabled(False)
00231         self.popular_answers.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00232         self.popular_answers.setEditable(False)
00233         self.popular_answers.setCurrentText("")
00234         self.popular_answers.setObjectName("popular_answers")
00235         self.popular_answers.addItem("")
00236         self.popular_answers.setItemText(0, "")
00237         self.verticalLayout_7.addWidget(self.popular_answers)
00238         self.tabs_for_log_and_resp = QtWidgets.QTabWidget(self.centralwidget)
00239         self.tabs_for_log_and_resp.setEnabled(True)
00240         self.tabs_for_log_and_resp.setMinimumSize(QtCore.QSize(770, 30))
00241         self.tabs_for_log_and_resp.setMaximumSize(QtCore.QSize(20000, 3700))
00242         self.tabs_for_log_and_resp.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00243         self.tabs_for_log_and_resp.setTabShape(QtWidgets.QTabWidget.Rounded)
00244         self.tabs_for_log_and_resp.setObjectName("tabs_for_log_and_resp")
00245         self.response_tab = QtWidgets.QWidget()
00246         self.response_tab.setMinimumSize(QtCore.QSize(0, 180))
00247         self.response_tab.setMaximumSize(QtCore.QSize(16777215, 300))
00248         self.response_tab.setObjectName("response_tab")
00249         self.verticalLayout_9 = QtWidgets.QVBoxLayout(self.response_tab)
00250         self.verticalLayout_9.setObjectName("verticalLayout_9")
00251         self.splitter = QtWidgets.QSplitter(self.response_tab)
00252         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Expanding)
00253         sizePolicy.setHorizontalStretch(0)
00254         sizePolicy.setVerticalStretch(0)
00255         sizePolicy.setHeightForWidth(self.splitter.sizePolicy().hasHeightForWidth())
00256         self.splitter.setSizePolicy(sizePolicy)
00257         self.splitter.setOrientation(QtCore.Qt.Vertical)
00258         self.splitter.setObjectName("splitter")
00259         self.input_response_browser = QtWidgets.QPlainTextEdit(self.splitter)
00260         self.input_response_browser.setEnabled(True)
00261         self.input_response_browser.setMinimumSize(QtCore.QSize(0, 30))
00262         self.input_response_browser.setReadOnly(True)
00263         self.input_response_browser.setTextInteractionFlags(QtCore.Qt.TextSelectableByKeyboard|QtCore.Qt.TextSelectableByMouse)
00264         self.input_response_browser.setObjectName("input_response_browser")
00265         self.input_response_browser_user = BetterPlainTextEdit(self.splitter)
00266         self.input_response_browser_user.setEnabled(False)
```

```
00267         self.input_response_browser_user.setMinimumSize(QtCore.QSize(0, 30))
00268         self.input_response_browser_user.setObjectName("input_response_browser_user")
00269         self.verticalLayout_9.addWidget(self.splitter)
00270         self.tabs_for_log_and_resp.addTab(self.response_tab, "")
00271         self.tab_prev_resp = QtWidgets.QWidget()
00272         self.tab_prev_resp.setObjectName("tab_prev_resp")
00273         self.verticalLayout_5 = QtWidgets.QVBoxLayout(self.tab_prev_resp)
00274         self.verticalLayout_5.setObjectName("verticalLayout_5")
00275         self.input_prev_response = QtWidgets.QPlainTextEdit(self.tab_prev_resp)
00276         self.input_prev_response.setEnabled(True)
00277         self.input_prev_response.setTextInteractionFlags(QtCore.Qt.TextSelectableByKeyboard|QtCore.Qt.TextSelectableByMouse)
00278         self.input_prev_response.setObjectName("input_prev_response")
00279         self.verticalLayout_5.addWidget(self.input_prev_response)
00280         self.tabs_for_log_and_resp.addTab(self.tab_prev_resp, "")
00281         self.tab_message_to_all = QtWidgets.QWidget()
00282         self.tab_message_to_all.setObjectName("tab_message_to_all")
00283         self.verticalLayout_8 = QtWidgets.QVBoxLayout(self.tab_message_to_all)
00284         self.verticalLayout_8.setObjectName("verticalLayout_8")
00285         self.input_message_to_all = QtWidgets.QPlainTextEdit(self.tab_message_to_all)
00286         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Expanding)
00287         sizePolicy.setHorizontalStretch(0)
00288         sizePolicy.setVerticalStretch(0)
00289         sizePolicy.setHeightForWidth(self.input_message_to_all.sizePolicy().hasHeightForWidth())
00290         self.input_message_to_all.setSizePolicy(sizePolicy)
00291         self.input_message_to_all.setObjectName("input_message_to_all")
00292         self.verticalLayout_8.addWidget(self.input_message_to_all)
00293         self.tabs_for_log_and_resp.addTab(self.tab_message_to_all, "")
00294         self.log_tab = QtWidgets.QWidget()
00295         self.log_tab.setObjectName("log_tab")
00296         self.verticalLayout_6 = QtWidgets.QVBoxLayout(self.log_tab)
00297         self.verticalLayout_6.setObjectName("verticalLayout_6")
00298         self.input_log_browser = QtWidgets.QTextBrowser(self.log_tab)
00299         self.input_log_browser.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00300         self.input_log_browser.setObjectName("input_log_browser")
00301         self.verticalLayout_6.addWidget(self.input_log_browser)
00302         self.tabs_for_log_and_resp.addTab(self.log_tab, "")
00303         self.verticalLayout_7.addWidget(self.tabs_for_log_and_resp)
00304         self.horizontalLayout_11 = QtWidgets.QHBoxLayout()
00305         self.horizontalLayout_11.setObjectName("horizontalLayout_11")
00306         self.but_save_response = QtWidgets.QPushButton(self.centralwidget)
00307         self.but_save_response.setEnabled(False)
00308         self.but_save_response.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00309         self.but_save_response.setObjectName("but_save_response")
00310         self.horizontalLayout_11.addWidget(self.but_save_response)
00311         self.check_autosave = QtWidgets.QCheckBox(self.centralwidget)
00312         self.check_autosave.setEnabled(False)
00313         self.check_autosave.setObjectName("check_autosave")
00314         self.horizontalLayout_11.addWidget(self.check_autosave)
00315         self.manage_labs_but = QtWidgets.QPushButton(self.centralwidget)
00316         self.manage_labs_but.setEnabled(False)
00317         self.manage_labs_but.setObjectName("manage_labs_but")
00318         self.horizontalLayout_11.addWidget(self.manage_labs_but)
00319         self.set_style_checkbox = QtWidgets.QCheckBox(self.centralwidget)
00320         self.set_style_checkbox.setObjectName("set_style_checkbox")
00321         self.horizontalLayout_11.addWidget(self.set_style_checkbox)
00322         self.settings_but = QtWidgets.QToolButton(self.centralwidget)
00323         self.settings_but.setEnabled(True)
00324         self.settings_but.setObjectName("settings_but")
00325         self.horizontalLayout_11.addWidget(self.settings_but)
00326         self.but_save_all = QtWidgets.QPushButton(self.centralwidget)
00327         self.but_save_all.setEnabled(False)
00328         self.but_save_all.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00329         self.but_save_all.setObjectName("but_save_all")
00330         self.horizontalLayout_11.addWidget(self.but_save_all)
00331         self.but_create_report = QtWidgets.QPushButton(self.centralwidget)
00332         self.but_create_report.setEnabled(False)
00333         self.but_create_report.setObjectName("but_create_report")
00334         self.horizontalLayout_11.addWidget(self.but_create_report)
00335         self.verticalLayout_7.addLayout(self.horizontalLayout_11)
00336         self.progressBar = QtWidgets.QProgressBar(self.centralwidget)
00337         self.progressBar.setEnabled(True)
00338         self.progressBar.setAutoFillBackground(False)
00339         self.progressBar.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00340         self.progressBar.setProperty("value", 0)
00341         self.progressBar.setTextVisible(True)
00342         self.progressBar.setInvertedAppearance(False)
00343         self.progressBar.setObjectName("progressBar")
00344         self.verticalLayout_7.addWidget(self.progressBar)
00345         mainWindow.setCentralWidget(self.centralwidget)
00346 
00347         self.retranslateUi(mainWindow)
```

```
00348         self.tabs_for_log_and_resp.setCurrentIndex(0)
00349         QtCore.QMetaObject.connectSlotsByName(mainWindow)
00350
```

7.13.3 Member Data Documentation

7.13.3.1 but_begin main_window.Ui_mainWindow.but_begin

Definition at line 42 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), and main_window.Ui_mainWindow.retranslateUi().

7.13.3.2 but_create_report main_window.Ui_mainWindow.but_create_report

Definition at line 331 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), main.UiMainWindow1.generate_reports(), and main_window.Ui_mainWindow.retranslateUi().

7.13.3.3 but_file_open main_window.Ui_mainWindow.but_file_open

Definition at line 37 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), main_window.Ui_mainWindow.retranslateUi(), and main.UiMainWindow1.setupUi().

7.13.3.4 but_next main_window.Ui_mainWindow.but_next

Definition at line 221 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), main.UiMainWindow1.next_circ(), main.UiMainWindow1.prev_circ(), and main_window.Ui_mainWindow.retranslateUi().

7.13.3.5 but_prev main_window.Ui_mainWindow.but_prev

Definition at line 201 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), main.UiMainWindow1.next_circ(), main.UiMainWindow1.prev_circ(), and main_window.Ui_mainWindow.retranslateUi().

7.13.3.6 but_regrade main_window.Ui_mainWindow.but_regrade

Definition at line 181 of file main_window.py.
Referenced by main.UiMainWindow1.disable_fields(), main.UiMainWindow1.next_circ(), main.UiMainWindow1.prev_circ(), main.UiMainWindow1.regrade(), main_window.Ui_mainWindow.retranslateUi(), and main.UiMainWindow1.show_stat().

7.13.3.7 but_reset main_window.Ui_mainWindow.but_reset

Definition at line 215 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), and main_window.Ui_mainWindow.retranslateUi().

7.13.3.8 but_save_all main_window.Ui_mainWindow.but_save_all

Definition at line 326 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), and main_window.Ui_mainWindow.retranslateUi().

7.13.3.9 but_save_response main_window.Ui_mainWindow.but_save_response

Definition at line 306 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), and main_window.Ui_mainWindow.retranslateUi().

7.13.3.10 centralwidget main_window.Ui_mainWindow.centralwidget

Definition at line 20 of file main_window.py.
Referenced by main.UiMainWindow1.open_manage_labs_diag().

7.13.3.11 check_autosave main_window.Ui_mainWindow.check_autosave

Definition at line 311 of file main_window.py.
Referenced by main.UiMainWindow1.disable_fields(), main.UiMainWindow1.enable_fields(), main.UiMainWindow1.next_circ(), and main_window.Ui_mainWindow.retranslateUi().

7.13.3.12 checkB_input_pin_status main_window.Ui_mainWindow.checkB_input_pin_status

Definition at line 186 of file main_window.py.
Referenced by main.UiMainWindow1.check_file(), main.UiMainWindow1.disable_fields(), main.UiMainWindow1.enable_fields(), main_window.Ui_mainWindow.retranslateUi(), and main.UiMainWindow1.show_stat().

7.13.3.13 checkB_output_pin_status main_window.Ui_mainWindow.checkB_output_pin_status

Definition at line 191 of file main_window.py.
Referenced by main.UiMainWindow1.check_file(), main.UiMainWindow1.disable_fields(), main.UiMainWindow1.enable_fields(), main_window.Ui_mainWindow.retranslateUi(), and main.UiMainWindow1.show_stat().

7.13.3.14 checkB_wrong main_window.Ui_mainWindow.checkB_wrong

Definition at line 208 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), main.UiMainWindow1.check_wrong(), main.UiMainWindow1.disable_fields(), main.UiMainWindow1.enable_fields(), and main_window.Ui_mainWindow.retranslateUi().

7.13.3.15 dateTimeEdit_from main_window.Ui_mainWindow.dateTimeEdit_from

Definition at line 62 of file main_window.py.

7.13.3.16 dateTimeEdit_submitted main_window.Ui_mainWindow.dateTimeEdit_submitted

Definition at line 76 of file main_window.py.
Referenced by main.UiMainWindow1.show_stat().

7.13.3.17 dateTimeEdit_to main_window.Ui_mainWindow.dateTimeEdit_to

Definition at line 91 of file main_window.py.

7.13.3.18 filename_lineEdit main_window.Ui_mainWindow.filename_lineEdit

Definition at line 32 of file main_window.py.

7.13.3.19 horizontalLayout main_window.Ui_mainWindow.horizontalLayout

Definition at line 57 of file main_window.py.

7.13.3.20 horizontalLayout_10 main_window.Ui_mainWindow.horizontalLayout_10

Definition at line 198 of file main_window.py.

7.13.3.21 horizontalLayout_11 main_window.Ui_mainWindow.horizontalLayout_11

Definition at line 304 of file main_window.py.

7.13.3.22 horizontalLayout_12 main_window.Ui_mainWindow.horizontalLayout_12

Definition at line 24 of file main_window.py.

7.13.3.23 horizontalLayout_2 main_window.Ui_mainWindow.horizontalLayout_2

Definition at line 71 of file main_window.py.

7.13.3.24 horizontalLayout_3 main_window.Ui_mainWindow.horizontalLayout_3

Definition at line 86 of file main_window.py.

7.13.3.25 horizontalLayout_4 main_window.Ui_mainWindow.horizontalLayout_4

Definition at line 148 of file main_window.py.

7.13.3.26 horizontalLayout_5 main_window.Ui_mainWindow.horizontalLayout_5

Definition at line 162 of file main_window.py.

7.13.3.27 horizontalLayout_6 main_window.Ui_mainWindow.horizontalLayout_6

Definition at line 132 of file main_window.py.

7.13.3.28 horizontalLayout_7 main_window.Ui_mainWindow.horizontalLayout_7

Definition at line 52 of file main_window.py.

7.13.3.29 horizontalLayout_8 main_window.Ui_mainWindow.horizontalLayout_8

Definition at line 101 of file main_window.py.

7.13.3.30 horizontalLayout_9 main_window.Ui_mainWindow.horizontalLayout_9

Definition at line 113 of file main_window.py.

7.13.3.31 input_attempt main_window.Ui_mainWindow.input_attempt

Definition at line 115 of file main_window.py.

7.13.3.32 input_current_id main_window.Ui_mainWindow.input_current_id

Definition at line 103 of file main_window.py.
Referenced by main.UiMainWindow1.disable_fields(), and main.UiMainWindow1.show_stat().

7.13.3.33 input_file_location main_window.Ui_mainWindow.input_file_location

Definition at line 26 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), main.UiMainWindow1.my_open_file(), main.UiMainWindow1.open_file_diag(), main_window.Ui_mainWindow.retranslateUi(), and main.UiMainWindow1.setupUi().

7.13.3.34 input_final_grade main_window.Ui_mainWindow.input_final_grade

Definition at line 164 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), main.UiMainWindow1.check_file(), main.UiMainWindow1.check_wrong(), main.UiMainWindow1.disable_fields(), main.UiMainWindow1.enable_fields(), main.UiMainWindow1.reset_grade_resp(), main.UiMainWindow1.show_stat(), and main.UiMainWindow1.track_final_grade().

7.13.3.35 input_log_browser main_window.Ui_mainWindow.input_log_browser

Definition at line 298 of file main_window.py.
Referenced by main.UiMainWindow1.check_file(), main.UiMainWindow1.show_stat(), and main.UiMainWindow1.track_final_grade().

7.13.3.36 input_max_pos_grade main_window.Ui_mainWindow.input_max_pos_grade

Definition at line 134 of file main_window.py.

7.13.3.37 input_message_to_all main_window.Ui_mainWindow.input_message_to_all

Definition at line 285 of file main_window.py.

7.13.3.38 input_prev_response main_window.Ui_mainWindow.input_prev_response

Definition at line 275 of file main_window.py.
Referenced by main.UiMainWindow1.show_stat().

7.13.3.39 input_response_browser main_window.Ui_mainWindow.input_response_browser

Definition at line 259 of file main_window.py.
Referenced by main.UiMainWindow1.check_file(), main.UiMainWindow1.check_wrong(), main.UiMainWindow1.my_open_file(), main.UiMainWindow1.regrade(), main.UiMainWindow1.reset_grade_resp(), main_window.Ui_mainWindow.retranslateUi(), main.UiMainWindow1.save_response(), and main.UiMainWindow1.show_stat().

7.13.3.40 input_response_browser_user main_window.Ui_mainWindow.input_response_browser_user

Definition at line 265 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), main.UiMainWindow1.memorize_user_comment(), main_window.Ui_mainWindow.retranslateUi(), main.UiMainWindow1.save_response(), main.UiMainWindow1.show_stat(), and main.UiMainWindow1.update_user_comment_from_popular_answers().

7.13.3.41 input_subtract main_window.Ui_mainWindow.input_subtract

Definition at line 150 of file main_window.py.
Referenced by main.UiMainWindow1.check_file(), and main.UiMainWindow1.show_stat().

7.13.3.42 label_attempt main_window.Ui_mainWindow.label_attempt

Definition at line 123 of file main_window.py.
Referenced by main_window.Ui_mainWindow.retranslateUi().

7.13.3.43 label_current_id main_window.Ui_mainWindow.label_current_id

Definition at line 109 of file main_window.py.
Referenced by main_window.Ui_mainWindow.retranslateUi().

7.13.3.44 label_final main_window.Ui_mainWindow.label_final

Definition at line 172 of file main_window.py.
Referenced by main_window.Ui_mainWindow.retranslateUi().

7.13.3.45 label_from main_window.Ui_mainWindow.label_from

Definition at line 59 of file main_window.py.
Referenced by main_window.Ui_mainWindow.retranslateUi().

7.13.3.46 label_max_pos main_window.Ui_mainWindow.label_max_pos

Definition at line 142 of file main_window.py.
Referenced by main_window.Ui_mainWindow.retranslateUi().

7.13.3.47 label_submitted main_window.Ui_mainWindow.label_submitted

Definition at line 73 of file main_window.py.
Referenced by main_window.Ui_mainWindow.retranslateUi().

7.13.3.48 label_subtr main_window.Ui_mainWindow.label_subtr

Definition at line 157 of file main_window.py.
Referenced by main_window.Ui_mainWindow.retranslateUi().

7.13.3.49 label_to main_window.Ui_mainWindow.label_to

Definition at line 88 of file main_window.py.
Referenced by main_window.Ui_mainWindow.retranslateUi().

7.13.3.50 log_tab main_window.Ui_mainWindow.log_tab

Definition at line 294 of file main_window.py.
Referenced by main_window.Ui_mainWindow.retranslateUi().

7.13.3.51 manage_labs_but main_window.Ui_mainWindow.manage_labs_but

Definition at line 315 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), main.UiMainWindow1.open_manage_labs_diag(), main_window.Ui_mainWindow.retranslateUi(), and main.UiMainWindow1.setupUi().

7.13.3.52 popular_answers main_window.Ui_mainWindow.popular_answers

Definition at line 229 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), main.UiMainWindow1.disable_fields(), main.UiMainWindow1.enable_fields(), main.UiMainWindow1.memorize_user_comment(), main.UiMainWindow1.show_stat(), main.UiMainWindow1.update_popular_answers(), and main.UiMainWindow1.update_user_comment_from_popular_answers().

7.13.3.53 progressBar main_window.Ui_mainWindow.progressBar

Definition at line 336 of file main_window.py.
Referenced by main.UiMainWindow1.change_win_style(), main.UiMainWindow1.next_circ(), main.UiMainWindow1.prev_circ(), and main_window.Ui_mainWindow.retranslateUi().

7.13.3.54 response_tab main_window.Ui_mainWindow.response_tab

Definition at line 245 of file main_window.py.
Referenced by main_window.Ui_mainWindow.retranslateUi().

7.13.3.55 set_style_checkbox main_window.Ui_mainWindow.set_style_checkbox

Definition at line 319 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), main.UiMainWindow1.change_win_style(), and main_window.Ui_mainWindow.retranslateUi().

7.13.3.56 settings_but main_window.Ui_mainWindow.settings_but

Definition at line 322 of file main_window.py.
Referenced by main.UiMainWindow1.bind_functions(), main.UiMainWindow1.open_settings_dialog(), and main_window.Ui_mainWindow.retranslateUi().

7.13.3.57 splitter main_window.Ui_mainWindow.splitter

Definition at line 251 of file main_window.py.

7.13.3.58 tab_message_to_all main_window.Ui_mainWindow.tab_message_to_all

Definition at line 281 of file main_window.py.
Referenced by main_window.Ui_mainWindow.retranslateUi().

7.13.3.59 tab_prev_resp main_window.Ui_mainWindow.tab_prev_resp

Definition at line 271 of file main_window.py.
Referenced by main_window.Ui_mainWindow.retranslateUi().

7.13.3.60 tabs_for_log_and_resp main_window.Ui_mainWindow.tabs_for_log_and_resp

Definition at line 238 of file main_window.py.
Referenced by main_window.Ui_mainWindow.retranslateUi().

7.13.3.61 verticalLayout main_window.Ui_mainWindow.verticalLayout

Definition at line 55 of file main_window.py.

7.13.3.62 verticalLayout_2 main_window.Ui_mainWindow.verticalLayout_2

Definition at line 130 of file main_window.py.

7.13.3.63 verticalLayout_3 main_window.Ui_mainWindow.verticalLayout_3

Definition at line 99 of file main_window.py.

7.13.3.64 verticalLayout_4 main_window.Ui_mainWindow.verticalLayout_4

Definition at line 179 of file main_window.py.

7.13.3.65 verticalLayout_5 main_window.Ui_mainWindow.verticalLayout_5

Definition at line 273 of file main_window.py.

7.13.3.66 verticalLayout_6 main_window.Ui_mainWindow.verticalLayout_6

Definition at line 296 of file main_window.py.

7.13.3.67 verticalLayout_7 main_window.Ui_mainWindow.verticalLayout_7

Definition at line 22 of file main_window.py.

7.13.3.68 verticalLayout_8 main_window.Ui_mainWindow.verticalLayout_8

Definition at line 283 of file main_window.py.

7.13.3.69 verticalLayout_9 main_window.Ui_mainWindow.verticalLayout_9

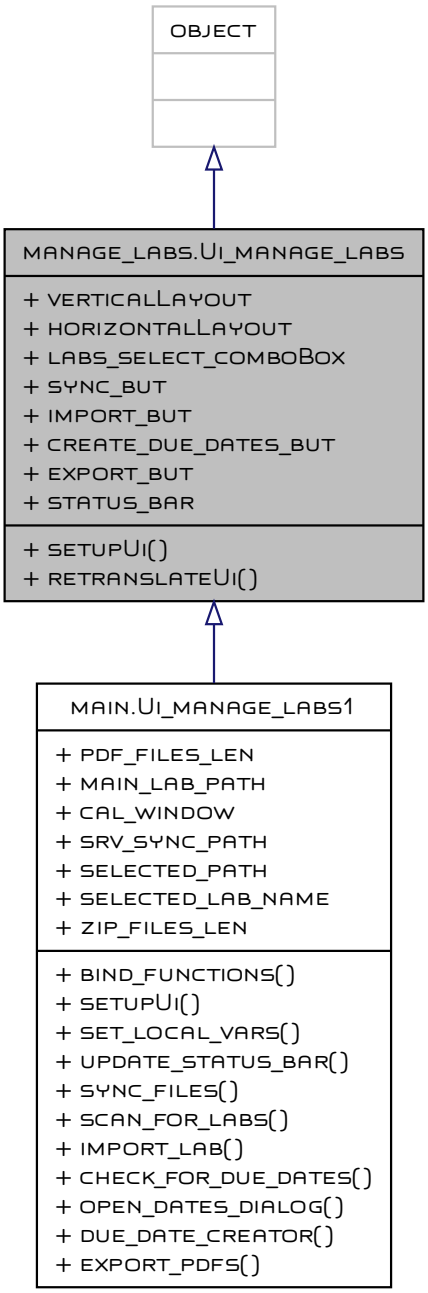
Definition at line 249 of file main_window.py.

The documentation for this class was generated from the following file:

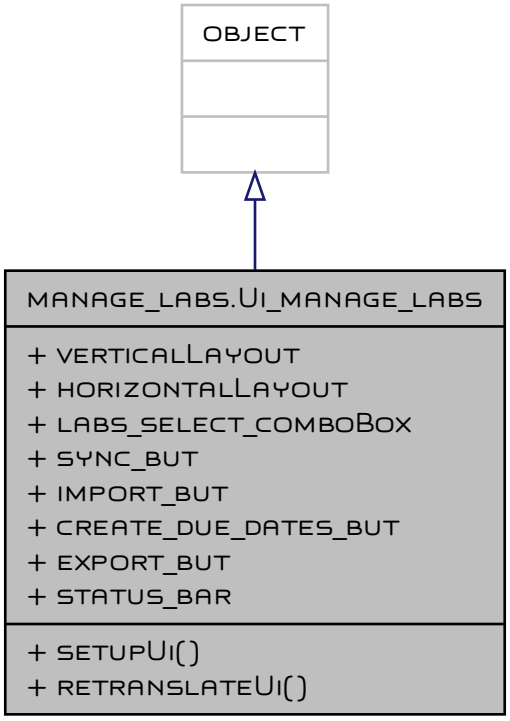
- main_window.py

7.14 manage_labs.Ui_manage_labs Class Reference

Inheritance diagram for manage_labs.Ui_manage_labs:



Collaboration diagram for manage_labs.Ui_manage_labs:



Public Member Functions

- def setupUi (self, manage_labs)
- def retranslateUi (self, manage_labs)

Public Attributes

- verticalLayout
- horizontalLayout
- labs_select_comboBox
- sync_but
- import_but
- create_due_dates_but
- export_but
- status_bar

7.14.1 Detailed Description

Definition at line 11 of file manage_labs.py.

7.14.2 Member Function Documentation

7.14.2.1 retranslateUi() def manage_labs.Ui_manage_labs.retranslateUi (self, manage_labs)

Definition at line 47 of file manage_labs.py.

```
00047     def retranslateUi(self, manage_labs):
00048         _translate = QtCore.QCoreApplication.translate
```

```
00049         manage_labs.setWindowTitle(_translate("manage_labs", "Manage labs"))
00050         self.sync_but.setText(_translate("manage_labs", "Sync to local storage"))
00051         self.import_but.setText(_translate("manage_labs", "import labs"))
00052         self.create_due_dates_but.setText(_translate("manage_labs", "Create due dates"))
00053         self.export_but.setText(_translate("manage_labs", "Export pdfs"))
00054
```

References manage_labs.Ui_manage_labs.create_due_dates_but, manage_labs.Ui_manage_labs.export_but, manage_labs.Ui_manage_labs.import_but, and manage_labs.Ui_manage_labs.sync_but.

7.14.2.2 setupUi()

def manage_labs.Ui_manage_labs.setupUi (self, manage_labs)

Reimplemented in main.Ui_manage_labs1.

Definition at line 12 of file manage_labs.py.

```
00012     def setupUi(self, manage_labs):
00013         manage_labs.setObjectName("manage_labs")
00014         manage_labs.resize(753, 90)
00015         manage_labs.setWindowFilePath("")
00016         self.verticalLayout = QtWidgets.QVBoxLayout(manage_labs)
00017         self.verticalLayout.setObjectName("verticalLayout")
00018         self.horizontalLayout = QtWidgets.QHBoxLayout()
00019         self.horizontalLayout.setObjectName("horizontalLayout")
00020         self.labs_select_comboBox = QtWidgets.QComboBox(manage_labs)
00021         self.labs_select_comboBox.setEnabled(False)
00022         self.labs_select_comboBox.setObjectName("labs_select_comboBox")
00023         self.horizontalLayout.addWidget(self.labs_select_comboBox)
00024         self.sync_but = QtWidgets.QPushButton(manage_labs)
00025         self.sync_but.setObjectName("sync_but")
00026         self.horizontalLayout.addWidget(self.sync_but)
00027         self.import_but = QtWidgets.QPushButton(manage_labs)
00028         self.import_but.setEnabled(False)
00029         self.import_but.setObjectName("import_but")
00030         self.horizontalLayout.addWidget(self.import_but)
00031         self.create_due_dates_but = QtWidgets.QPushButton(manage_labs)
00032         self.create_due_dates_but.setEnabled(False)
00033         self.create_due_dates_but.setObjectName("create_due_dates_but")
00034         self.horizontalLayout.addWidget(self.create_due_dates_but)
00035         self.export_but = QtWidgets.QPushButton(manage_labs)
00036         self.export_but.setEnabled(False)
00037         self.export_but.setObjectName("export_but")
00038         self.horizontalLayout.addWidget(self.export_but)
00039         self.verticalLayout.addLayout(self.horizontalLayout)
00040         self.status_bar = QtWidgets.QLineEdit(manage_labs)
00041         self.status_bar.setObjectName("status_bar")
00042         self.verticalLayout.addWidget(self.status_bar)
00043
00044         self.retranslateUi(manage_labs)
00045         QtCore.QMetaObject.connectSlotsByName(manage_labs)
00046
```

7.14.3 Member Data Documentation

7.14.3.1 create_due_dates_but

manage_labs.Ui_manage_labs.create_due_dates_but

Definition at line 31 of file manage_labs.py.

Referenced by main.Ui_manage_labs1.bind_functions(), main.Ui_manage_labs1.open_dates_dialog(), manage_labs.Ui_manage_labs.retranslateUi(), main.Ui_manage_labs1.setupUi(), and main.Ui_manage_labs1.sync_files().

7.14.3.2 export_but

manage_labs.Ui_manage_labs.export_but

Definition at line 35 of file manage_labs.py.

Referenced by main.Ui_manage_labs1.bind_functions(), main.Ui_manage_labs1.export_pdfs(), manage_labs.Ui_manage_labs.retranslateUi(), main.Ui_manage_labs1.setupUi(), and main.Ui_manage_labs1.sync_files().

7.14.3.3 horizontalLayout

manage_labs.Ui_manage_labs.horizontalLayout

Definition at line 18 of file manage_labs.py.

7.14.3.4 import_but

manage_labs.Ui_manage_labs.import_but

Definition at line 27 of file manage_labs.py.

Referenced by main.Ui_manage_labs1.bind_functions(), main.Ui_manage_labs1.import_lab(), manage_labs.Ui_manage_labs.retranslateUi(), main.Ui_manage_labs1.setupUi(), and main.Ui_manage_labs1.sync_files().

7.14.3.5 labs_select_comboBox

manage_labs.Ui_manage_labs.labs_select_comboBox

Definition at line 20 of file manage_labs.py.

Referenced by main.Ui_manage_labs1.bind_functions(), main.Ui_manage_labs1.setupUi(), and main.Ui_manage_labs1.sync_files().

7.14.3.6 status_bar manage_labs.Ui_manage_labs.status_bar

Definition at line 40 of file manage_labs.py.
Referenced by main.Ui_manage_labs1.import_lab(), and main.Ui_manage_labs1.sync_files().

7.14.3.7 sync_but manage_labs.Ui_manage_labs.sync_but

Definition at line 24 of file manage_labs.py.
Referenced by main.Ui_manage_labs1.bind_functions(), manage_labs.Ui_manage_labs.retranslateUi(), and main.Ui_manage_labs1.sync_files().

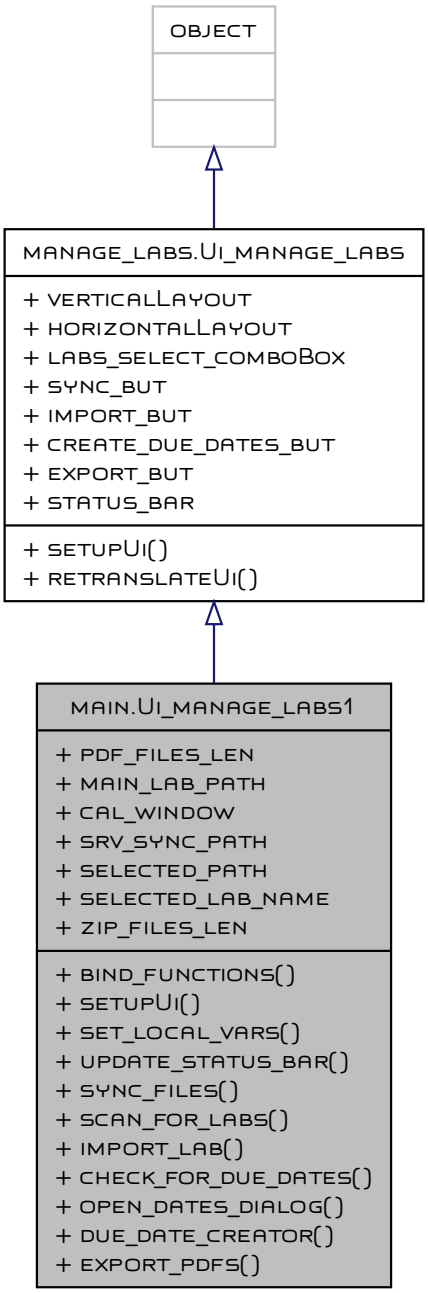
7.14.3.8 verticalLayout manage_labs.Ui_manage_labs.verticalLayout

Definition at line 16 of file manage_labs.py.
The documentation for this class was generated from the following file:

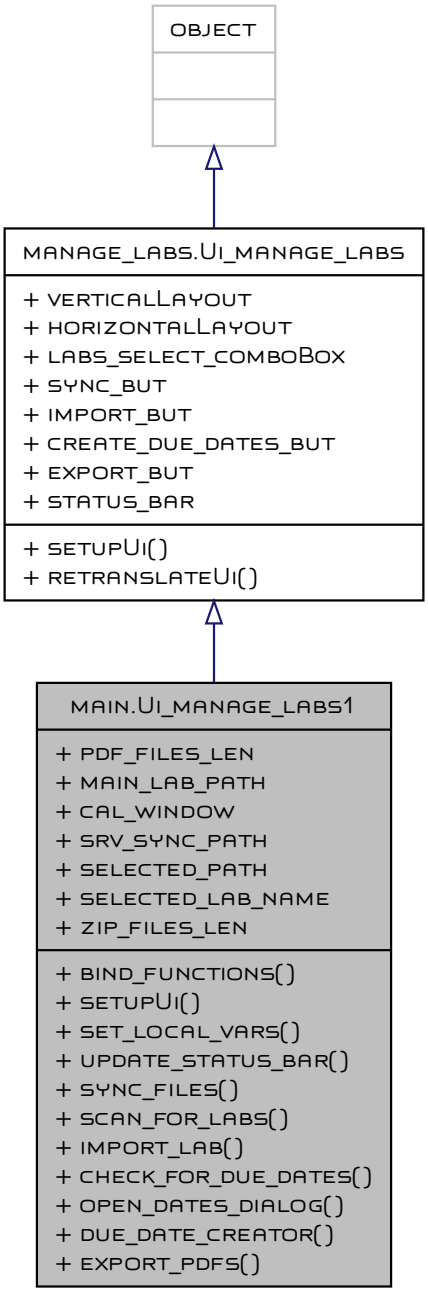
- manage_labs.py

7.15 main.Ui_manage_labs1 Class Reference

Inheritance diagram for main.Ui_manage_labs1:



Collaboration diagram for main.Ui_manage_labs1:



Public Member Functions

- def bind_functions (self)
- def setupUi (self, manage_labs)
- def set_local_vars (self)
- def update_status_bar (self, force=False)
- def sync_files (self)

- def scan_for_labs (self)
- def import_lab (self)
- def check_for_due_dates (self, dir)
- def open_dates_dialog (self)
- def due_date_creator (self, due_location, due_dates)
- def export_pdfs (self)

Public Attributes

- pdf_files_len
- main_lab_path
- cal_window

Static Public Attributes

- srv_sync_path = None
- selected_path = None
- selected_lab_name = None
- zip_files_len = None

7.15.1 Detailed Description

Definition at line 1574 of file main.py.

7.15.2 Member Function Documentation

7.15.2.1 bind_functions() def main.Ui_manage_labs1.bind_functions (self)

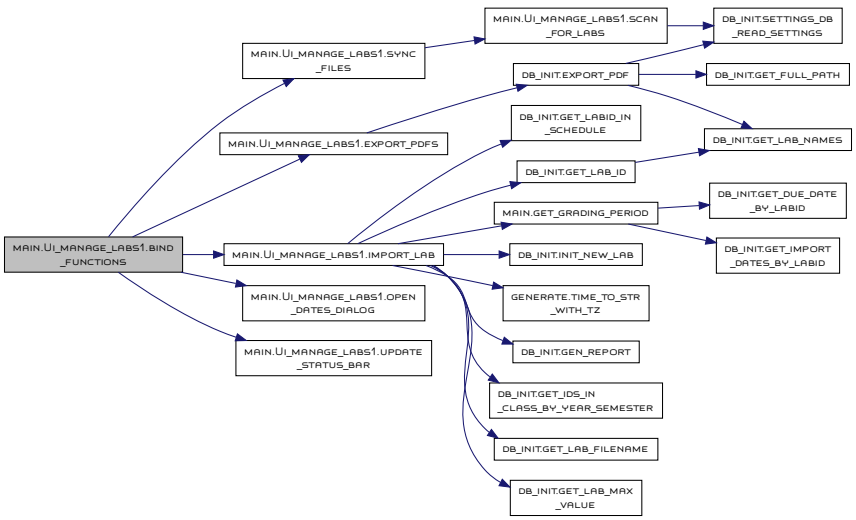
Definition at line 1580 of file main.py.

```
01580         self.import_but.clicked.connect(self.import_lab)
01581         self.create_due_dates_but.clicked.connect(self.open_dates_dialog)
01582         # self.sync_but.clicked.connect(lambda i: self.sync_but.setDisabled(True))
01583         self.sync_but.clicked.connect(self.sync_files)
01584         self.export_but.clicked.connect(self.export_pdfs)
01585 
01586     def setupUi(self, manage_labs):
01587         super().setupUi(manage_labs)
```

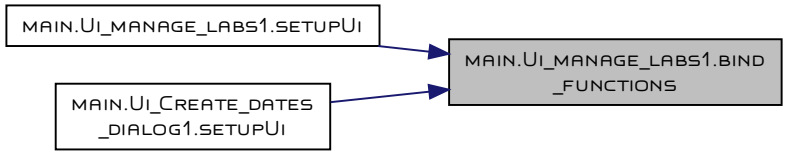
References manage_labs.Ui_manage_labs.create_due_dates_but, manage_labs.Ui_manage_labs.export_but, main.Ui_manage_labs1.export_pdfs(), manage_labs.Ui_manage_labs.import_but, main.Ui_manage_labs1.import_lab(), manage_labs.Ui_manage_labs.labs_select_comboBox, main.Ui_manage_labs1.open_dates_dialog(), manage_labs.Ui_manage_labs.sync_but, main.Ui_manage_labs1.sync_files(), and main.Ui_manage_labs1.update_status_bar().

Referenced by main.Ui_manage_labs1.setupUi(), and main.Ui_Create_dates_dialog1.setupUi().

Here is the call graph for this function:



Here is the caller graph for this function:



7.15.2.2 check_for_due_dates()

```
def main.Ui_manage_labs1.check_for_due_dates (
                self,
                dir )
```

Definition at line 1818 of file main.py.

```
01818
01819
01820     def open_dates_dialog(self):
01821         """
```

Definition at line 1856 of file main.py.

```
01856             i = 1
01857             for due_date in due_dates:
01858                 with open('%sdue_%d_%d' % (due_location, i, due_date), 'w'):
01859                     i += 1
01860         else:
01861             print('Location was not specified.')
01862
01863     def export_pdfs(self):
01864         self.export_but.setDisabled(True)
```

7.15.2.3 export_pdfs()

```
def main.Ui_manage_labs1.export_pdfs (
                self )
```

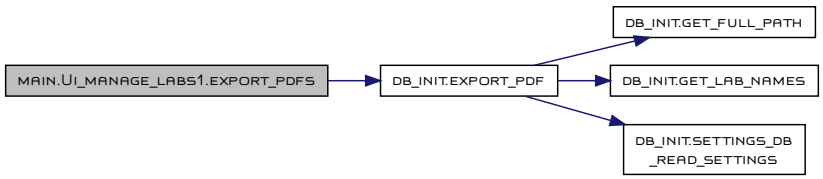
Definition at line 1865 of file main.py.

```
01865         self.export_but.setText('Exporting..')
01866         self.export_but.repaint()
01867         export_pdf()
01868         self.export_but.setText('Export pdfs')
01869         self.export_but.setEnabled(True)
01870
01871
01872 def get_grading_period(lid, cur_only=False):
01873     # should comput correct grading period and return the due date in Unix timestamp format
```

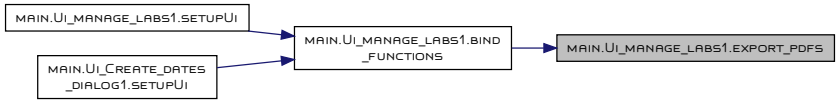
References manage_labs.Ui_manage_labs.export_but, and db_init.export_pdf().

Referenced by main.Ui_manage_labs1.bind_functions().

Here is the call graph for this function:



Here is the caller graph for this function:



7.15.2.4 import_lab() def main.Ui_manage_labs1.import_lab (self)

Definition at line 1664 of file main.py.

```
01664             self.import_but.setDisabled(True)
01665             self.import_but.setText('Importing..')
01666             self.import_but.repaint()
01667 
01668             # due_file = self.check_for_due_dates(self.selected_path)
01669             if False:
01670             # if len(due_file) < 4:
01671                 self.status_bar.setText('Create due dates !')
01672                 self.import_but.setText('Import labs')
01673                 self.import_but.setEnabled(True)
01674                 return False
01675             else:
01676                 from shutil import copy2 as cp2
01677                 zip_files = [f for f in os.listdir(self.selected_path) if 'zip' in f]
01678                 real_zip_files_rev = sorted([f for f in zip_files if os.path.isfile(os.path.join(self.selected_path, f))], reverse=True)
01679 
01680                 year, semester = self.main_lab_path.split('/')[-1].split('_')
01681                 ltype, _, lab_num = self.selected_lab_name.split('_')
01682                 lid = get_labid_in_schedule(get_lab_id(ltype, int(lab_num)), year, semester)
01683                 if lid is None:
01684                     self.status_bar.setText('Create due dates ! Lab is not initialised in lab_schedule')
01685                     self.import_but.setText('Import labs')
01686                     self.import_but.setEnabled(True)
01687                     return False
01688                 current_check, prev_due, next_due, current_timestamp = get_grading_period(lid)
01689 
01690                 if current_check > 4:
01691                     self.status_bar.setText('This lab has no more resubmissions (graded 4 times).')
01692                     self.import_but.setText('Import labs')
01693                     self.import_but.setEnabled(True)
01694                     return False
01695 
01696                 if current_timestamp < next_due:
01697                     # we cannot grade before the due date
01698                     self.status_bar.setText('Current date is less than next due date. It is too early to import.')
01699                     self.import_but.setText('Import labs')
01700                     self.import_but.setEnabled(True)
01701                     return False
01702 
01703 
01704 
01705                 penalty_mess = ''
01706                 if current_check == 1:
01707                     penalty_mess = '100% - this is your max point(no resubmissions)'
01708                 elif current_check == 2:
01709                     penalty_mess = '90% - first resubmission'
01710                 elif current_check == 3:
01711                     penalty_mess = '70% - second resubmission'
01712                 elif current_check == 4:
01713                     penalty_mess = '50% - third resubmission'
01714 
01715                 lab_type, _, lab_num = self.selected_lab_name.split('_')
01716                 lab_corr_name = lab_type[0] + 'LA' + lab_num
01717                 max_points = get_lab_max_value(lab_corr_name)
01718                 lab_filename = get_lab_filename(lab_corr_name)
01719 
01720                 # temporary solution. path should be stored as local var
01721                 paths_to_grading_dir = self.main_lab_path + '/' + self.selected_lab_name + '_' + str(current_check) + '/'
01722 
01723                 # proc_time = datetime.utcfromtimestamp(current_timestamp).strftime('%Y-%m-%d %H:%M:%S')
01724                 proc_time = time_to_str_with_tz(current_timestamp)
```

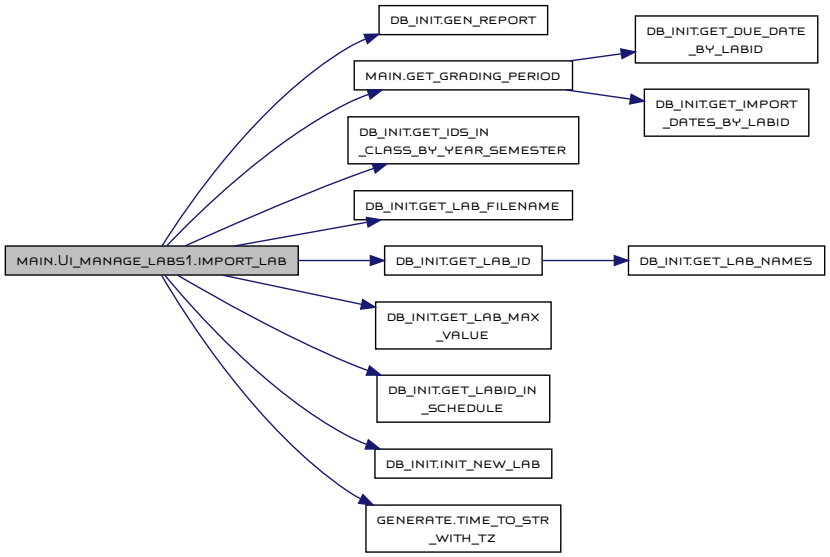
```
01725
01726                 # File manipulations goes below:
01727
01728                 if not os.path.isdir(paths_to_grading_dir):
01729                     os.makedirs(paths_to_grading_dir)
01730
01731                 cur_year, cur_sem = paths_to_grading_dir.split('/')[-3].split('_')
01732                 id_to_classId = get_ids_in_class_by_year_semester(cur_year, cur_sem)[0]
01733                 imported_files_counter = 0
01734
01735                 selected_files = []
01736                 for file in real_zip_files_rev:
01737                     parts = file.split('.')[0].split('-')
01738                     if int(parts[2]) > prev_due and int(parts[2]) <= next_due:
01739                         if len(selected_files) == 0:
01740                             selected_files.append(file)
01741                         elif selected_files[-1].split('.')[0].split('-')[0] != parts[0]:
01742                             selected_files.append(file)
01743
01744                 for file in reversed(selected_files):
01745                     zipped_file = zipfile.ZipFile(self.selected_path + file)
01746                     extraction_dir = paths_to_grading_dir + file.split('.')[0]
01747                     try:
01748                         zipped_file.extractall(paths_to_grading_dir + file.split('.')[0])
01749                     except Exception as e:
01750                         print(self.selected_path + file)
01751                         print(e)
01752                     finally:
01753                         zipped_file.close()
01754                     parts = file.split('.')[0].split('-')
01755                     subm_int = int(extraction_dir.split('-')[-1])
01756                     # subm_time = datetime.utcfromtimestamp(subm_int).replace(tzinfo=tz.tzutc()).astimezone(tz.tzlocal()).strftime('%Y-%m-%d
      %H:%M:%S')
01757                     subm_time = time_to_str_with_tz(subm_int)
01758                     # check for required files
01759                     if not lab_filename[0] or os.path.isfile(extraction_dir + '/' + lab_filename[0]):
01760                         lab_responce = 'I did not find any errors. Good job !'
01761                         cur_grade = max_points
01762                     else:
01763                         lab_responce = 'File "' + lab_filename[0] +'" was not found.\nThese files were found: ' +\
01764                                        " ".join(os.listdir(extraction_dir))
01765                         cur_grade = 0
01766
01767                     # This check is for a case when you graded the lab and trying to import it again.
01768                     # No existing files should be wiped
01769                     if not os.path.isfile(extraction_dir+'/penalty.txt'):
01770                         with open(extraction_dir+'/penalty.txt', 'w') as f:
01771                             f.write(penalty_mess)
01772
01773                     if not os.path.isfile(extraction_dir + '/grade.txt'):
01774                         with open(extraction_dir + '/grade.txt', 'w') as f:
01775                             f.write(str(cur_grade))
01776
01777                     if not os.path.isfile(extraction_dir + '/responce.txt'):
01778                         with open(extraction_dir + '/responce.txt', 'w') as f:
01779                             f.write(lab_responce)
01780
01781                     if not os.path.isfile(extraction_dir + '/tech_info.txt'):
01782                         with open(extraction_dir + '/tech_info.txt', 'w') as f:
01783                             f.writelines(['File was submited at %s<br/>\n' % subm_time,
01784                                           'I started processing your file at %s<br/>\n' % proc_time,
01785                                           "I found that your lab type is '%s' and it's number is %s <br/>" % (lab_type, lab_num),
01786                                           'So max points for this lab type is <u>%d</u><br/>' % max_points,
01787                                           'Theoretical max points: %s)' % penalty_mess])
01788
01789                     init_new_lab(id_to_classId[parts[0]], lid, current_check, subm_int, extraction_dir)
01790                     imported_files_counter += 1
01791
01792                 # cp2(self.selected_path + due_file[current_check-1], paths_to_grading_dir)
01793
01794                 # check_filename = paths_to_grading_dir + 'check_' + str(current_check) + '_' + str(current_timestamp)
01795                 # with open(check_filename, 'w'): pass
01796                 gen_report(lid, att=current_check)
01797
01798                 # cp2(check_filename, self.selected_path)
01799
01800                 self.import_but.setEnabled(True)
01801                 self.import_but.setText('Import labs')
01802                 self.status_bar.setText("Imported " + str(imported_files_counter) + " files.")
01803                 return True
01804
```

```
01805         return False
01806
01807
01808
01809     def check_for_due_dates(self, dir):
01810         """
```

References db_init.gen_report(), main.get_grading_period(), db_init.get_ids_in_class_by_year_semester(), db_init.get_lab_filename(), db_init.get_lab_id(), db_init.get_lab_max_value(), db_init.get_labid_in_schedule(), manage_labs.Ui_manage_labs.import_but, db_init.init_new_lab(), main.Ui_manage_labs1.main_lab_path, main.Ui_manage_labs1.selected_lab_name, main.Ui_manage_labs1.selected_path, manage_labs.Ui_manage_labs.status_bar, and generate.time_to_str_with_tz().

Referenced by main.Ui_manage_labs1.bind_functions().

Here is the call graph for this function:



Here is the caller graph for this function:

7.15.2.5 open_dates_dialog() def main.Ui_manage_labs1.open_dates_dialog (self)

Definition at line 1827 of file main.py.

```
01827         self.create_due_dates_but.repaint()
01828         self.cal_window = QtWidgets.QDialog()
01829         dui = Ui_Create_dates_dialog1()
01830         dui.setupUi(self.cal_window, self.selected_lab_name)
01831         # self.cal_window.finished.connect(self.check_new_win_result)
01832         self.cal_window.show()
01833         accepted = self.cal_window.exec_()
01834         if accepted:
01835             due_dates = list()
01836             due_dates.append(dui.init_subm_date_time.dateTime().toTime_t())
01837             due_dates.append(dui.first_subm_date_time.dateTime().toTime_t())
01838             due_dates.append(dui.second_subm_date_time.dateTime().toTime_t())
01839             due_dates.append(dui.third_subm_date_time.dateTime().toTime_t())
01840             due_location = dui.lab_path.text()
```

```
01841             self.due_date_creator(due_location, due_dates)
01842             year, semester = self.main_lab_path.split('/')[-1].split('_')
01843             ltype, _, lab_num = self.selected_lab_name.split('_')
01844             register_lab_in_semester(ltype, lab_num, year, semester, due_dates)
01845         self.create_due_dates_but.setEnabled(True)
01846
01847     # noinspection PyMethodMayBeStatic
01848     def due_date_creator(self, due_location, due_dates):
```

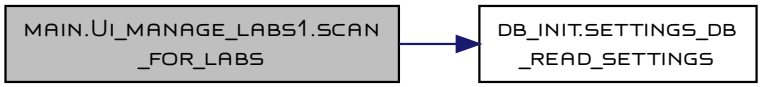
References manage_labs.Ui_manage_labs.create_due_dates_but.
Referenced by main.Ui_manage_labs1.bind_functions().
Here is the caller graph for this function:

7.15.2.6 scan_for_labs() def main.Ui_manage_labs1.scan_for_labs (self)

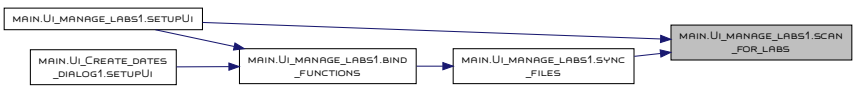
Definition at line 1651 of file main.py.

```
01651         # self.local_path = paths[1] + str(local[1]) + '_' + str(local[2]) + '/'
01652         self.main_lab_path = get_full_path(paths, local)
01653         self.srv_sync_path = self.main_lab_path + "/server_sync/"
01654         dirs = os.walk(self.srv_sync_path).__next__()[1]
01655         if len(dirs) > 0:
01656             self.labs_select_comboBox.addItems(sorted(dirs))
01657             self.labs_select_comboBox.setCurrentIndex(0)
01658             self.labs_select_comboBox.setFocus(True)
01659             self.update_status_bar(force=True)
01660
01661
01662     def import_lab(self):
01663         if self.selected_path:
```

References db_init.settings_db_read_settings().
Referenced by main.Ui_manage_labs1.setupUi(), and main.Ui_manage_labs1.sync_files().
Here is the call graph for this function:



Here is the caller graph for this function:



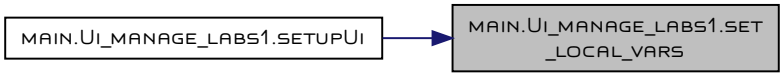
7.15.2.7 set_local_vars() def main.Ui_manage_labs1.set_local_vars (self)

Definition at line 1604 of file main.py.

```
01604
01605     def update_status_bar(self, force=False):
01606         # no need to scan files in background, but only when user selects it intentionally, or if it is first run
```

Referenced by main.Ui_manage_labs1.setupUi().

Here is the caller graph for this function:



7.15.2.8 setupUi()

```
def main.Ui_manage_labs1.setupUi (
        self,
        manage_labs )
```

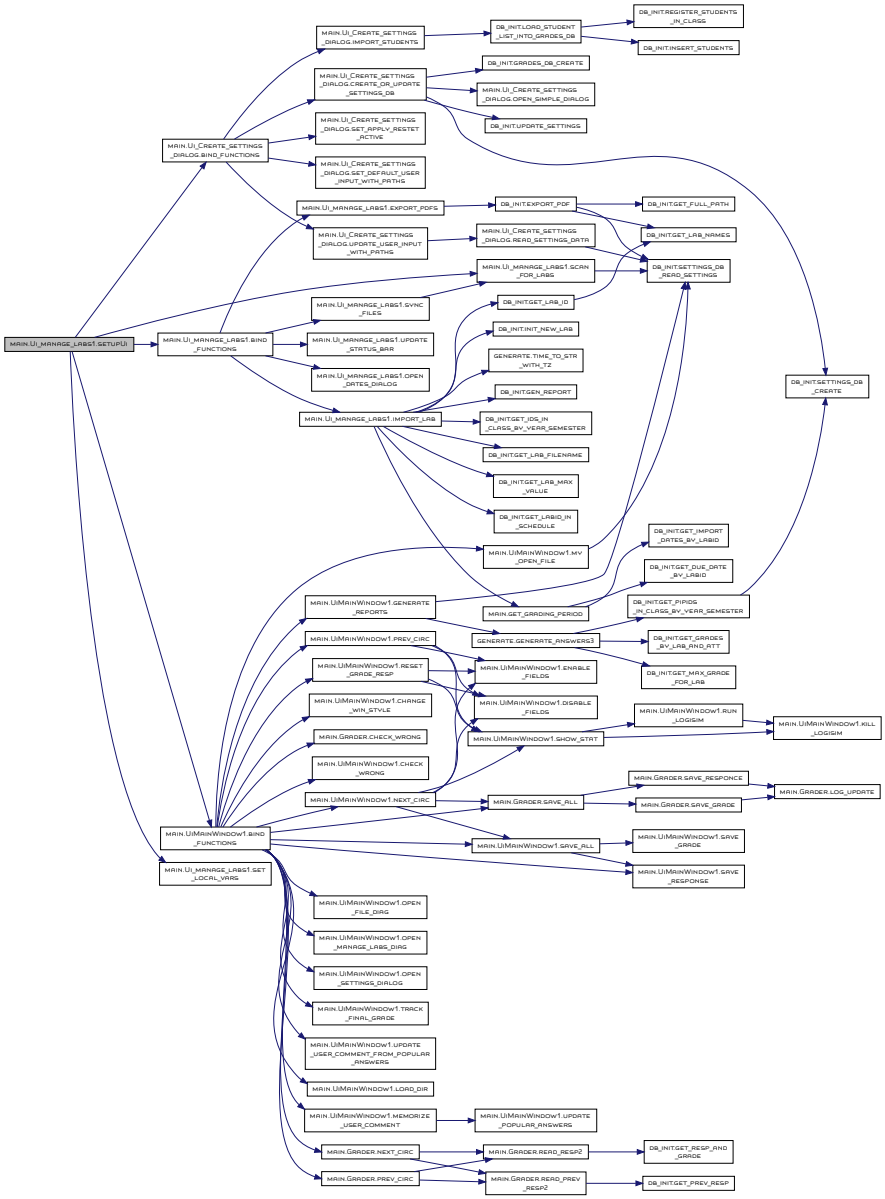
Reimplemented from manage_labs.Ui_manage_labs.

Definition at line 1588 of file main.py.

```
01588         self.bind_functions()
01589         self.set_local_vars()
01590
01591         try:
01592             self.scan_for_labs()
01593             if self.labs_select_comboBox.count() > 0:
01594                 self.labs_select_comboBox.setEnabled(True)
01595                 self.import_but.setEnabled(True)
01596                 self.create_due_dates_but.setEnabled(True)
01597                 self.export_but.setEnabled(True)
01598         except Exception as e:
01599             print('Error in manage labs. Probably your grading path was not set properly: ', e)
01600
01601
01602     def set_local_vars(self):
01603         pass
```

References main.UiMainWindow1.bind_functions(), main.Ui_Create_settings_dialog.bind_functions(), main.Ui_manage_labs1.bind_functions(), manage_labs.Ui_manage_labs.create_due_dates_but, manage_labs.Ui_manage_labs.export_but, manage_labs.Ui_manage_labs.import_but, manage_labs.Ui_manage_labs.labs_select_comboBox, main.Ui_manage_labs1.scan_for_labs(), and main.Ui_manage_labs1.set_local_vars().

Here is the call graph for this function:



7.15.2.9 sync_files() def main.Ui_manage_labs1.sync_files (self)

Definition at line 1626 of file main.py.

```
01626         self.sync_but.setText('Synchronizing...')
01627         self.sync_but.repaint()
01628         self.status_bar.setText("Synchronizing...")
01629         self.status_bar.repaint()
01630         sync_files()
01631         self.status_bar.setText("Done.")
01632         self.sync_but.setText('Sync to local storage')
01633         self.sync_but.setEnabled(True)
01634 
01635         sync_success = True  # there are no tools to check it at this point.
01636         if sync_success and not self.labs_select_comboBox.isEnabled():
01637             self.labs_select_comboBox.setEnabled(True)
```

```
01638             self.create_due_dates_but.setEnabled(True)
01639             self.scan_for_labs()
01640             #  TODO: There should be additional checks to enable import and export, but I do not have enough time to implement them.
01641             self.import_but.setEnabled(True)
01642             self.export_but.setEnabled(True)
01643 
01644     def scan_for_labs(self):
01645         """
```

References manage_labs.Ui_manage_labs.create_due_dates_but, manage_labs.Ui_manage_labs.export_but, manage_labs.Ui_manage_labs.import_but, manage_labs.Ui_manage_labs.labs_select_comboBox, main.Ui_manage_labs1.scan_for_labs(), manage_labs.Ui_manage_labs.status_bar, and manage_labs.Ui_manage_labs.sync_but.

Referenced by main.Ui_manage_labs1.bind_functions().

Here is the call graph for this function:

Here is the caller graph for this function:

7.15.2.10 update_status_bar() def main.Ui_manage_labs1.update_status_bar (self, force = False)

```
Definition at line 1607 of file main.py.
01607         if self.labs_select_comboBox.hasFocus() or force:
01608             self.selected_lab_name = self.labs_select_comboBox.currentText()
01609             self.selected_path = self.srv_sync_path + self.selected_lab_name + '/'
01610             zip_pdf_files = [f for f in os.listdir(self.selected_path) if '.zip' in f or '.pdf' in f]
01611 
01612             self.pdf_files_len = len([f for f in zip_pdf_files if f.split('.')[1] == 'pdf'])
01613             self.zip_files_len = len([f for f in zip_pdf_files if f.split('.')[1] == 'zip'])
01614 
01615             self.status_bar.setText("Contains " + str(self.zip_files_len) + ' zip files and ' + str(self.pdf_files_len) + ' pdf files.')
01616 
01617             if self.zip_files_len > 0 and not self.create_due_dates_but.isEnabled():
01618                 self.export_but.setEnabled(True)
01619                 self.import_but.setEnabled(True)
01620                 self.labs_select_comboBox.setEnabled(True)
01621 
01622             # good_zip_files_size = len([f for f in zip_files if os.isfile(os.path.join(selected_path, f))])
01623 
01624     def sync_files(self):
01625         self.sync_but.setDisabled(True)
```

Referenced by main.Ui_manage_labs1.bind_functions().

Here is the caller graph for this function:

7.15.3 Member Data Documentation

7.15.3.1 cal_window main.Ui_manage_labs1.cal_window

Definition at line 1830 of file main.py.

7.15.3.2 main_lab_path main.Ui_manage_labs1.main_lab_path

Definition at line 1654 of file main.py.
Referenced by main.Ui_manage_labs1.import_lab().

7.15.3.3 pdf_files_len main.Ui_manage_labs1.pdf_files_len

Definition at line 1614 of file main.py.

7.15.3.4 selected_lab_name main.Ui_manage_labs1.selected_lab_name = None [static]

Definition at line 1577 of file main.py.
Referenced by main.Ui_manage_labs1.import_lab().

7.15.3.5 selected_path main.Ui_manage_labs1.selected_path = None [static]

Definition at line 1576 of file main.py.
Referenced by main.Ui_manage_labs1.import_lab().

7.15.3.6 srv_sync_path main.Ui_manage_labs1.srv_sync_path = None [static]

Definition at line 1575 of file main.py.

7.15.3.7 zip_files_len main.Ui_manage_labs1.zip_files_len = None [static]

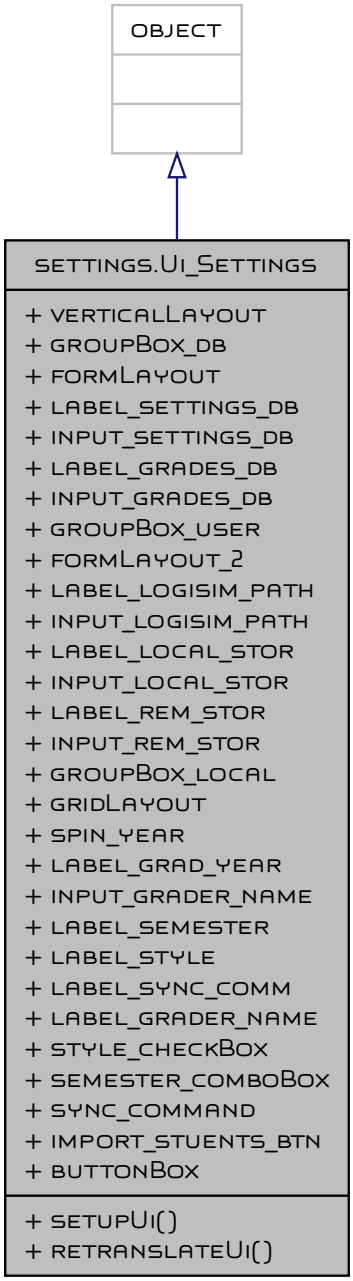
Definition at line 1578 of file main.py.
The documentation for this class was generated from the following file:

- main.py

7.16 settings.Ui_Settings Class Reference

Inheritance diagram for settings.Ui_Settings:

Collaboration diagram for settings.Ui_Settings:



Public Member Functions

- def setupUi (self, Settings)
- def retranslateUi (self, Settings)

Public Attributes

- verticalLayout
- groupBox_db
- formLayout
- label_settings_db
- input_settings_db
- label_grades_db
- input_grades_db
- groupBox_user
- formLayout_2
- label_logisim_path
- input_logisim_path
- label_local_stor
- input_local_stor
- label_rem_stor
- input_rem_stor
- groupBox_local
- gridLayout
- spin_year
- label_grad_year
- input_grader_name
- label_semester
- label_style
- label_sync_comm
- label_grader_name
- style_checkBox
- semester_comboBox
- sync_command
- import_stuents_btn
- buttonBox

7.16.1 Detailed Description

Definition at line 11 of file settings.py.

7.16.2 Member Function Documentation

7.16.2.1 retranslateUi() def settings.Ui_Settings.retranslateUi (self, Settings)

Definition at line 227 of file settings.py.

```
00227     def retranslateUi(self, Settings):
00228         _translate = QtCore.QCoreApplication.translate
00229         Settings.setWindowTitle(_translate("Settings", "Settings"))
00230         self.groupBox_db.setTitle(_translate("Settings", "&Database paths:"))
00231         self.label_settings_db.setText(_translate("Settings", "Settings"))
00232         self.input_settings_db.setText(_translate("Settings", "./settings.sqlite3"))
00233         self.label_grades_db.setText(_translate("Settings", "Grades"))
00234         self.input_grades_db.setPlaceholderText(_translate("Settings", " /Documents/3130_labs/grades.sqlite3"))
00235         self.groupBox_user.setTitle(_translate("Settings", "User paths"))
00236         self.label_logisim_path.setText(_translate("Settings", "Logisim path"))
00237         self.input_logisim_path.setPlaceholderText(_translate("Settings", "path to logisim executable logisim.jar"))
00238         self.label_local_stor.setText(_translate("Settings", "Local lab storage"))
00239         self.input_local_stor.setPlaceholderText(_translate("Settings", "local directory that contains labs, reports, and other working
      files"))
00240         self.label_rem_stor.setText(_translate("Settings", "Remote lab storage"))
00241         self.input_rem_stor.setPlaceholderText(_translate("Settings", "sshfs mounted dir that points to submission directory on the remote
      server"))
00242         self.groupBox_local.setTitle(_translate("Settings", "&Local settings"))
00243         self.label_grad_year.setText(_translate("Settings", "Grading year"))
00244         self.label_semester.setText(_translate("Settings", "Grading semester"))
00245         self.label_style.setText(_translate("Settings", "Use styles"))
00246         self.label_sync_comm.setText(_translate("Settings", "Sync command"))
00247         self.label_grader_name.setText(_translate("Settings", "Grader name"))
00248         self.semester_comboBox.setItemText(0, _translate("Settings", "Spring"))
00249         self.semester_comboBox.setItemText(1, _translate("Settings", "Summer"))
00250         self.semester_comboBox.setItemText(2, _translate("Settings", "Fall"))
00251         self.sync_command.setPlaceholderText(_translate("Settings", "rsync -avz ? cp -v ? dd ... ?"))
00252         self.import_stuents_btn.setText(_translate("Settings", "Import_students"))
00253
```

References settings.Ui_Settings.groupBox_db, settings.Ui_Settings.groupBox_local, settings.Ui_Settings.groupBox_user, settings.Ui_Settings.import_stuents_btn, settings.Ui_Settings.input_grades_db, settings.Ui_Settings.input_local_stor, settings.Ui_Settings.input_logisim_path, settings.Ui_Settings.input_rem_stor, settings.Ui_Settings.input_settings_db, settings.Ui_Settings.label_grad_year, settings.Ui_Settings.label_grader_name, settings.Ui_Settings.label_grades_db, settings.Ui_Settings.label_local_stor, settings.Ui_Settings.label_logisim_path, settings.Ui_Settings.label_rem_stor, settings.Ui_Settings.label_semester, settings.Ui_Settings.label_settings_db, settings.Ui_Settings.label_style, settings.Ui_Settings.label_sync_comm, settings.Ui_Settings.semester_comboBox, and settings.Ui_Settings.sync_command.

7.16.2.2 setupUi()

```
def settings.Ui_Settings.setupUi (
        self,
        Settings )
```

Reimplemented in main.Ui_Create_settings_dialog.

Definition at line 12 of file settings.py.

```
00012     def setupUi(self, Settings):
00013         Settings.setObjectName("Settings")
00014         Settings.setEnabled(True)
00015         Settings.resize(800, 487)
00016         Settings.setMinimumSize(QtCore.QSize(600, 0))
00017         icon = QtGui.QIcon()
00018         icon.addPixmap(QtGui.QPixmap("os_linux_1.ico"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
00019         Settings.setWindowIcon(icon)
00020         Settings.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00021         self.verticalLayout = QtWidgets.QVBoxLayout(Settings)
00022         self.verticalLayout.setObjectName("verticalLayout")
00023         self.groupBox_db = QtWidgets.QGroupBox(Settings)
00024         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)
00025         sizePolicy.setHorizontalStretch(0)
00026         sizePolicy.setVerticalStretch(0)
00027         sizePolicy.setHeightForWidth(self.groupBox_db.sizePolicy().hasHeightForWidth())
00028         self.groupBox_db.setSizePolicy(sizePolicy)
00029         self.groupBox_db.setMinimumSize(QtCore.QSize(0, 0))
00030         self.groupBox_db.setAutoFillBackground(False)
00031         self.groupBox_db.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignTop)
00032         self.groupBox_db.setFlat(False)
00033         self.groupBox_db.setCheckable(False)
00034         self.groupBox_db.setObjectName("groupBox_db")
00035         self.formLayout = QtWidgets.QFormLayout(self.groupBox_db)
00036         self.formLayout.setObjectName("formLayout")
00037         self.label_settings_db = QtWidgets.QLabel(self.groupBox_db)
00038         self.label_settings_db.setMinimumSize(QtCore.QSize(110, 0))
00039         self.label_settings_db.setObjectName("label_settings_db")
00040         self.formLayout.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.label_settings_db)
00041         self.input_settings_db = QtWidgets.QLineEdit(self.groupBox_db)
00042         self.input_settings_db.setEnabled(False)
00043         self.input_settings_db.setMinimumSize(QtCore.QSize(550, 31))
00044         self.input_settings_db.setObjectName("input_settings_db")
00045         self.formLayout.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.input_settings_db)
00046         self.label_grades_db = QtWidgets.QLabel(self.groupBox_db)
00047         self.label_grades_db.setMinimumSize(QtCore.QSize(110, 0))
00048         self.label_grades_db.setObjectName("label_grades_db")
00049         self.formLayout.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.label_grades_db)
00050         self.input_grades_db = QtWidgets.QLineEdit(self.groupBox_db)
00051         self.input_grades_db.setMinimumSize(QtCore.QSize(550, 31))
00052         self.input_grades_db.setText("")
00053         self.input_grades_db.setObjectName("input_grades_db")
00054         self.formLayout.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.input_grades_db)
00055         self.verticalLayout.addWidget(self.groupBox_db)
00056         self.groupBox_user = QtWidgets.QGroupBox(Settings)
00057         self.groupBox_user.setEnabled(False)
00058         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)
00059         sizePolicy.setHorizontalStretch(0)
00060         sizePolicy.setVerticalStretch(0)
00061         sizePolicy.setHeightForWidth(self.groupBox_user.sizePolicy().hasHeightForWidth())
00062         self.groupBox_user.setSizePolicy(sizePolicy)
00063         self.groupBox_user.setMinimumSize(QtCore.QSize(0, 0))
00064         self.groupBox_user.setObjectName("groupBox_user")
00065         self.formLayout_2 = QtWidgets.QFormLayout(self.groupBox_user)
00066         self.formLayout_2.setObjectName("formLayout_2")
00067         self.label_logisim_path = QtWidgets.QLabel(self.groupBox_user)
00068         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)
00069         sizePolicy.setHorizontalStretch(0)
00070         sizePolicy.setVerticalStretch(0)
00071         sizePolicy.setHeightForWidth(self.label_logisim_path.sizePolicy().hasHeightForWidth())
00072         self.label_logisim_path.setSizePolicy(sizePolicy)
00073         self.label_logisim_path.setMinimumSize(QtCore.QSize(110, 0))
00074         self.label_logisim_path.setObjectName("label_logisim_path")
00075         self.formLayout_2.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.label_logisim_path)
00076         self.input_logisim_path = QtWidgets.QLineEdit(self.groupBox_user)
00077         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Fixed)
00078         sizePolicy.setHorizontalStretch(0)
00079         sizePolicy.setVerticalStretch(0)
00080         sizePolicy.setHeightForWidth(self.input_logisim_path.sizePolicy().hasHeightForWidth())
00081         self.input_logisim_path.setSizePolicy(sizePolicy)
00082         self.input_logisim_path.setMinimumSize(QtCore.QSize(637, 31))
00083         self.input_logisim_path.setText("")
00084         self.input_logisim_path.setObjectName("input_logisim_path")
00085         self.formLayout_2.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.input_logisim_path)
00086         self.label_local_stor = QtWidgets.QLabel(self.groupBox_user)
00087         self.label_local_stor.setMinimumSize(QtCore.QSize(110, 0))
```

```
00088         self.label_local_stor.setObjectName("label_local_stor")
00089         self.formLayout_2.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.label_local_stor)
00090         self.input_local_stor = QtWidgets.QLineEdit(self.groupBox_user)
00091         self.input_local_stor.setMinimumSize(QtCore.QSize(637, 31))
00092         self.input_local_stor.setText("")
00093         self.input_local_stor.setObjectName("input_local_stor")
00094         self.formLayout_2.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.input_local_stor)
00095         self.label_rem_stor = QtWidgets.QLabel(self.groupBox_user)
00096         self.label_rem_stor.setMinimumSize(QtCore.QSize(110, 0))
00097         self.label_rem_stor.setObjectName("label_rem_stor")
00098         self.formLayout_2.setWidget(2, QtWidgets.QFormLayout.LabelRole, self.label_rem_stor)
00099         self.input_rem_stor = QtWidgets.QLineEdit(self.groupBox_user)
00100         self.input_rem_stor.setMinimumSize(QtCore.QSize(637, 31))
00101         self.input_rem_stor.setInputMask("")
00102         self.input_rem_stor.setText("")
00103         self.input_rem_stor.setObjectName("input_rem_stor")
00104         self.formLayout_2.setWidget(2, QtWidgets.QFormLayout.FieldRole, self.input_rem_stor)
00105         self.verticalLayout.addWidget(self.groupBox_user)
00106         self.groupBox_local = QtWidgets.QGroupBox(Settings)
00107         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.MinimumExpanding)
00108         sizePolicy.setHorizontalStretch(0)
00109         sizePolicy.setVerticalStretch(0)
00110         sizePolicy.setHeightForWidth(self.groupBox_local.sizePolicy().hasHeightForWidth())
00111         self.groupBox_local.setSizePolicy(sizePolicy)
00112         self.groupBox_local.setMinimumSize(QtCore.QSize(0, 145))
00113         self.groupBox_local.setMaximumSize(QtCore.QSize(16777215, 300))
00114         self.groupBox_local.setFlat(False)
00115         self.groupBox_local.setCheckable(False)
00116         self.groupBox_local.setObjectName("groupBox_local")
00117         self.gridLayout = QtWidgets.QGridLayout(self.groupBox_local)
00118         self.gridLayout.setObjectName("gridLayout")
00119         self.spin_year = QtWidgets.QSpinBox(self.groupBox_local)
00120         self.spin_year.setEnabled(False)
00121         self.spin_year.setMinimumSize(QtCore.QSize(110, 31))
00122         self.spin_year.setMaximumSize(QtCore.QSize(110, 16777215))
00123         self.spin_year.setWrapping(True)
00124         self.spin_year.setReadOnly(False)
00125         self.spin_year.setButtonSymbols(QtWidgets.QAbstractSpinBox.PlusMinus)
00126         self.spin_year.setAccelerated(True)
00127         self.spin_year.setProperty("showGroupSeparator", False)
00128         self.spin_year.setMinimum(2012)
00129         self.spin_year.setMaximum(2026)
00130         self.spin_year.setProperty("value", 2018)
00131         self.spin_year.setObjectName("spin_year")
00132         self.gridLayout.addWidget(self.spin_year, 0, 1, 1, 1)
00133         self.label_grad_year = QtWidgets.QLabel(self.groupBox_local)
00134         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Fixed)
00135         sizePolicy.setHorizontalStretch(0)
00136         sizePolicy.setVerticalStretch(0)
00137         sizePolicy.setHeightForWidth(self.label_grad_year.sizePolicy().hasHeightForWidth())
00138         self.label_grad_year.setSizePolicy(sizePolicy)
00139         self.label_grad_year.setMinimumSize(QtCore.QSize(110, 31))
00140         self.label_grad_year.setMaximumSize(QtCore.QSize(110, 16777215))
00141         self.label_grad_year.setObjectName("label_grad_year")
00142         self.gridLayout.addWidget(self.label_grad_year, 0, 0, 1, 1)
00143         self.input_grader_name = QtWidgets.QLineEdit(self.groupBox_local)
00144         self.input_grader_name.setEnabled(False)
00145         self.input_grader_name.setMinimumSize(QtCore.QSize(110, 31))
00146         self.input_grader_name.setMaximumSize(QtCore.QSize(110, 16777215))
00147         self.input_grader_name.setObjectName("input_grader_name")
00148         self.gridLayout.addWidget(self.input_grader_name, 2, 1, 1, 1)
00149         self.label_semester = QtWidgets.QLabel(self.groupBox_local)
00150         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Fixed)
00151         sizePolicy.setHorizontalStretch(0)
00152         sizePolicy.setVerticalStretch(0)
00153         sizePolicy.setHeightForWidth(self.label_semester.sizePolicy().hasHeightForWidth())
00154         self.label_semester.setSizePolicy(sizePolicy)
00155         self.label_semester.setMinimumSize(QtCore.QSize(110, 31))
00156         self.label_semester.setMaximumSize(QtCore.QSize(110, 16777215))
00157         self.label_semester.setObjectName("label_semester")
00158         self.gridLayout.addWidget(self.label_semester, 0, 3, 1, 1)
00159         self.label_style = QtWidgets.QLabel(self.groupBox_local)
00160         self.label_style.setMinimumSize(QtCore.QSize(110, 31))
00161         self.label_style.setObjectName("label_style")
00162         self.gridLayout.addWidget(self.label_style, 1, 0, 1, 1)
00163         self.label_sync_comm = QtWidgets.QLabel(self.groupBox_local)
00164         self.label_sync_comm.setObjectName("label_sync_comm")
00165         self.gridLayout.addWidget(self.label_sync_comm, 2, 3, 1, 1)
00166         self.label_grader_name = QtWidgets.QLabel(self.groupBox_local)
00167         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Fixed)
00168         sizePolicy.setHorizontalStretch(0)
```

```
00169         sizePolicy.setVerticalStretch(0)
00170         sizePolicy.setHeightForWidth(self.label_grader_name.sizePolicy().hasHeightForWidth())
00171         self.label_grader_name.setSizePolicy(sizePolicy)
00172         self.label_grader_name.setMinimumSize(QtCore.QSize(110, 31))
00173         self.label_grader_name.setObjectName("label_grader_name")
00174         self.gridLayout.addWidget(self.label_grader_name, 2, 0, 1, 1)
00175         self.style_checkBox = QtWidgets.QCheckBox(self.groupBox_local)
00176         self.style_checkBox.setEnabled(False)
00177         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.MinimumExpanding, QtWidgets.QSizePolicy.Fixed)
00178         sizePolicy.setHorizontalStretch(0)
00179         sizePolicy.setVerticalStretch(0)
00180         sizePolicy.setHeightForWidth(self.style_checkBox.sizePolicy().hasHeightForWidth())
00181         self.style_checkBox.setSizePolicy(sizePolicy)
00182         self.style_checkBox.setMinimumSize(QtCore.QSize(0, 31))
00183         self.style_checkBox.setMaximumSize(QtCore.QSize(110, 16777215))
00184         self.style_checkBox.setLayoutDirection(QtCore.Qt.LeftToRight)
00185         self.style_checkBox.setText("")
00186         self.style_checkBox.setObjectName("style_checkBox")
00187         self.gridLayout.addWidget(self.style_checkBox, 1, 1, 1, 1)
00188         self.semester_comboBox = QtWidgets.QComboBox(self.groupBox_local)
00189         self.semester_comboBox.setEnabled(False)
00190         self.semester_comboBox.setMinimumSize(QtCore.QSize(110, 31))
00191         self.semester_comboBox.setMaximumSize(QtCore.QSize(110, 16777215))
00192         self.semester_comboBox.setMaxVisibleItems(3)
00193         self.semester_comboBox.setMaxCount(5)
00194         self.semester_comboBox.setObjectName("semester_comboBox")
00195         self.semester_comboBox.addItem("")
00196         self.semester_comboBox.addItem("")
00197         self.semester_comboBox.addItem("")
00198         self.gridLayout.addWidget(self.semester_comboBox, 0, 4, 1, 1)
00199         self.sync_command = QtWidgets.QLineEdit(self.groupBox_local)
00200         self.sync_command.setEnabled(False)
00201         self.sync_command.setMinimumSize(QtCore.QSize(0, 31))
00202         self.sync_command.setInputMask("")
00203         self.sync_command.setObjectName("sync_command")
00204         self.gridLayout.addWidget(self.sync_command, 2, 4, 1, 4)
00205         self.import_stuents_btn = QtWidgets.QPushButton(self.groupBox_local)
00206         self.import_stuents_btn.setObjectName("import_stuents_btn")
00207         self.gridLayout.addWidget(self.import_stuents_btn, 0, 6, 1, 1)
00208         spacerItem = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
00209         self.gridLayout.addItem(spacerItem, 0, 5, 1, 1)
00210         self.verticalLayout.addWidget(self.groupBox_local)
00211         self.buttonBox = QtWidgets.QDialogButtonBox(Settings)
00212         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Fixed)
00213         sizePolicy.setHorizontalStretch(0)
00214         sizePolicy.setVerticalStretch(0)
00215         sizePolicy.setHeightForWidth(self.buttonBox.sizePolicy().hasHeightForWidth())
00216         self.buttonBox.setSizePolicy(sizePolicy)
00217         self.buttonBox.setOrientation(QtCore.Qt.Horizontal)
00218
      self.buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Apply|QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Ok|QtWidgets.QDialogButtonBox.Reset|Qt
00219         self.buttonBox.setObjectName("buttonBox")
00220         self.verticalLayout.addWidget(self.buttonBox)
00221
00222         self.retranslateUi(Settings)
00223         self.buttonBox.accepted.connect(Settings.accept)
00224         self.buttonBox.rejected.connect(Settings.reject)
00225         QtCore.QMetaObject.connectSlotsByName(Settings)
00226
```

7.16.3 Member Data Documentation

7.16.3.1 buttonBox `settings.Ui_Settings.buttonBox`

Definition at line 211 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.bind_functions(), main.Ui_Create_settings_dialog.create_or_update_settings_db(), main.Ui_Create_settings_dialog.read_settings_data(), main.Ui_Create_settings_dialog.set_apply_restet_active(), main.Ui_Create_settings_dialog.set_default_user_input_with_paths(), main.Ui_Create_settings_dialog.setupUi(), and main.Ui_Create_settings_dialog.update_user_input_with_paths().

7.16.3.2 formLayout `settings.Ui_Settings.formLayout`

Definition at line 35 of file settings.py.

7.16.3.3 formLayout_2 `settings.Ui_Settings.formLayout_2`

Definition at line 65 of file settings.py.

7.16.3.4 gridLayout settings.Ui_Settings.gridLayout

Definition at line 117 of file settings.py.

7.16.3.5 groupBox_db settings.Ui_Settings.groupBox_db

Definition at line 23 of file settings.py.
Referenced by settings.Ui_Settings.retranslateUi().

7.16.3.6 groupBox_local settings.Ui_Settings.groupBox_local

Definition at line 106 of file settings.py.
Referenced by settings.Ui_Settings.retranslateUi().

7.16.3.7 groupBox_user settings.Ui_Settings.groupBox_user

Definition at line 56 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.create_or_update_settings_db(), settings.Ui_Settings.retranslateUi(), main.Ui_Create_settings_dialog.set_default_user_input_with_paths(), and main.Ui_Create_settings_dialog.update_user_input_with_paths().

7.16.3.8 import_stuents_btn settings.Ui_Settings.import_stuents_btn

Definition at line 205 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.bind_functions(), main.Ui_Create_settings_dialog.import_students(), and settings.Ui_Settings.retranslateUi().

7.16.3.9 input_grader_name settings.Ui_Settings.input_grader_name

Definition at line 143 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.bind_functions(), main.Ui_Create_settings_dialog.create_or_update_settings_db(), and main.Ui_Create_settings_dialog.update_user_input_with_paths().

7.16.3.10 input_grades_db settings.Ui_Settings.input_grades_db

Definition at line 50 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.bind_functions(), main.Ui_Create_settings_dialog.create_or_update_settings_db(), main.Ui_Create_settings_dialog.import_students(), settings.Ui_Settings.retranslateUi(), main.Ui_Create_settings_dialog.set_default_user_input_with_paths(), and main.Ui_Create_settings_dialog.update_user_input_with_paths().

7.16.3.11 input_local_stor settings.Ui_Settings.input_local_stor

Definition at line 90 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.bind_functions(), main.Ui_Create_settings_dialog.create_or_update_settings_db(), settings.Ui_Settings.retranslateUi(), main.Ui_Create_settings_dialog.set_default_user_input_with_paths(), and main.Ui_Create_settings_dialog.update_user_input_with_paths().

7.16.3.12 input_logisim_path settings.Ui_Settings.input_logisim_path

Definition at line 76 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.bind_functions(), main.Ui_Create_settings_dialog.create_or_update_settings_db(), settings.Ui_Settings.retranslateUi(), main.Ui_Create_settings_dialog.set_default_user_input_with_paths(), and main.Ui_Create_settings_dialog.update_user_input_with_paths().

7.16.3.13 input_rem_stor settings.Ui_Settings.input_rem_stor

Definition at line 99 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.bind_functions(), main.Ui_Create_settings_dialog.create_or_update_settings_db(), settings.Ui_Settings.retranslateUi(), main.Ui_Create_settings_dialog.set_default_user_input_with_paths(), and main.Ui_Create_settings_dialog.update_user_input_with_paths().

7.16.3.14 input_settings_db settings.Ui_Settings.input_settings_db

Definition at line 41 of file settings.py.
Referenced by settings.Ui_Settings.retranslateUi().

7.16.3.15 label_grad_year settings.Ui_Settings.label_grad_year

Definition at line 133 of file settings.py.
Referenced by settings.Ui_Settings.retranslateUi().

7.16.3.16 label_grader_name settings.Ui_Settings.label_grader_name

Definition at line 166 of file settings.py.
Referenced by settings.Ui_Settings.retranslateUi().

7.16.3.17 label_grades_db settings.Ui_Settings.label_grades_db

Definition at line 46 of file settings.py.
Referenced by settings.Ui_Settings.retranslateUi().

7.16.3.18 label_local_stor settings.Ui_Settings.label_local_stor

Definition at line 86 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.create_or_update_settings_db(), and settings.Ui_Settings.retranslateUi().

7.16.3.19 label_logisim_path settings.Ui_Settings.label_logisim_path

Definition at line 67 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.create_or_update_settings_db(), and settings.Ui_Settings.retranslateUi().

7.16.3.20 label_rem_stor settings.Ui_Settings.label_rem_stor

Definition at line 95 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.create_or_update_settings_db(), and settings.Ui_Settings.retranslateUi().

7.16.3.21 label_semester settings.Ui_Settings.label_semester

Definition at line 149 of file settings.py.
Referenced by settings.Ui_Settings.retranslateUi().

7.16.3.22 label_settings_db settings.Ui_Settings.label_settings_db

Definition at line 37 of file settings.py.
Referenced by settings.Ui_Settings.retranslateUi().

7.16.3.23 label_style settings.Ui_Settings.label_style

Definition at line 159 of file settings.py.
Referenced by settings.Ui_Settings.retranslateUi().

7.16.3.24 label_sync_comm settings.Ui_Settings.label_sync_comm

Definition at line 163 of file settings.py.
Referenced by settings.Ui_Settings.retranslateUi().

7.16.3.25 semester_comboBox settings.Ui_Settings.semester_comboBox

Definition at line 188 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.bind_functions(), main.Ui_Create_settings_dialog.create_or_update_settings_db(), main.Ui_Create_settings_dialog.import_students(), settings.Ui_Settings.retranslateUi(), and main.Ui_Create_settings_dialog.update_user_input_with_paths().

7.16.3.26 spin_year settings.Ui_Settings.spin_year

Definition at line 119 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.bind_functions(), main.Ui_Create_settings_dialog.create_or_update_settings_db(), main.Ui_Create_settings_dialog.import_students(), and main.Ui_Create_settings_dialog.update_user_input_with_paths().

7.16.3.27 style_checkBox settings.Ui_Settings.style_checkBox

Definition at line 175 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.bind_functions(), main.Ui_Create_settings_dialog.create_or_update_settings_db(), and main.Ui_Create_settings_dialog.update_user_input_with_paths().

7.16.3.28 sync_command settings.Ui_Settings.sync_command

Definition at line 199 of file settings.py.
Referenced by main.Ui_Create_settings_dialog.bind_functions(), main.Ui_Create_settings_dialog.create_or_update_settings_db(), settings.Ui_Settings.retranslateUi(), and main.Ui_Create_settings_dialog.update_user_input_with_paths().

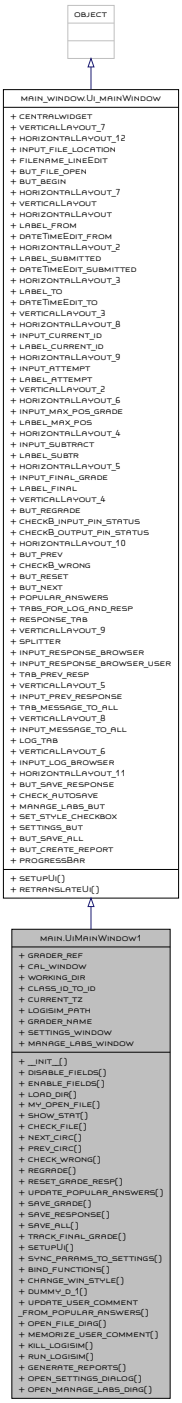
7.16.3.29 verticalLayout settings.Ui_Settings.verticalLayout

Definition at line 21 of file settings.py.
The documentation for this class was generated from the following file:

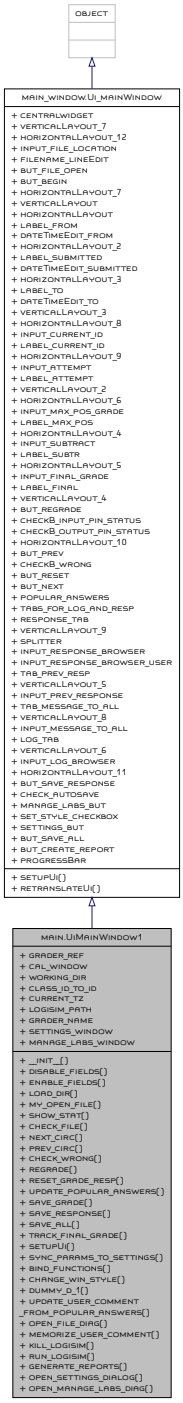
- settings.py

7.17 main.UiMainWindow1 Class Reference

Inheritance diagram for main.UiMainWindow1:



Collaboration diagram for main.UiMainWindow1:



Public Member Functions

- def __init__ (self)
- def disable_fields (self)
- def enable_fields (self)
- def load_dir (self)
- def my_open_file (self)

- def show_stat (self)
- def check_file (self)
- def next_circ (self)
- def prev_circ (self)
- def check_wrong (self)
- def regrade (self)
- def reset_grade_resp (self)
- def update_popular_answers (self)
- def save_grade (self)
- def save_response (self)
- def save_all (self)
- def track_final_grade (self)
- def setupUi (self, main_window)
- def sync_params_to_settings (self)
- def bind_functions (self)
- def change_win_style (self)
- def dummy_d_1 (self)
- def update_user_comment_from_popular_answers (self)
- def open_file_diag (self)
- def memorize_user_comment (self)
- def kill_logisim (self)
- def run_logisim (self, filename)
- def generate_reports (self)
- def open_settings_dialog (self)
- def open_manage_labs_diag (self)

Public Attributes

- grader_ref
- cal_window
- working_dir
- class_id_to_id
- current_tz
- logisim_path
- grader_name
- settings_window
- manage_labs_window

7.17.1 Detailed Description

Definition at line 719 of file main.py.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 __init__() def main.UiMainWindow1.__init__ (self)

Definition at line 721 of file main.py.

```
00721         self.grader_ref = None
00722         self.cal_window = None
00723         self.working_dir = None
00724 
00725 
00726 
00727     def disable_fields(self):
00728         """
```

7.17.3 Member Function Documentation

7.17.3.1 bind_functions() def main.UiMainWindow1.bind_functions (self)

Definition at line 1124 of file main.py.

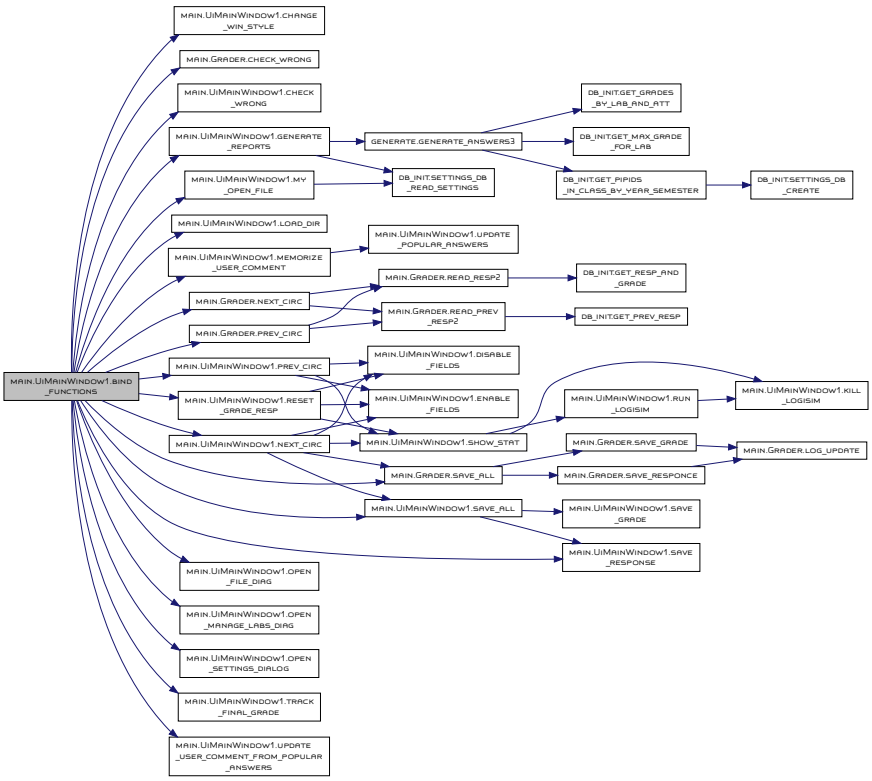
```
01124         self.but_begin.clicked.connect(self.load_dir)
01125         self.but_next.clicked.connect(self.next_circ)
01126         self.but_prev.clicked.connect(self.prev_circ)
01127         self.checkB_wrong.clicked.connect(self.check_wrong)
01128         #  self.but_regrade.clicked.connect(self.regrade)
01129         self.but_save_all.clicked.connect(self.save_all)
01130         self.but_save_response.clicked.connect(self.save_response)
01131         self.input_final_grade.textEdited.connect(self.track_final_grade)
01132         #  self.but_edit_done.clicked.connect(self.resp_edit_done)
01133         #  self.popular_answers.activated.connect(self.select_saved_answer)
01134         # self.but_create_report.setEnabled(True)  # Debug
```

```
01135         self.but_create_report.clicked.connect(self.generate_reports)
01136         # self.new_window_but.clicked.connect(self.open_dates_dialog)
01137         #  self.input_response_browser_user.focusInEvent(self, self.memorize_user_comment)
01138         #  self.custom_but_test.right_clicked[int].connect(self.dummy_d)
01139         self.input_file_location.dclicked.connect(self.open_file_diag)
01140         self.input_response_browser_user.focus_lost.connect(self.memorize_user_comment)
01141         self.popular_answers.currentIndexChanged.connect(self.update_user_comment_from_popular_answers)
01142         self.set_style_checkbox.stateChanged.connect(self.change_win_style)
01143         self.but_reset.clicked.connect(self.reset_grade_resp)
01144         self.settings_but.clicked.connect(self.open_settings_dialog)
01145         self.manage_labs_but.clicked.connect(self.open_manage_labs_diag)
01146         # self.sync_but.clicked.connect(self.sync_files)
01147 
01148 
01149 
01150     def change_win_style(self):
01151         """
```

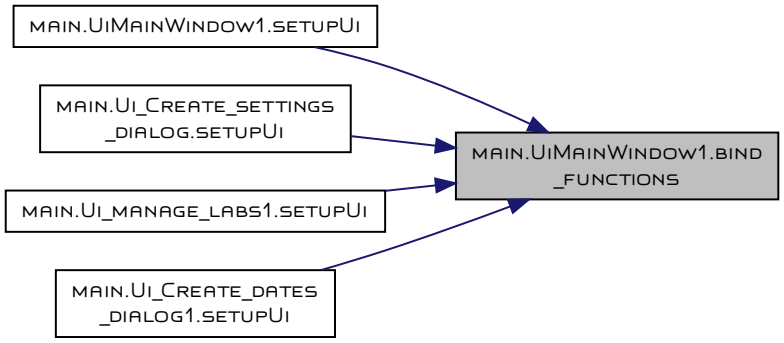
References main_window.Ui_mainWindow.but_begin, main_window.Ui_mainWindow.but_create_report, main_window.Ui_mainWindow.but_file_open, main_window.Ui_mainWindow.but_next, main_window.Ui_mainWindow.but_prev, main_window.Ui_mainWindow.but_reset, main_window.Ui_mainWindow.but_save_all, main_window.Ui_mainWindow.but_save_response, main.UiMainWindow1.change_win_style(), main.Grader.check_wrong(), main.UiMainWindow1.check_wrong(), main_window.Ui_mainWindow.checkB_wrong, main.UiMainWindow1.generate_reports(), main_window.Ui_mainWindow.input_file_location, main_window.Ui_mainWindow.input_final_grade, main_window.Ui_mainWindow.input_response_browser_user, main.UiMainWindow1.load_dir(), main_window.Ui_mainWindow.manage_labs_but, main.UiMainWindow1.memorize_user_comment(), main.UiMainWindow1.my_open_file(), main.Grader.next_circ(), main.UiMainWindow1.next_circ(), main.UiMainWindow1.open_file_diag(), main.UiMainWindow1.open_manage_labs_diag(), main.UiMainWindow1.open_settings_dialog(), main_window.Ui_mainWindow.popular_answers, main.Grader.prev_circ(), main.UiMainWindow1.prev_circ(), main.UiMainWindow1.reset_grade_resp(), main.Grader.save_all(), main.UiMainWindow1.save_all(), main.UiMainWindow1.save_response(), main_window.Ui_mainWindow.set_style_checkbox, main_window.Ui_mainWindow.settings_but, main.UiMainWindow1.track_final_grade(), and main.UiMainWindow1.update_user_comment_from_popular_answers().

Referenced by main.UiMainWindow1.setupUi(), main.Ui_Create_settings_dialog.setupUi(), main.Ui_manage_labs1.setupUi(), and main.Ui_Create_dates_dialog1.setupUi().

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.3.2 change_win_style() def main.UiMainWindow1.change_win_style (self)

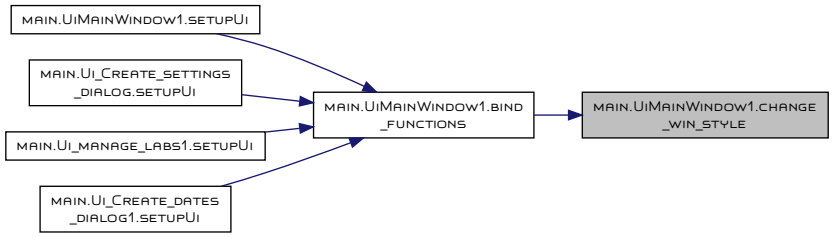
Definition at line 1157 of file main.py.

```
01157             self.progressBar.setStyleSheet(styleData)
01158         else:
01159             self.progressBar.setStyleSheet('')
01160 
01161     # noinspection PyMethodMayBeStatic
01162     def dummy_d_1(self):
```

References main_window.Ui_mainWindow.progressBar, and main_window.Ui_mainWindow.set_style_checkbox.

Referenced by main.UiMainWindow1.bind_functions().

Here is the caller graph for this function:



7.17.3.3 check_file() def main.UiMainWindow1.check_file (self)

Definition at line 900 of file main.py.

```
00900         self.input_final_grade.setText(str(self.grader_ref.final_grade))
00901 
00902         self.input_log_browser.setText(self.grader_ref.global_log)
00903         # self.input_log_browser.append(self.grader_ref.global_log)
00904 
00905         if self.grader_ref.input_correct:
00906             self.checkB_input_pin_status.setChecked(True)
00907         if self.grader_ref.output_correct:
00908             self.checkB_output_pin_status.setChecked(True)
00909 
00910         # self.but_save_response.setDisabled(True)
00911         # self.but_save_all.setDisabled(True)
00912 
```

```
00913         # self.but_edit_done.setDisabled(True)
00914         try:
00915             # self.grader_ref.generate_response()  #TODO this overwrites File not found.
00916             self.input_response_browser.setPlainText(self.grader_ref.resp_text)
00917             # self.but_edit_done.setEnabled(True)
00918             # self.but_save_response.setEnabled(True)
00919             # self.but_save_all.setEnabled(True)
00920         except Exception as e:
00921             print('Error in generate response:', e)
00922 
00923     def next_circ(self):
00924         """
```

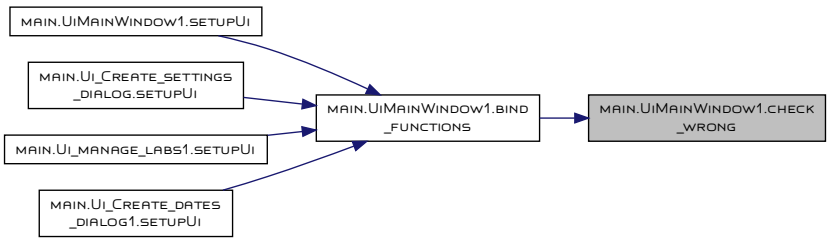
References main_window.Ui_mainWindow.checkB_input_pin_status, main_window.Ui_mainWindow.checkB_output_pin_status, main.UiMainWindow1.grader_ref, main_window.Ui_mainWindow.input_final_grade, main_window.Ui_mainWindow.input_log_browser, main_window.Ui_mainWindow.input_response_browser, and main_window.Ui_mainWindow.input_subtract.

7.17.3.4 check_wrong() def main.UiMainWindow1.check_wrong (self)

Definition at line 976 of file main.py.

```
00976             self.grader_ref.check_wrong()
00977             self.input_final_grade.setText(str(self.grader_ref.final_grade))
00978             self.grader_ref.log_update('Lab was marked as wrong manually. Zero was assigned to final grade.')
00979             self.input_response_browser.setPlainText(self.grader_ref.resp_text)
00980             self.checkB_wrong.setDisabled(True)
00981 
00982     def regrade(self):
00983         """
```

References main_window.Ui_mainWindow.checkB_wrong, main.UiMainWindow1.grader_ref, main_window.Ui_mainWindow.input_final_grade, and main_window.Ui_mainWindow.input_response_browser.
Referenced by main.UiMainWindow1.bind_functions().
Here is the caller graph for this function:



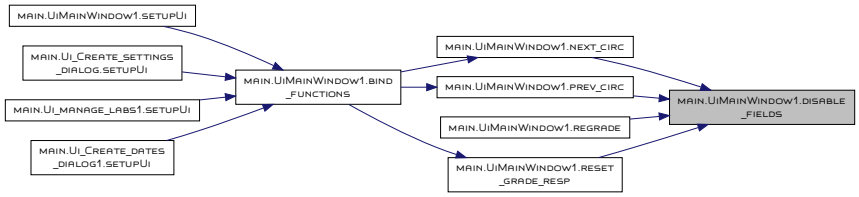
7.17.3.5 disable_fields() def main.UiMainWindow1.disable_fields (self)

Definition at line 733 of file main.py.

```
00733         self.checkB_output_pin_status.setDisabled(True)
00734         # self.input_response_browser.setDisabled(True)
00735         self.checkB_wrong.setDisabled(True)
00736 
00737         # self.input_subtract.setDisabled(True)
00738         self.but_regrade.setDisabled(True)
00739         self.popular_answers.setDisabled(True)
00740         self.input_final_grade.setDisabled(True)
00741         self.checkB_wrong.setChecked(False)
00742         self.check_autosave.setDisabled(True)
00743         self.input_current_id.setText('')
00744 
00745     def enable_fields(self):
00746         """
```

References main_window.Ui_mainWindow.but_regrade, main_window.Ui_mainWindow.check_autosave, main_window.Ui_mainWindow.checkB_input_pin_status, main_window.Ui_mainWindow.checkB_output_pin_status, main_window.Ui_mainWindow.checkB_wrong, main_window.Ui_mainWindow.input_current_id, main_window.Ui_mainWindow.input_final_grade, and main_window.Ui_mainWindow.popular_answers.
Referenced by main.UiMainWindow1.next_circ(), main.UiMainWindow1.prev_circ(), main.UiMainWindow1.regrade(), and main.UiMainWindow1.reset_grade_resp().

Here is the caller graph for this function:



7.17.3.6 dummy_d_1() def main.UiMainWindow1.dummy_d_1 (self)

Definition at line 1164 of file main.py.

```
01164
01165     def update_user_comment_from_popular_answers(self):
01166         """
```

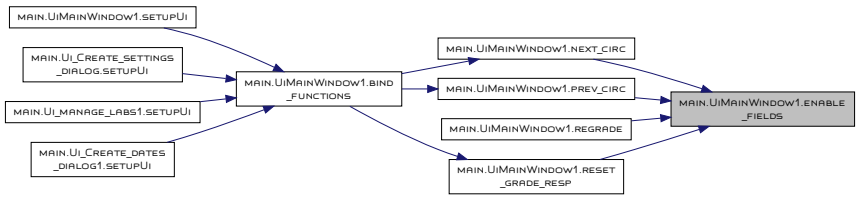
Definition at line 751 of file main.py.

```
00751         self.checkB_output_pin_status.setEnabled(True)
00752         # self.input_response_browser.setEnabled(True)
00753         self.checkB_wrong.setEnabled(True)
00754         self.input_final_grade.setEnabled(True)
00755         self.check_autosave.setEnabled(True)
00756
00757         # self.input_subtract.setEnabled(True)
00758         # self.but_regrade.setEnabled(True)
00759         self.popular_answers.setEnabled(True)
00760
00761     def load_dir(self):
00762         """
```

References main_window.Ui_mainWindow.check_autosave, main_window.Ui_mainWindow.checkB_input_pin_status, main_window.Ui_mainWindow.checkB_output_pin_status, main_window.Ui_mainWindow.checkB_wrong, main_window.Ui_mainWindow.input_final_grade, and main_window.Ui_mainWindow.popular_answers.

Referenced by main.UiMainWindow1.next_circ(), main.UiMainWindow1.prev_circ(), main.UiMainWindow1.regrade(), and main.UiMainWindow1.reset_grade_resp().

Here is the caller graph for this function:



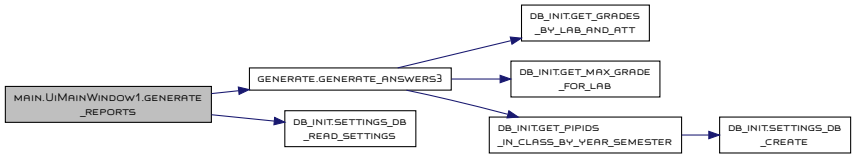
7.17.3.7 generate_reports() def main.UiMainWindow1.generate_reports (self)

Definition at line 1241 of file main.py.

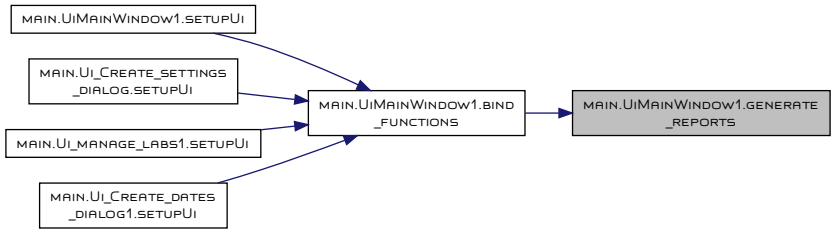
```
01241         self.but_create_report.setText('Generating..')
01242         self.but_create_report.repaint()
01243         # from generate import generate_answers
01244         # (resubmit_num, dir_name, lab_type, lab_num)
01245         if hasattr(self, 'grader_ref'):
01246             loc_settings = settings_db_read_settings()[1]
01247             generate_answers3(self.grader_ref.lid, self.grader_ref.attempt, self.grader_ref.year, self.grader_ref.semester)
01248             # generate_answers(self.grader_ref.attempt, self.grader_ref.working_dir, self.grader_ref.lab_type, self.grader_ref.lab_num,
      loc_settings[1], loc_settings[2], self.grader_name)
01249             # generate_answers2(self.grader_ref.attempt, self.grader_ref.working_dir, self.grader_ref.lab_type, self.grader_ref.lab_num,
      loc_settings[1], loc_settings[2], self.grader_name)
01250             self.but_create_report.setEnabled(True)
01251             self.but_create_report.setText('Create reports')
01252
```

```
01253
01254
01255     # def open_dates_dialog(self):
01256     #     """
```

References main_window.Ui_mainWindow.but_create_report, generate.generate_answers3(), main.UiMainWindow1.grader_ref, and db_init.settings_db_read_settings().
Referenced by main.UiMainWindow1.bind_functions().
Here is the call graph for this function:



Here is the caller graph for this function:

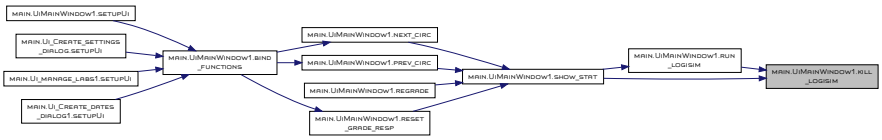


7.17.3.8 kill_logisim() def main.UiMainWindow1.kill_logisim (self)

Definition at line 1216 of file main.py.

```
01216             self.grader_ref.logisim_pid.kill()
01217         except Exception as e:
01218             print("was not able to kill : ", e)
01219
01220     def run_logisim(self, filename):
01221         """
```

References main.UiMainWindow1.grader_ref.
Referenced by main.UiMainWindow1.run_logisim(), and main.UiMainWindow1.show_stat().
Here is the caller graph for this function:



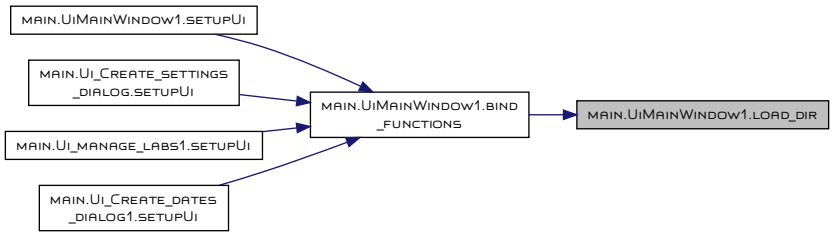
7.17.3.9 load_dir() def main.UiMainWindow1.load_dir (self)

Definition at line 767 of file main.py.

```
00767         cur_year, cur_sem = self.grader_ref.working_dir.split('/')[-3].split('_')
00768         self.class_id_to_id = get_ids_in_class_by_year_semester(cur_year, cur_sem)[1]
00769         self.but_begin.setDisabled(True)
00770         self.but_begin.repaint()
```

```
00771         self.progressBar.setEnabled(True)
00772
00773         self.disable_fields()
00774
00775         self.grader_ref.tot_elem = len(self.grader_ref.lab_paths)
00776         if self.grader_ref.tot_elem > 1:
00777             self.but_next.setEnabled(True)
00778
00779         self.progressBar.setMaximum(self.grader_ref.tot_elem)
00780         self.progressBar.setValue(0)
00781         self.popular_answers.clear()
00782
00783         # self.grader_ref.check_file(0)
00784         # self.grader_ref.stud_id = self.grader_ref.stud_ids[self.grader_ref.cur_idx]
00785         self.grader_ref.cur_idx = -1
00786         # graded = self.grader_ref.read_resp2()
00787         # if graded:
00788         #     self.grader_ref.read_prev_resp2()
00789         self.next_circ()
00790         # self.grader_ref.read_resp()
00791         # self.grader_ref.read_prev_resp()
00792         # self.show_stat()
00793         # self.check_file()
00794         # self.input_current_id.setPlainText(self.grader_ref.get_stud_id())
00795
00796         self.enable_fields()
00797         self.input_response_browser_user.setEnabled(True)
00798         self.but_regrade.setText('GRADE')
00799         self.but_save_all.setEnabled(True)
00800         self.but_save_response.setEnabled(True)
00801         self.check_autosave.setEnabled(True)
00802         self.but_reset.setEnabled(True)
00803
00804     def my_open_file(self):
00805         """
```

References main.UiMainWindow1.grader_ref.
Referenced by main.UiMainWindow1.bind_functions().
Here is the caller graph for this function:



7.17.3.10 memorize_user_comment() def main.UiMainWindow1.memorize_user_comment (self)

Definition at line 1191 of file main.py.

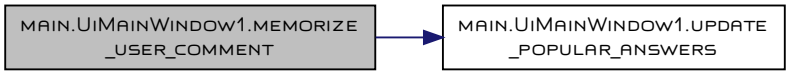
```
01191         if hasattr(self, 'grader_ref') and typed:
01192             try:
01193                 index = self.popular_answers.findText(self.input_response_browser_user.toPlainText(),
01194                                                       QtCore.Qt.MatchFixedString)
01195                 if index >= 0:
01196                     self.popular_answers.setCurrentIndex(index)
01197                 else:
01198                     self.grader_ref.add_to_common_answers(typed)
01199                     self.update_popular_answers()
01200                     index = self.popular_answers.findText(self.input_response_browser_user.toPlainText(),
01201                                                           QtCore.Qt.MatchFixedString)
01202                     try:
01203                         self.popular_answers.setCurrentIndex(index)
01204                     except Exception as e:
01205                         print('Failed to select proper index: ', e)
01206                         raise
01207             except Exception as e:
01208                 print('failed to add popular answer: ', e)
```

```
01209
01210     def kill_logisim(self):
01211         """
```

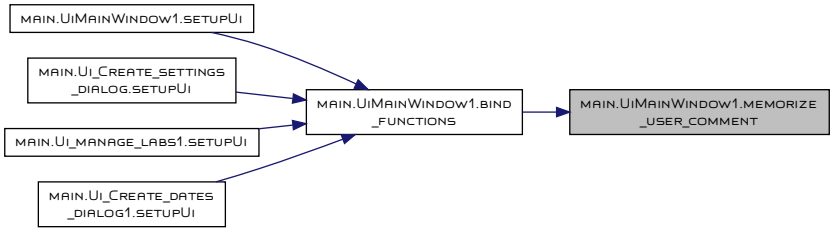
References main.UiMainWindow1.grader_ref, main_window.Ui_mainWindow.input_response_browser_user, main_window.Ui_mainWindow.popular_answers, and main.UiMainWindow1.update_popular_answers().

Referenced by main.UiMainWindow1.bind_functions().

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.3.11 my_open_file()

```
def main.UiMainWindow1.my_open_file (
        self )
```

Definition at line 812 of file main.py.

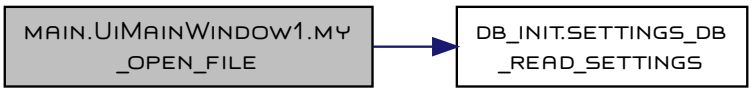
```
00812         # self.input_response_browser.clear()
00813         # self.input_response_browser_user.clear()
00814         self.input_response_browser.setPlainText('I did not find any errors. Good job!')
00815         grader_name = settings_db_read_settings()[1][0]
00816         self.current_tz = QDateTime.currentDateTime().timeZoneAbbreviation()
00817 
00818         try:
00819             my_grader = Grader(working_dir, grader_name)
00820             my_grader.open_dir()
00821 
00822             self.grader_ref = my_grader
00823 
00824             self.input_max_pos_grade.setText(str(my_grader.lab_max_grade))
00825             self.input_attempt.setText(str(my_grader.attempt))
00826             self.dateTimeEdit_from.setDateTime(my_grader.time_from_qt)
00827             self.dateTimeEdit_to.setDateTime(my_grader.time_to_qt)
00828             self.grader_ref.add_to_common_answers("")  # helps to remove all text in user comment section
00829             # QDateTime.currentDateTime().timeZone()
00830             # global MAIN_FILE_NAME, MAIN_FILE_NAME_OVERRIDE
00831 
00832             # MAIN_FILE_NAME = get_lab_filename(my_grader.lab_id)[0]
00833             # if not MAIN_FILE_NAME:
00834             #     # Old way, I was determining filename as the most common submitted file.
00835             #     if not MAIN_FILE_NAME_OVERRIDE:
00836             #         a = []
00837             #         for root, dirs, files in os.walk(working_dir):
00838             #             for file in files:
00839             #                 if file.endswith(".circ"):
00840             #                     a.append(file)
00841             #         a = np.array(a)
00842             #
00843             #         MAIN_FILE_NAME = Counter(a.flat).most_common(1)[0][0]
00844             #     else:
```

```
00845             #         MAIN_FILE_NAME = MAIN_FILE_NAME_OVERRIDE
00846             #     # Now I can just read it from DB
00847
00848             # self.grader_ref.circ_file_name = MAIN_FILE_NAME
00849             self.filename_lineEdit.setText(self.grader_ref.circ_file_name.split('.')[0])
00850             # self.reset_grade_resp()
00851             self.but_save_all.setChecked(False)
00852
00853             self.but_create_report.setEnabled(True)
00854             self.but_begin.setEnabled(True)
00855
00856         except Exception as e:  # TODO add log error
00857             print('Error in open_file : ', e)
00858             print(sys.exc_info()[0])
00859
00860     def show_stat(self):
00861         """
```

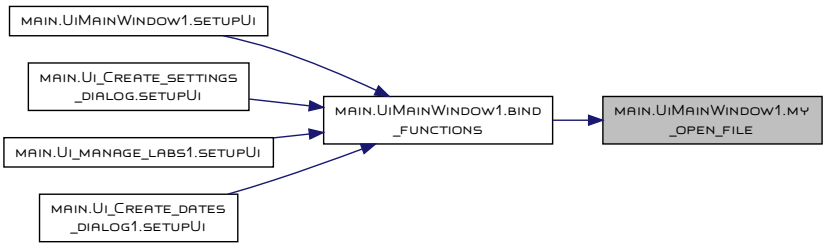
References main_window.Ui_mainWindow.input_file_location, main_window.Ui_mainWindow.input_response_browser, and db_init.settings_db_read_settings().

Referenced by main.UiMainWindow1.bind_functions().

Here is the call graph for this function:



Here is the caller graph for this function:

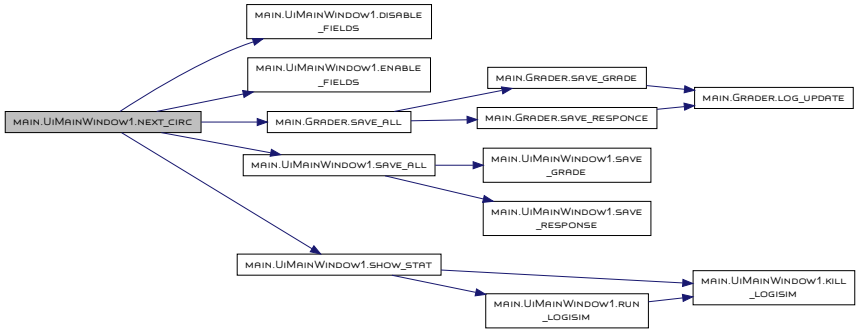


7.17.3.12 next_circ() def main.UiMainWindow1.next_circ (self)

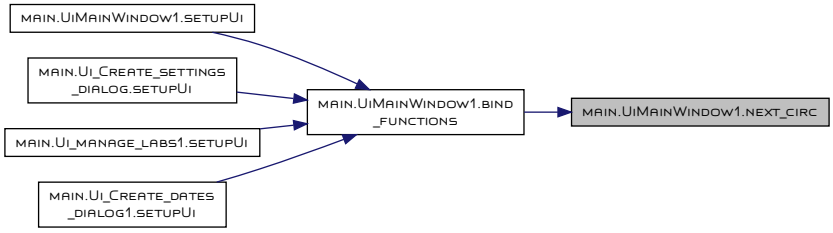
Definition at line 932 of file main.py.

```
00932         self.but_regrade.setText('GRADE')
00933         if self.check_autosave.isChecked() and self.grader_ref.cur_idx >= 0:
00934             self.save_all()
00935         # else:
00936         #     self.check_autosave.setDisabled(True)
00937         next_idx = self.grader_ref.next_circ()
00938         # self.check_file()
00939         self.show_stat()
00940         if next_idx >= self.grader_ref.tot_elem-1:
00941             self.but_next.setDisabled(True)
00942         if next_idx == 1:
00943             self.but_prev.setEnabled(True)
00944
00945         self.progressBar.setValue(next_idx)
00946         self.enable_fields()
00947
00948     def prev_circ(self):
```

```
00949         """
```
References main_window.Ui_mainWindow.but_next, main_window.Ui_mainWindow.but_prev, main_window.Ui_mainWindow.but_regrade, main_window.Ui_mainWindow.check_autosave, main.UiMainWindow1.disable_fields(), main.UiMainWindow1.enable_fields(), main.UiMainWindow1.grader_ref, main_window.Ui_mainWindow.progressBar, main.Grader.save_all(), main.UiMainWindow1.save_all(), and main.UiMainWindow1.show_stat().
Referenced by main.UiMainWindow1.bind_functions().
Here is the call graph for this function:



Here is the caller graph for this function:



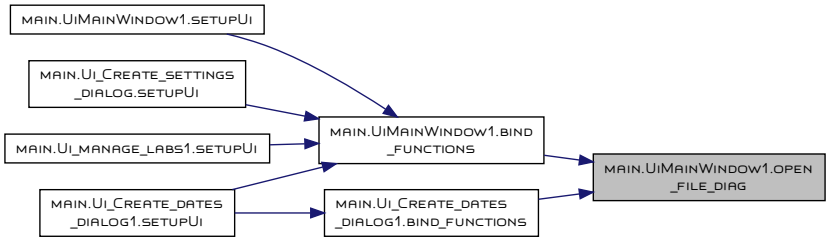
7.17.3.13 open_file_diag()

```
def main.UiMainWindow1.open_file_diag (
        self )
```

Definition at line 1180 of file main.py.

```
01180                                                 directory=self.input_file_location.text())
01181         if len(obtained_dir) > 1:
01182             self.input_file_location.setText(obtained_dir+'/')
01183 
01184     def memorize_user_comment(self):
01185         """
```
References main_window.Ui_mainWindow.input_file_location.
Referenced by main.UiMainWindow1.bind_functions(), and main.Ui_Create_dates_dialog1.bind_functions().

Here is the caller graph for this function:



7.17.3.14 open_manage_labs_diag() def main.UiMainWindow1.open_manage_labs_diag (self)

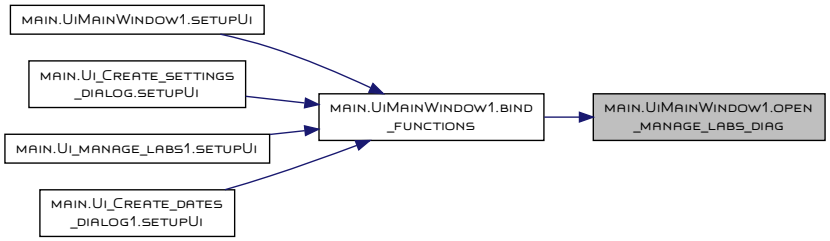
Definition at line 1315 of file main.py.

```
01315         self.manage_labs_but.repaint()
01316         self.centralwidget.setDisabled(True)
01317         self.centralwidget.repaint()
01318         self.manage_labs_window = QtWidgets.QDialog()
01319         dui = Ui_manage_labs1()
01320         dui.setupUi(self.manage_labs_window)
01321 
01322         self.manage_labs_window.show()
01323         self.manage_labs_window.exec_()
01324 
01325         self.centralwidget.setEnabled(True)
01326         self.manage_labs_but.setEnabled(True)
01327 
01328         if not self.but_file_open.isEnabled():
01329             paths, local = settings_db_read_settings()
01330             # if there are some labs in server sync directory:
01331             if len(os.walk(get_full_path(paths, local) + "/server_sync/").__next__()[1]) > 0:
01332                 self.but_file_open.setEnabled(True)
01333                 self.input_file_location.setEnabled(True)
01334 
01335 
01336 class Ui_Create_settings_dialog(Ui_Settings):
01337     """
```

References main_window.Ui_mainWindow.centralwidget, and main_window.Ui_mainWindow.manage_labs_but.

Referenced by main.UiMainWindow1.bind_functions().

Here is the caller graph for this function:



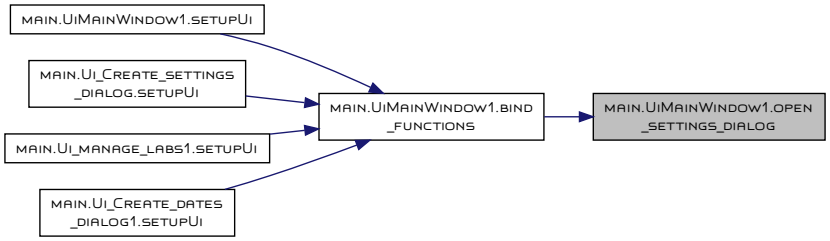
7.17.3.15 open_settings_dialog() def main.UiMainWindow1.open_settings_dialog (self)

Definition at line 1285 of file main.py.

```
01285         self.settings_but.repaint()
01286         self.settings_window = QtWidgets.QDialog()
01287         dui = Ui_Create_settings_dialog()
```

```
01288         dui.setupUi(self.settings_window)
01289
01290         self.centralwidget.setDisabled(True)
01291         self.centralwidget.repaint()
01292
01293         self.settings_window.show()
01294         self.settings_window.exec_()
01295
01296         self.sync_params_to_settings()
01297         self.centralwidget.setEnabled(True)
01298
01299         self.settings_but.setEnabled(True)
01300
01301         if not self.manage_labs_but.isEnabled():
01302             from pathlib import Path
01303             settings_location = str(Path(os.path.expandvars(os.path.expanduser('./settings.sqlite3'))).absolute())
01304             if os.path.isfile(settings_location):
01305                 self.manage_labs_but.setEnabled(True)
01306
01307
01308     def open_manage_labs_diag(self):
01309         """
```

References main_window.Ui_mainWindow.settings_but.
Referenced by main.UiMainWindow1.bind_functions().
Here is the caller graph for this function:



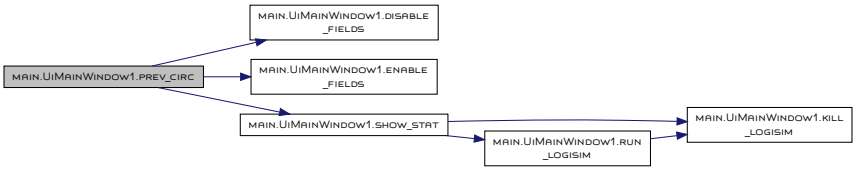
7.17.3.16 prev_circ() def main.UiMainWindow1.prev_circ (self)

Definition at line 957 of file main.py.

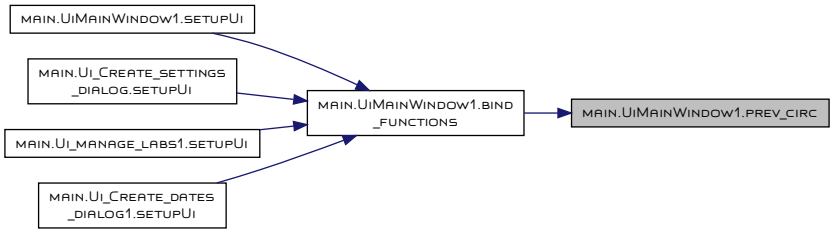
```
00957         self.but_regrade.setText('GRADE')
00958         next_idx = self.grader_ref.prev_circ()
00959         # self.check_file()
00960         self.show_stat()
00961         if next_idx <= self.grader_ref.tot_elem-1:
00962             self.but_next.setEnabled(True)
00963         if next_idx == 0:
00964             self.but_prev.setDisabled(True)
00965
00966         self.progressBar.setValue(next_idx)
00967         self.enable_fields()
00968
00969     def check_wrong(self):
00970         """
```

References main_window.Ui_mainWindow.but_next, main_window.Ui_mainWindow.but_prev, main_window.Ui_mainWindow.but_regrade, main.UiMainWindow1.disable_fields(), main.UiMainWindow1.enable_fields(), main.UiMainWindow1.grader_ref, main_window.Ui_mainWindow.progressBar, and main.UiMainWindow1.show_stat().
Referenced by main.UiMainWindow1.bind_functions().

Here is the call graph for this function:



Here is the caller graph for this function:



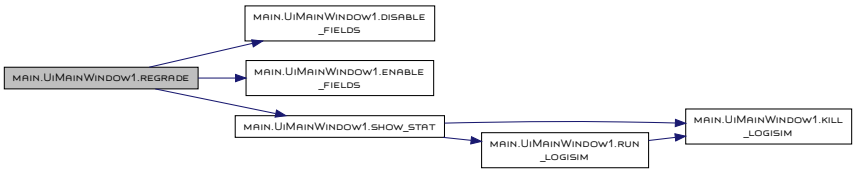
7.17.3.17 regrade() def main.UiMainWindow1.regrade (self)

Definition at line 988 of file main.py.

```
00988         self.but_regrade.setText('regrade')
00989         # if self.lab_num > 8 and self.lab_type == 'Closed':
00990         #    self.precheck_PLDs(i, cur_path)
00991         self.show_stat()
00992         # self.grader_ref.check_file()
00993         # if self.grader_ref.check_circ_exist():
00994         #     self.check_file()
00995         self.input_response_browser.setPlainText(self.grader_ref.resp_text)
00996         self.enable_fields()
00997 
00998     def reset_grade_resp(self):
00999         """
```

References main_window.Ui_mainWindow.but_regrade, main.UiMainWindow1.disable_fields(), main.UiMainWindow1.enable_fields(), main.UiMainWindow1.grader_ref, main_window.Ui_mainWindow.input_response_browser, and main.UiMainWindow1.show_stat().

Here is the call graph for this function:



7.17.3.18 reset_grade_resp() def main.UiMainWindow1.reset_grade_resp (self)

Definition at line 1004 of file main.py.

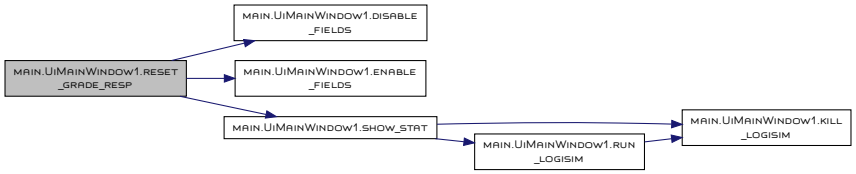
```
01004         self.show_stat()
01005         # self.grader_ref.check_file()
```

```
01006         # if self.grader_ref.check_circ_exist():
01007         if self.grader_ref.lab_num > 8 and self.grader_ref.lab_type == 'Closed':
01008            self.grader_ref.final_grade, report = self.grader_ref.precheck_PLDs(self.grader_ref.cur_idx)
01009            self.input_response_browser.setPlainText(report)
01010         else:
01011             self.grader_ref.final_grade = self.grader_ref.lab_max_grade
01012             self.input_response_browser.setPlainText('I did not find any errors. Good job!')
01013 
01014         self.input_final_grade.setText(str(self.grader_ref.final_grade))
01015         self.enable_fields()
01016 
01017     def update_popular_answers(self):
01018         """
```

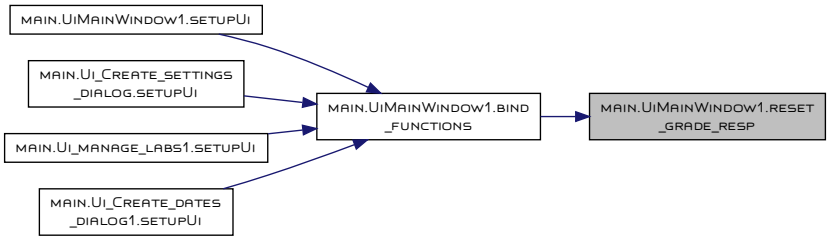
References main.UiMainWindow1.disable_fields(), main.UiMainWindow1.enable_fields(), main.UiMainWindow1.grader_ref, main_window.Ui_mainWindow.input_final_grade, main_window.Ui_mainWindow.input_response_browser, and main.UiMainWindow1.show_stat().

Referenced by main.UiMainWindow1.bind_functions().

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.3.19 run_logisim()

```
def main.UiMainWindow1.run_logisim (
        self,
        filename )
```

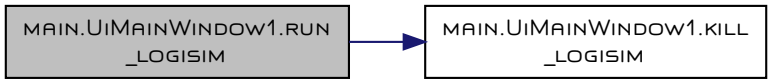
Definition at line 1228 of file main.py.

```
01228         command = 'java -jar ' + self.logisim_path + 'logisim-generic-2.7.1.jar {}'.format(filename)
01229         # command_with_file = command + os.path.join(self.grader_ref.file_list[self.grader_ref.cur_idx], MAIN_FILE_NAME)
01230         #  if self.grader_ref.logisim_pid.pid > 0:
01231         self.kill_logisim()
01232         self.grader_ref.logisim_pid = subprocess.Popen(command, shell=True)
01233 
01234     def generate_reports(self):
01235         """
```

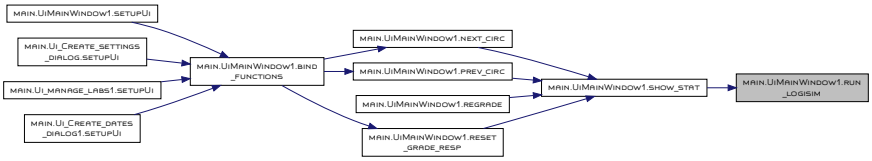
References main.UiMainWindow1.grader_ref, main.UiMainWindow1.kill_logisim(), and main.UiMainWindow1.logisim_path.

Referenced by main.UiMainWindow1.show_stat().

Here is the call graph for this function:



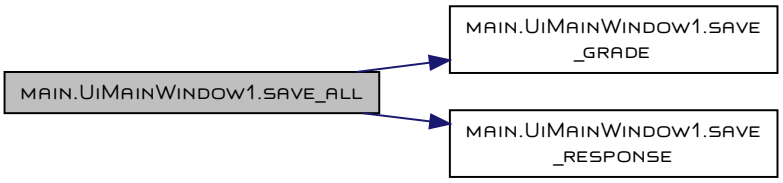
Here is the caller graph for this function:



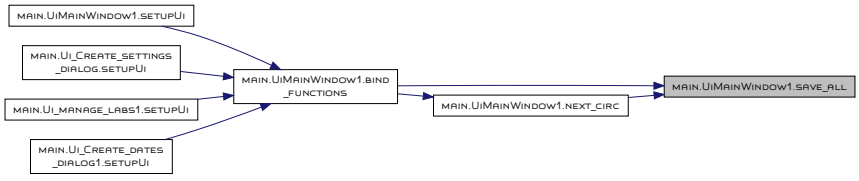
7.17.3.20 save_all() def main.UiMainWindow1.save_all (self)

```
Definition at line 1053 of file main.py.
01053         # self.grader_ref.save_responce()
01054         self.save_response()
01055         self.grader_ref.save_all2()
01056 
01057     def track_final_grade(self):
01058         """
```

References main.UiMainWindow1.grader_ref, main.UiMainWindow1.save_grade(), and main.UiMainWindow1.save_response().
Referenced by main.UiMainWindow1.bind_functions(), and main.UiMainWindow1.next_circ().
Here is the call graph for this function:



Here is the caller graph for this function:



7.17.3.21 save_grade() def main.UiMainWindow1.save_grade (self)

Definition at line 1035 of file main.py.

```
01035
01036     def save_response(self):
01037         """
```

References main.UiMainWindow1.grader_ref.

Referenced by main.UiMainWindow1.save_all().

Here is the caller graph for this function:

7.17.3.22 save_response() def main.UiMainWindow1.save_response (self)

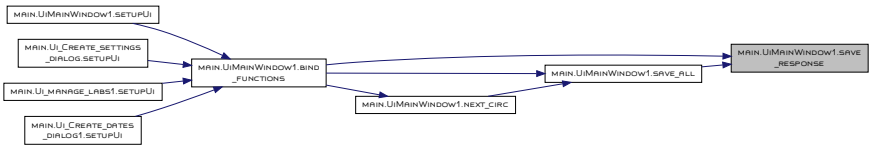
Definition at line 1043 of file main.py.

```
01043         self.grader_ref.user_comment = self.input_response_browser_user.toPlainText()
01044         self.grader_ref.save_responce()
01045
01046     def save_all(self):
01047         """
```

References main.UiMainWindow1.grader_ref, main_window.Ui_mainWindow.input_response_browser, and main_window.Ui_mainWindow.input_response_browser_user.

Referenced by main.UiMainWindow1.bind_functions(), and main.UiMainWindow1.save_all().

Here is the caller graph for this function:



7.17.3.23 setupUi() def main.UiMainWindow1.setupUi (self, main_window)

Reimplemented from main_window.Ui_mainWindow.

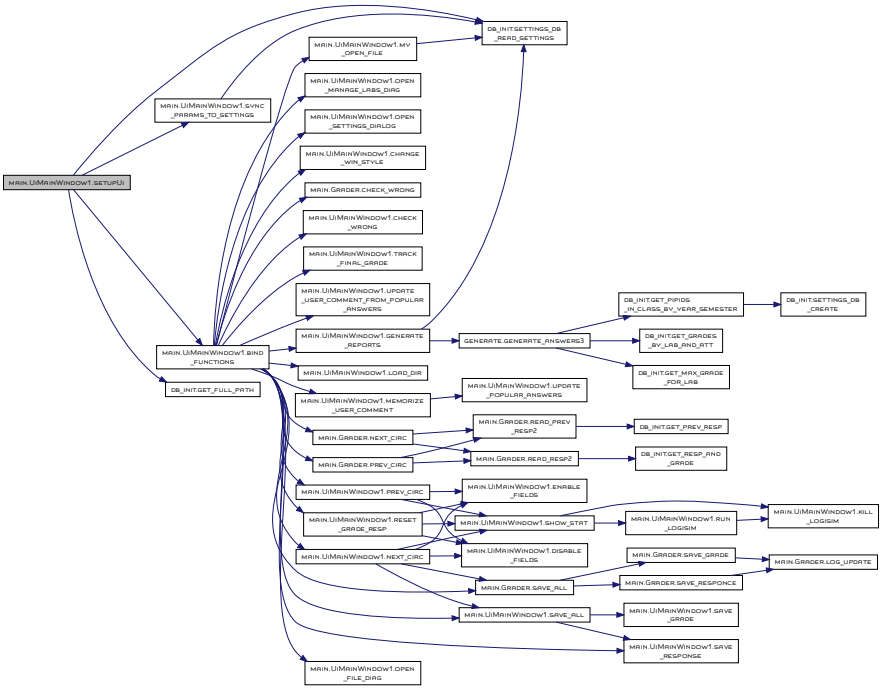
Definition at line 1076 of file main.py.

```
01076
01077         self.bind_functions()
01078         self.sync_params_to_settings()
01079
01080         from pathlib import Path
01081         settings_location = str(Path(os.path.expandvars(os.path.expanduser('./settings.sqlite3'))).absolute())
01082         if os.path.isfile(settings_location):
01083             paths, local = settings_db_read_settings()
01084             try:
01085                 if len(os.walk(get_full_path(paths, local) + "/server_sync/").__next__()[1]) > 0:
01086                     if not self.manage_labs_but.isEnabled():
01087                         self.manage_labs_but.setEnabled(True)
01088                     if not self.but_file_open.isEnabled():
01089                         self.but_file_open.setEnabled(True)
01090                         self.input_file_location.setEnabled(True)
01091             except Exception as e:
01092                 print("Most likely you did not fill all the settings: ", e)
01093
01094
01095     def sync_params_to_settings(self):
```

```
01096         """
```

References main.UiMainWindow1.bind_functions(), main_window.Ui_mainWindow.but_file_open, db_init.get_full_path(), main_window.Ui_mainWindow.input_file_location, main_window.Ui_mainWindow.manage_labs_but, db_init.settings_db_read_settings(), and main.UiMainWindow1.sync_params_to_settings().

Here is the call graph for this function:



7.17.3.24 show_stat() def main.UiMainWindow1.show_stat (self)

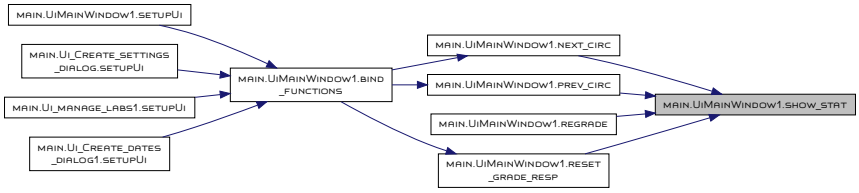
Definition at line 867 of file main.py.

```
00867         file_path = os.path.join(self.grader_ref.lab_paths[self.grader_ref.cur_idx], self.grader_ref.circ_file_name)
00868         if not Path(file_path).is_file():
00869             self.kill_logisim()
00870             self.grader_ref.final_grade = 0
00871             self.input_response_browser.setPlainText('File does not exist.')
00872             self.grader_ref.final_grade = 0
00873         else:
00874             if self.but_regrade.text() == '&GRADE' or self.but_regrade.text() == 'GRADE':
00875                 try:
00876                     self.run_logisim(file_path)
00877                 except Exception as e:
00878                     print('Error in run_logisim: ', e)
00879                     print(sys.exc_info()[0])
00880 
00881         self.input_current_id.setText(self.class_id_to_id[self.grader_ref.get_stud_id()])
00882         self.dateTimeEdit_submitted.setDateTime(QDateTime.fromSecsSinceEpoch(self.grader_ref.timestamps[self.grader_ref.cur_idx]))
00883         self.input_subtract.setText("")
00884         self.input_final_grade.setText(str(self.grader_ref.final_grade))
00885         self.input_log_browser.setText(self.grader_ref.global_log)
00886         self.input_response_browser.setPlainText(self.grader_ref.resp_text)
00887         self.input_response_browser_user.setPlainText(self.grader_ref.user_comment)
00888         self.checkB_input_pin_status.setChecked(False)
00889         self.checkB_output_pin_status.setChecked(False)
00890         self.popular_answers.setCurrentIndex(-1)
00891 
00892     def check_file(self):
00893         """
```

References main_window.Ui_mainWindow.but_regrade, main_window.Ui_mainWindow.checkB_input_pin_status, main_window.Ui_mainWindow.checkB_output_pin_status, main.UiMainWindow1.class_id_to_id, main_window.Ui_mainWindow.dateTimeEdit_submitted, main.UiMainWindow1.grader_ref, main_window.Ui_mainWindow.input_current_id, main_window.Ui_mainWindow.input_final_grade, main_window.Ui_mainWindow.input_log_browser, main_window.Ui_mainWindow.input_prev_response, main_window.Ui_mainWindow.input_response_browser,

main_window.Ui_mainWindow.input_response_browser_user, main_window.Ui_mainWindow.input_subtract, main.UiMainWindow1.kill_logisim(), main_window.Ui_mainWindow.popular_answers, and main.UiMainWindow1.run_logisim().
Referenced by main.UiMainWindow1.next_circ(), main.UiMainWindow1.prev_circ(), main.UiMainWindow1.regrade(), and main.UiMainWindow1.reset_grade_resp().
Here is the call graph for this function:

Here is the caller graph for this function:

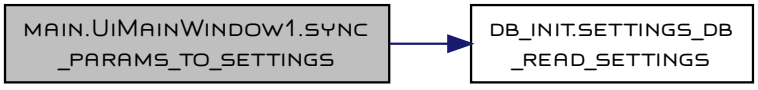


7.17.3.25 sync_params_to_settings() def main.UiMainWindow1.sync_params_to_settings (self)

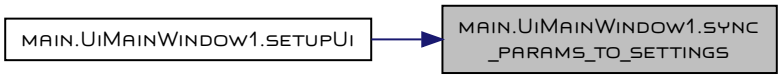
Definition at line 1101 of file main.py.

```
01101         working_dir = ''
01102         if paths and len(paths) == 4:
01103             self.logisim_path = paths[0]
01104             if len(paths[1]) > 0:
01105                 working_dir = paths[1]
01106             else:
01107                 working_dir = './'
01108         if local and len(local) >= 4:
01109             self.grader_name = local[0]
01110             working_dir += str(local[1])
01111             working_dir += '_' + local[2] + '/'
01112             self.set_style_checkbox.setChecked(bool(local[3]))
01113 
01114         if len(working_dir) > 0:
01115             self.input_file_location.setText(os.path.expanduser(working_dir))
01116 
01117 
01118     def bind_functions(self):
01119         """
```

References db_init.settings_db_read_settings().
Referenced by main.UiMainWindow1.setupUi().
Here is the call graph for this function:



Here is the caller graph for this function:



7.17.3.26 track_final_grade() def main.UiMainWindow1.track_final_grade (self)

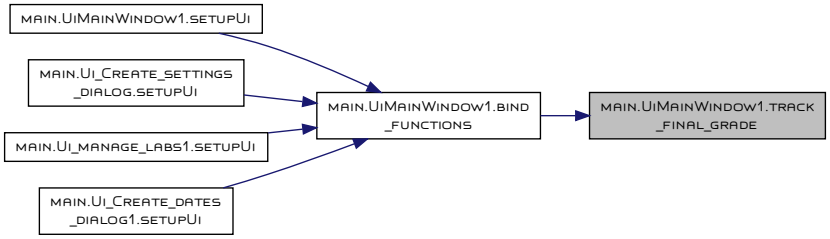
Definition at line 1063 of file main.py.

```
01063         self.grader_ref.log_update('Manual grade change from : ' + str(self.grader_ref.final_grade))
01064         self.input_log_browser.setText(self.grader_ref.global_log)
01065         self.grader_ref.final_grade = int(grade)
01066         self.grader_ref.log_update('Manual grade change to: ' + str(grade))
01067         self.input_log_browser.setText(self.grader_ref.global_log)
01068 
01069     def setupUi(self, main_window):
01070         """
```

References main.UiMainWindow1.grader_ref, main_window.Ui_mainWindow.input_final_grade, and main_window.Ui_mainWindow.input_log_browser.

Referenced by main.UiMainWindow1.bind_functions().

Here is the caller graph for this function:



7.17.3.27 update_popular_answers() def main.UiMainWindow1.update_popular_answers (self)

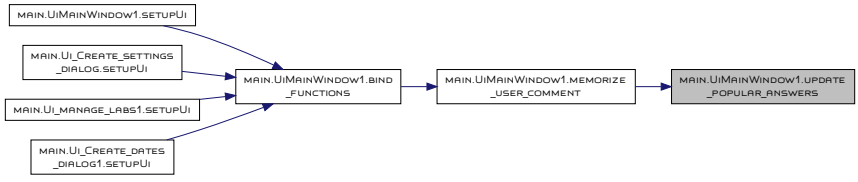
Definition at line 1024 of file main.py.

```
01024             self.popular_answers.clear()
01025             self.popular_answers.addItems(self.grader_ref.input_suggestion)
01026             # for item in self.grader_ref.input_suggestion:
01027 
01028     def save_grade(self):
01029         """
```

References main.UiMainWindow1.grader_ref, and main_window.Ui_mainWindow.popular_answers.

Referenced by main.UiMainWindow1.memorize_user_comment().

Here is the caller graph for this function:



7.17.3.28 update_user_comment_from_popular_answers()

```
 def main.UiMainWindow1.update_user_comment_from_popular_answers (
                  self )
```

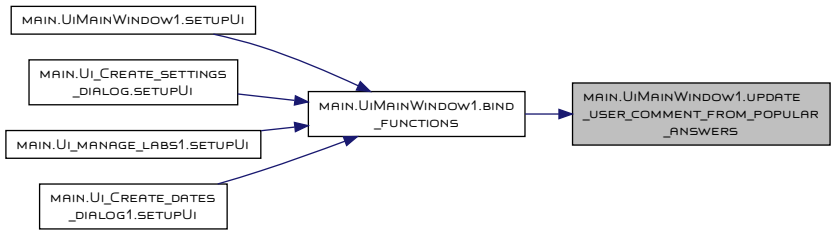
Definition at line 1171 of file main.py.

```
01171             self.input_response_browser_user.setPlainText(self.popular_answers.currentText())
01172
01173     def open_file_diag(self):
01174         """
```

References main_window.Ui_mainWindow.input_response_browser_user, and main_window.Ui_mainWindow.popular_answers.

Referenced by main.UiMainWindow1.bind_functions().

Here is the caller graph for this function:



7.17.4 Member Data Documentation

7.17.4.1 cal_window main.UiMainWindow1.cal_window

Definition at line 724 of file main.py.

7.17.4.2 class_id_to_id main.UiMainWindow1.class_id_to_id

Definition at line 770 of file main.py.

Referenced by main.UiMainWindow1.show_stat().

7.17.4.3 current_tz main.UiMainWindow1.current_tz

Definition at line 818 of file main.py.

7.17.4.4 grader_name main.UiMainWindow1.grader_name

Definition at line 1111 of file main.py.

7.17.4.5 grader_ref main.UiMainWindow1.grader_ref

Definition at line 723 of file main.py.

Referenced by main.UiMainWindow1.check_file(), main.UiMainWindow1.check_wrong(), main.UiMainWindow1.generate_reports(), main.UiMainWindow1.kill_logisim(), main.UiMainWindow1.load_dir(), main.UiMainWindow1.memorize_user_comment(), main.UiMainWindow1.next_circ(), main.UiMainWindow1.prev_circ(), main.UiMainWindow1.regrade(), main.UiMainWindow1.reset_grade_resp(), main.UiMainWindow1.run_logisim(), main.UiMainWindow1.save_all(), main.UiMainWindow1.save_grade(), main.UiMainWindow1.save_response(), main.UiMainWindow1.show_stat(), main.UiMainWindow1.track_final_grade(), and main.UiMainWindow1.update_popular_answers().

7.17.4.6 logisim_path main.UiMainWindow1.logisim_path

Definition at line 1105 of file main.py.

Referenced by main.UiMainWindow1.run_logisim().

7.17.4.7 manage_labs_window main.UiMainWindow1.manage_labs_window

Definition at line 1320 of file main.py.

7.17.4.8 settings_window main.UiMainWindow1.settings_window

Definition at line 1288 of file main.py.

7.17.4.9 working_dir main.UiMainWindow1.working_dir

Definition at line 725 of file main.py.

The documentation for this class was generated from the following file:

- main.py

8 File Documentation

8.1 create_dates_diag.py File Reference

Classes

- class create_dates_diag.Ui_Create_dates_dialog

Namespaces

- create_dates_diag

8.2 create_dates_diag.py

```
00001 # -*- coding: utf-8 -*-
00002
00003 # Form implementation generated from reading ui file 'create_dates_diag.ui'
00004 #
00005 # Created by: PyQt5 UI code generator 5.12.dev1812231618
00006 #
00007 # WARNING! All changes made in this file will be lost!
00008
00009 from PyQt5 import QtCore, QtGui, QtWidgets
00010
00011 class Ui_Create_dates_dialog(object):
00012     def setupUi(self, Create_dates_dialog):
00013         Create_dates_dialog.setObjectName("Create_dates_dialog")
00014         Create_dates_dialog.resize(589, 250)
00015         Create_dates_dialog.setMinimumSize(QtCore.QSize(500, 250))
00016         Create_dates_dialog.setMaximumSize(QtCore.QSize(1000, 300))
00017         icon = QtGui.QIcon()
00018         icon.addPixmap(QtGui.QPixmap("os_linux_1.ico"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
00019         Create_dates_dialog.setWindowIcon(icon)
00020         self.verticalLayout = QtWidgets.QVBoxLayout(Create_dates_dialog)
00021         self.verticalLayout.setObjectName("verticalLayout")
00022         self.horizontalLayout_5 = QtWidgets.QHBoxLayout()
00023         self.horizontalLayout_5.setObjectName("horizontalLayout_5")
00024         self.lab_path = BetterLineEdit(Create_dates_dialog)
00025         self.lab_path.setFocusPolicy(QtCore.Qt.StrongFocus)
00026         self.lab_path.setStatusTip("")
00027         self.lab_path.setWhatsThis("")
00028         self.lab_path.setAccessibleName("")
00029         self.lab_path.setAccessibleDescription("")
00030         self.lab_path.setInputMask("")
00031         self.lab_path.setReadOnly(False)
00032         self.lab_path.setCursorMoveStyle(QtCore.Qt.LogicalMoveStyle)
00033         self.lab_path.setClearButtonEnabled(False)
00034         self.lab_path.setObjectName("lab_path")
00035         self.horizontalLayout_5.addWidget(self.lab_path)
00036         self.verticalLayout.addLayout(self.horizontalLayout_5)
00037         self.horizontalLayout = QtWidgets.QHBoxLayout()
00038         self.horizontalLayout.setObjectName("horizontalLayout")
00039         self.init_subm_date_time = QtWidgets.QDateTimeEdit(Create_dates_dialog)
00040         self.init_subm_date_time.setMaximumSize(QtCore.QSize(150, 40))
00041         self.init_subm_date_time.setCalendarPopup(True)
00042         self.init_subm_date_time.setObjectName("init_subm_date_time")
00043         self.horizontalLayout.addWidget(self.init_subm_date_time)
00044         self.init_label = QtWidgets.QLabel(Create_dates_dialog)
00045         self.init_label.setObjectName("init_label")
00046         self.horizontalLayout.addWidget(self.init_label)
00047         self.verticalLayout.addLayout(self.horizontalLayout)
00048         self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
00049         self.horizontalLayout_2.setObjectName("horizontalLayout_2")
00050         self.first_subm_date_time = QtWidgets.QDateTimeEdit(Create_dates_dialog)
00051         self.first_subm_date_time.setMaximumSize(QtCore.QSize(150, 35))
00052         self.first_subm_date_time.setCalendarPopup(True)
00053         self.first_subm_date_time.setObjectName("first_subm_date_time")
00054         self.horizontalLayout_2.addWidget(self.first_subm_date_time)
00055         self.first_label = QtWidgets.QLabel(Create_dates_dialog)
00056         self.first_label.setObjectName("first_label")
00057         self.horizontalLayout_2.addWidget(self.first_label)
00058         self.verticalLayout.addLayout(self.horizontalLayout_2)
00059         self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
00060         self.horizontalLayout_3.setObjectName("horizontalLayout_3")
00061         self.second_subm_date_time = QtWidgets.QDateTimeEdit(Create_dates_dialog)
00062         self.second_subm_date_time.setMaximumSize(QtCore.QSize(150, 35))
00063         self.second_subm_date_time.setCalendarPopup(True)
00064         self.second_subm_date_time.setObjectName("second_subm_date_time")
00065         self.horizontalLayout_3.addWidget(self.second_subm_date_time)
00066         self.second_label = QtWidgets.QLabel(Create_dates_dialog)
```

```
00067         self.second_label.setObjectName("second_label")
00068         self.horizontalLayout_3.addWidget(self.second_label)
00069         self.verticalLayout.addLayout(self.horizontalLayout_3)
00070         self.horizontalLayout_4 = QtWidgets.QHBoxLayout()
00071         self.horizontalLayout_4.setObjectName("horizontalLayout_4")
00072         self.third_subm_date_time = QtWidgets.QDateTimeEdit(Create_dates_dialog)
00073         self.third_subm_date_time.setMaximumSize(QtCore.QSize(150, 35))
00074         self.third_subm_date_time.setCalendarPopup(True)
00075         self.third_subm_date_time.setObjectName("third_subm_date_time")
00076         self.horizontalLayout_4.addWidget(self.third_subm_date_time)
00077         self.third_label = QtWidgets.QLabel(Create_dates_dialog)
00078         self.third_label.setObjectName("third_label")
00079         self.horizontalLayout_4.addWidget(self.third_label)
00080         self.verticalLayout.addLayout(self.horizontalLayout_4)
00081         self.buttonBox = QtWidgets.QDialogButtonBox(Create_dates_dialog)
00082         self.buttonBox.setMaximumSize(QtCore.QSize(16777215, 40))
00083         self.buttonBox.setOrientation(QtCore.Qt.Horizontal)
00084         self.buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Abort|QtWidgets.QDialogButtonBox.SaveAll)
00085         self.buttonBox.setObjectName("buttonBox")
00086         self.verticalLayout.addWidget(self.buttonBox)
00087 
00088         self.retranslateUi(Create_dates_dialog)
00089         self.buttonBox.accepted.connect(Create_dates_dialog.accept)
00090         self.buttonBox.rejected.connect(Create_dates_dialog.reject)
00091         QtCore.QMetaObject.connectSlotsByName(Create_dates_dialog)
00092         Create_dates_dialog.setTabOrder(self.init_subm_date_time, self.first_subm_date_time)
00093         Create_dates_dialog.setTabOrder(self.first_subm_date_time, self.second_subm_date_time)
00094         Create_dates_dialog.setTabOrder(self.second_subm_date_time, self.third_subm_date_time)
00095 
00096     def retranslateUi(self, Create_dates_dialog):
00097         _translate = QtCore.QCoreApplication.translate
00098         Create_dates_dialog.setWindowTitle(_translate("Create_dates_dialog", "Dialog"))
00099         self.lab_path.setToolTip(_translate("Create_dates_dialog", "Tripple for file dialog"))
00100         self.lab_path.setPlaceholderText(_translate("Create_dates_dialog", "DoubleClick to select path"))
00101         self.init_label.setText(_translate("Create_dates_dialog", "Submission date"))
00102         self.first_label.setText(_translate("Create_dates_dialog", "1st resubmission"))
00103         self.second_label.setText(_translate("Create_dates_dialog", "2nd resubmission"))
00104         self.third_label.setText(_translate("Create_dates_dialog", "3rd resubmission"))
00105 
00106 from qt_class_improvements import BetterLineEdit
```

8.3 dates_window.py File Reference

Classes

- class dates_window.Ui_dates_window

Namespaces

- dates_window

8.4 dates_window.py

```
00001 # -*- coding: utf-8 -*-
00002 
00003 # Form implementation generated from reading ui file 'dates_window.ui'
00004 #
00005 # Created by: PyQt5 UI code generator 5.10.1
00006 #
00007 # WARNING! All changes made in this file will be lost!
00008 
00009 from PyQt5 import QtCore, QtGui, QtWidgets
00010 
00011 class Ui_dates_window(object):
00012     def setupUi(self, dates_window):
00013         dates_window.setObjectName("dates_window")
00014         dates_window.resize(251, 314)
00015         self.buttonBox = QtWidgets.QDialogButtonBox(dates_window)
00016         self.buttonBox.setGeometry(QtCore.QRect(40, 260, 191, 32))
00017         self.buttonBox.setOrientation(QtCore.Qt.Horizontal)
00018         self.buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Ok)
00019         self.buttonBox.setObjectName("buttonBox")
00020         self.calendarWidget = QtWidgets.QCalendarWidget(dates_window)
00021         self.calendarWidget.setGeometry(QtCore.QRect(10, 10, 224, 232))
00022         self.calendarWidget.setObjectName("calendarWidget")
00023 
00024         self.retranslateUi(dates_window)
00025         self.buttonBox.accepted.connect(dates_window.accept)
00026         self.buttonBox.rejected.connect(dates_window.reject)
00027         QtCore.QMetaObject.connectSlotsByName(dates_window)
```

```
00028
00029     def retranslateUi(self, dates_window):
00030         _translate = QtCore.QCoreApplication.translate
00031         dates_window.setWindowTitle(_translate("dates_window", "Check dates"))
00032         self.calendarWidget.setAccessibleName(_translate("dates_window", "cal_diag"))
00033
```

8.5 db_init.py File Reference

Namespaces

- db_init

Functions

- def db_init.settings_db_create (db_name=SETTINGS_DB_NAME, force=False)
- def db_init.settings_db_read_settings (db_name=SETTINGS_DB_NAME)
- def db_init.update_settings (paths, local, db_name=SETTINGS_DB_NAME)
- def db_init.grades_db_create (db_name, force=False)
- def db_init.load_student_list_into_grades_db (db_name, year, semester, filename='students_list3.txt')
- def db_init.insert_students (ids, fname, lname, db_name='./grades.sqlite3')
- def db_init.register_students_in_class (pipeline_ids, year, semester, db_name='./grades.sqlite3')
- def db_init.get_pipeline_ids (db_name='./grades.sqlite3')
- def db_init.get_ids_in_class_by_year_semester (year, semester, db_name='./grades.sqlite3')
- def db_init.import_previous_grades_into_db (year, semester, db_name='./grades.sqlite3', filename='./grades.xls')
- def db_init.gen_filenotfound_resp (lab_id, stud_path, corr_file, grader, att=None, next_date=None, db_name='./grades.sqlite3')
- def db_init.get_resp_and_grade (grade_id, db_name='./grades.sqlite3')
- def db_init.get_prev_resp (grade_id, class_id, lab_id, db_name='./grades.sqlite3')
- def db_init.save_a_grade_to_db (grade_id, grade, grader_comment, extra_comment, grader_name, graded=True, pass_fail=True, db_name='./grades.↩ sqlite3')
- def db_init.init_new_lab (stud_id, lab_name, att, submitted, lab_path, db_name='./grades.sqlite3')
- def db_init.get_lab_names (db_name='./grades.sqlite3')
- def db_init.update_lab_submissions_paths (db_name, repository_root, year, semester)
- def db_init.get_empty_grades_by_lid (lab_id, att, db_name='./grades.sqlite3')
- def db_init.get_all_grades_by_lid (lab_id, att, db_name='./grades.sqlite3')
- def db_init.reconstruct_grades_and_comments (db_name='./grades.sqlite3')
- def db_init.generate_final_grades (db_name, year, semester)
- def db_init.get_max_grade_for_lab (lid, year, semester, db_name='./grades.sqlite3')
- def db_init.get_grades_by_lab_and_att (lid, att, db_name='./grades.sqlite3')
- def db_init.get_lab_filename (lab_id, db_name='./grades.sqlite3')
- def db_init.get_lab_max_value (lab_id, db_name='./grades.sqlite3')
- def db_init.get_full_path (paths, local)
- def db_init.sync_files (self=None)
- def db_init.export_pdf (self=None)
- def db_init.save_grade_and_report (grade_id, grade, report, user_comment, grader, db_name='./grades.sqlite3')
- def db_init.commit_gen_report (grade_id, db_name='./grades.sqlite3')
- def db_init.get_lab_id (ltype, lab_num)
- def db_init.register_lab_in_semester (ltype, lab_num, year, semester, due_dates, db_name='./grades.sqlite3')
- def db_init.get_labid_in_schedule (lid, year, semester, db_name='./grades.sqlite3')
- def db_init.get_due_date_by_labid (lid_sem, att=None, db_name='./grades.sqlite3')
- def db_init.get_import_dates_by_labid (lid_sem, att=None, db_name='./grades.sqlite3')
- def db_init.gen_report (lid_sem, att=None, db_name='./grades.sqlite3')
- def db_init.get_pipids_in_class_by_year_semester (year, semester, db_name='./grades.sqlite3')

Variables

- string db_init.SETTINGS_DB_NAME = 'settings.sqlite3'

8.6 db_init.py

```
00001 #!/usr/bin/python3
00002
00003 import sqlite3 as lite
00004 import os
00005 import os.path
00006 # import pandas as pd
00007
00008 SETTINGS_DB_NAME = 'settings.sqlite3'  # Default settings filename
00009
00010
00011 def settings_db_create(db_name=SETTINGS_DB_NAME, force=False):
00012     """
00013     Creates sqlite3 database with settings in app's root directory.
00014     If found file with the same name - asks user to overwrite it, but only if force flag is False
00015     :param db_name: name of the database in app's root directory
00016     :param force: flag to overwrite existing db
```

```
00017     :return: False if wrong name was supplied, True after database was created
00018     """
00019     if not force and os.path.isfile(db_name):
00020         user_choice = input('Do you really want to drop database ? Type "yes" to continue\n ')
00021         if not user_choice.isalpha() or not user_choice.lower() == 'yes':
00022             return False
00023 
00024     # DB creation logic goes here
00025     with lite.connect(db_name) as con:
00026         cur = con.cursor()
00027         cur.execute('DROP TABLE IF EXISTS PATHS')
00028         cur.execute("CREATE TABLE PATHS "
00029                     "( LOGISIM_HOME VARCHAR NOT NULL,\
00030                        GRADING_PATH VARCHAR NOT NULL,\
00031                        IMPORT_PATH  VARCHAR,\
00032                        GRADES_DB  VARCHAR); ")
00033         cur.execute("CREATE TABLE LOCAL (\
00034                     GRADER_NAME VARCHAR,\
00035                     YEAR        INT,\
00036                     SEMESTER    CHAR (1),\
00037                     USE_STYLE   BOOLEAN,\
00038                      SYNC_COMMAND VARCHAR);")
00039         con.commit()
00040     return True
00041 
00042 
00043 def settings_db_read_settings(db_name=SETTINGS_DB_NAME):
00044     """
00045     Reads settings from the DB with specified name in 'db_name'
00046     :param db_name: name of DB to query
00047     :return: paths - list of paths to various locations and local - info about grader, grading year, etc.
00048     """
00049     paths = local = None
00050     if os.path.isfile(db_name):
00051         with lite.connect(db_name) as con:
00052             cur = con.cursor()
00053             result = cur.execute("SELECT LOGISIM_HOME, GRADING_PATH, IMPORT_PATH, GRADES_DB\
00054                                   FROM PATHS")
00055             paths = result.fetchone()
00056             result = cur.execute("SELECT GRADER_NAME, YEAR, SEMESTER, USE_STYLE, SYNC_COMMAND\
00057                                         FROM LOCAL")
00058             local = result.fetchone()
00059 
00060     return paths, local
00061 
00062 
00063 def update_settings(paths, local, db_name=SETTINGS_DB_NAME):
00064     """
00065     Procedure that loads parameters specified in paths and local into settings DB
00066     :param paths: list of paths to various locations
00067     :param local: local - info about grader, grading year, etc.
00068     :param db_name: name of DB to query to update
00069     :return: nothing
00070     """
00071     if os.path.isfile(db_name):
00072         with lite.connect(db_name) as con:
00073             cur = con.cursor()
00074             cur.execute('DELETE FROM PATHS;')
00075             cur.execute('INSERT OR REPLACE INTO PATHS (LOGISIM_HOME, GRADING_PATH, IMPORT_PATH, GRADES_DB)'
00076                         ' VALUES (?, ?, ?, ?);', paths)
00077             cur.execute('DELETE FROM LOCAL;')
00078             cur.execute('INSERT OR REPLACE INTO LOCAL (GRADER_NAME, YEAR, SEMESTER, USE_STYLE, SYNC_COMMAND)'
00079                         'VALUES (?, ?, ?, ?, ?);', local)
00080             con.commit()
00081 
00082         with lite.connect(db_name) as con:
00083             cur = con.cursor()
00084             cur.execute('VACUUM;')
00085             con.commit()
00086 
00087 
00088 def grades_db_create(db_name, force=False):
00089     """
00090     Will create database that contains all information about grades
00091     :param db_name: path and name of the database
00092     :param force: flag to overwrite existing db
00093     :return: Unknown
00094     """
00095     # from pathlib import Path
00096     print("I am going to create a grades DB with next name: ", db_name)
00097     db_name = str(db_name)
```

```
00098     if not os.path.isfile(db_name) or force:
00099         # compute some vars before the connection
00100         lab_names = list()
00101         for i in range(1, 13):
00102             lab_names.append(('CLA' + str(i), 'Closed', i, 10))
00103         for i in range(1, 9):
00104             lab_names.append(('OLA' + str(i), 'Open', i, 20))
00105         lab_names.append(('OLA9', 'Open', 9, 100))
00106         a = list(zip(*lab_names))
00107         a.append(('initial_labs.circ', 'initial_labs.circ', 'seven_seg.circ', 'RSC.circ', 'custom_reg.circ', 'RSC.circ', 'RSC.circ',
       'RSC.circ', 'PLDs.circ', 'PLDs.circ', 'PLDs.circ', '', ",", 'bin_converter.circ', 'mod_counter.circ', 'custom_reg.circ', 'RSC.circ',
       'RSC.circ', 'ram2.txt', ''))
00108         b = list(zip(*a))
00109 
00110         with lite.connect(db_name) as con:
00111             cur = con.cursor()
00112             # TODO: force should remove 'IF NOT EXISTS' and add 'DROP TABLE' to ensure new table creation
00113             # WISH: add TRY blocks for each CREATE and spawn new info window in case of error
00114             print('Creating students...')
00115             cur.execute("""CREATE TABLE students (
00116                            pipeline_id    TEXT    NOT NULL
00117                                                   PRIMARY KEY,
00118                            first_name     TEXT    NOT NULL,
00119                            second_name    TEXT    NOT NULL,
00120                            comment        TEXT,
00121                            cheating_ratio INTEGER DEFAULT (0)    );""")
00122             con.commit()
00123             print('Done.')
00124             print('Creating semesters...')
00125             cur.execute("""CREATE TABLE semesters (
00126                            semester CHAR (1) NOT NULL PRIMARY KEY,
00127                            name     VARCHAR      );""")
00128             con.commit()
00129             print('Done.')
00130             print('Creating class...')
00131             cur.execute("""CREATE TABLE class (
00132                            id             INTEGER PRIMARY KEY AUTOINCREMENT,
00133                            pipeline_id    TEXT    REFERENCES students (pipeline_id),
00134                            year           INTEGER,
00135                            semester       INTEGER REFERENCES semesters (semester),
00136                            cheating_ratio INTEGER DEFAULT (0),
00137                            UNIQUE (
00138                                pipeline_id,
00139                                year,
00140                                semester)     );""")
00141             con.commit()
00142             print('Done.')
00143             print('Creating labs...')
00144             cur.execute("""CREATE TABLE lab_names (
00145                            id              INT     NOT NULL PRIMARY KEY,
00146                            type            TEXT    NOT NULL,
00147                            num             INTEGER NOT NULL,
00148                            max_grade       INTEGER NOT NULL,
00149                            name            VARCHAR,
00150                            description     VARCHAR,
00151                            grader_comment  VARCHAR,
00152                            mandatory_files VARCHAR );""")
00153             con.commit()
00154             print('Done.')
00155             print('Creating grades...')
00156             cur.execute("""CREATE TABLE grades (
00157                            id               INTEGER PRIMARY KEY AUTOINCREMENT,
00158                            class_id                 NOT NULL
00159                                                     REFERENCES class (id) ON UPDATE CASCADE,
00160                            lab                      NOT NULL
00161                                                     REFERENCES lab_names (id) ON UPDATE CASCADE,
00162                            attempt          INT     DEFAULT (0),
00163                            submitted        INTEGER,
00164                            graded           INTEGER,
00165                            grade            INTEGER NOT NULL
00166                                                     DEFAULT (0),
00167                            pass_fail        BOOLEAN NOT NULL
00168                                                     DEFAULT (0),
00169                            grader_comment   TEXT,
00170                            extra_comment    TEXT,
00171                            report_generated BOOLEAN,
00172                            report_time      INTEGER,
00173                            lab_path         VARCHAR,
00174                            UNIQUE (
00175                                class_id,
00176                                lab,
```

```
00177                               attempt,
00178                               pass_fail) ON CONFLICT REPLACE );""")
00179             con.commit()
00180             print('Done.')
00181 
00182             print('Creating lab schedule...')
00183             cur.execute("""CREATE TABLE lab_schedule (
00184                             id         INTEGER PRIMARY KEY AUTOINCREMENT,
00185                             lab_id             REFERENCES lab_names (id),
00186                             year       INTEGER NOT NULL,
00187                             semester   INTEGER REFERENCES semesters (semester)
00188                                        NOT NULL,
00189                             due_date_1 INTEGER,
00190                             due_date_2 INTEGER,
00191                             due_date_3 INTEGER,
00192                             due_date_4 INTEGER,
00193                             imported_1 INTEGER,
00194                             imported_2 INTEGER,
00195                             imported_3 INTEGER,
00196                             imported_4 INTEGER,
00197                             posted_1   INTEGER,
00198                             posted_2   INTEGER,
00199                             posted_3   INTEGER,
00200                             posted_4   INTEGER
00201                         );""")
00202             con.commit()
00203             print('Done.')
00204 
00205 
00206 
00207             print('Filling semesters...')
00208             cur.executemany('INSERT OR REPLACE INTO semesters\
00209                         (semester, name) VALUES (?, ?)', [(1, 'SPRING'), (2, 'SUMMER'), (3, 'FALL')])
00210             con.commit()
00211             print('Done.')
00212             print('Filling labs...')
00213             cur.executemany('INSERT OR REPLACE INTO lab_names\
00214                         (id, type, num, max_grade, mandatory_files) VALUES (?, ?, ?, ?, ?)', b)
00215             con.commit()
00216             print('Done.')
00217             print('Vacuuming...')
00218 
00219             cur.execute('VACUUM;')
00220             con.commit()
00221 
00222             print('Done.')
00223             print('Creation of GRADES DB finished.')
00224 
00225             return True
00226 
00227 
00228 def load_student_list_into_grades_db(db_name, year, semester, filename='students_list3.txt'):
00229     """
00230     Imports list of students from file in format: `id % lname, fname` into Grades DB.
00231     Should be called before first grading.
00232     :param db_name: db that contains grades and student info
00233     :param year: grading (current) year
00234     :param semester: grading (current) semester
00235     :param filename: file that contains student list
00236     :return: nothing
00237     """
00238 
00239     with open(filename, 'r') as sl:
00240         ids, names = zip(*(line.strip().split('%') for line in sl))
00241         ids = list(sid.strip() for sid in ids)
00242         names = (name.strip() for name in names)  # for case when file contains extra whitespaces
00243         lname, fname = zip(*(namer.split(',') for namer in names))
00244         lname = (name.strip() for name in lname)
00245         fname = (name.strip() for name in fname)
00246 
00247     if os.path.isfile(db_name):
00248         insert_students(ids, fname, lname, db_name)
00249         register_students_in_class(ids, year, semester, db_name)
00250 
00251 
00252 def insert_students(ids, fname, lname, db_name='./grades.sqlite3'):
00253     """
00254     Takes students' info from the parameters and insert them into grades DB
00255     :param ids: pipeline ids
00256     :param fname: first name
00257     :param lname: last name
```

```
00258     :param db_name: specific name for grades DB
00259     :return: nothing
00260     """
00261     names_tupple = list(zip(ids, fname, lname, [0] * len(ids)))
00262     with lite.connect(db_name) as con:
00263         cur = con.cursor()
00264         cur.executemany('INSERT OR REPLACE INTO STUDENTS \
00265                     (pipeline_id, first_name, second_name, cheating_ratio)'
00266                         ' VALUES (?, ?, ?, ?)', names_tupple)
00267         con.commit()
00268
00269
00270 def register_students_in_class(pipeline_ids, year, semester, db_name='./grades.sqlite3'):
00271     """
00272
00273     :param pipeline_ids:
00274     :param year:
00275     :param semester:
00276     :param db_name:
00277     :return:
00278     """
00279     len_id = len(pipeline_ids)
00280     names_tupple = list(zip(pipeline_ids, [year] * len_id, [semester] * len_id, [0] * len_id))
00281     with lite.connect(db_name) as con:
00282         cur = con.cursor()
00283         cur.executemany('INSERT OR REPLACE INTO class\
00284                     (pipeline_id, year, semester, cheating_ratio) VALUES (?, ?, ?, ?)', names_tupple)
00285         con.commit()
00286
00287
00288 def get_pipeline_ids(db_name='./grades.sqlite3'):
00289     """
00290
00291     :param db_name:
00292     :return:
00293     """
00294     with lite.connect(db_name) as con:
00295         cur = con.cursor()
00296         result = cur.execute("SELECT pipeline_id FROM students")
00297         try:
00298             resut = (ids[0] for ids in result.fetchall())
00299         except Exception as e:
00300             print(e)
00301             return None
00302     return resut
00303
00304
00305 def get_ids_in_class_by_year_semester(year, semester, db_name='./grades.sqlite3'):
00306     """
00307
00308     :param year:
00309     :param semester:
00310     :param db_name:
00311     :return:
00312     """
00313     with lite.connect(db_name) as con:
00314         cur = con.cursor()
00315         result = cur.execute("SELECT pipeline_id, id FROM class\
00316                               WHERE year=" + str(year) + " and semester=" + str(semester))
00317         try:
00318             res = result.fetchall()
00319             pip_to_id = dict(res)
00320             to_id_to_pip = dict([(res_id[1], res_id[0]) for res_id in res])
00321         except Exception as e:
00322             print(e)
00323             return None
00324     return pip_to_id, to_id_to_pip
00325
00326
00327 def import_previous_grades_into_db(year, semester, db_name='./grades.sqlite3', filename='./grades.xls'):
00328     """
00329     Takes xls file with grades from previous semester(s) and loads all grades into DB.
00330     In case students are not found in the DB and xls file contains ids - loads them too
00331     :param year: year when grades were assigned
00332     :param semester: semester when grades were assigned
00333     :param db_name: specific name of the grades DB
00334     :param filename: xls file to load
00335     :return: nothing
00336     """
00337     if not os.path.isfile(db_name):
00338         raise Exception("DB not found")
```

```
00339
00340     df1 = pd.read_excel(filename)
00341
00342     try:
00343         cls = df1.filter(like='CL')
00344     except Exception as e:
00345         print(e)
00346         cls = None  # no CLA's found
00347
00348     try:
00349         ols = df1.filter(like='OL')
00350     except Exception as e:
00351         print(e)
00352         ols = None  # no OLAs found
00353
00354     try:
00355         ids = df1.filter(like='sername').values.ravel().tolist()
00356         ids_len = len(ids)
00357     except Exception as e:
00358         print('Was not able to parse user ids, check xls file you are trying to import: ', e)
00359         raise e  # may be improved in the future - strange case
00360     try:
00361         names = df1.filter(like='Name').values.ravel().tolist()
00362     except Exception as e:  # either does not exist or has different name
00363         print(e)
00364         names = None
00365
00366     class_dict = get_ids_in_class_by_year_semester(year, semester, db_name)
00367
00368     if (not class_dict and not names) or (class_dict and len(class_dict) < ids_len and not names):
00369         raise Exception('Did not find ids in table CLASS and did not find names in xls file')
00370     elif names and (not class_dict or (class_dict and len(class_dict) < ids_len)):
00371         print('Did not find existing students, but found names in xsl\nAdding new students...\n')
00372         existing_ids = get_pipeline_ids(db_name)
00373         need_to_update_students = False
00374         # otherwise just add ids to the class list
00375         if existing_ids:
00376             for sid in ids:
00377                 if sid not in existing_ids:
00378                     need_to_update_students = True
00379         else:
00380             need_to_update_students = True
00381
00382         if need_to_update_students:
00383             fname, lname = zip(*(name.split(', ') for name in names))
00384             fname = (name.strip() for name in fname)
00385             lname = (name.strip() for name in lname)
00386             insert_students(ids, fname, lname, db_name)
00387         register_students_in_class(ids, year, semester, db_name)
00388
00389     class_ids = [class_dict[sid] for sid in ids]
00390     if ols is None and cls is None or len(class_ids) == 0:
00391         raise Exception('No grades to load')
00392
00393     grades_tupples = list()
00394     if ols is not None:
00395         for lab_name in ols:
00396             grades = (str(grade) for grade in ols[lab_name].values)
00397             grades_tupples += list(zip(class_ids, [lab_name] * ids_len, [-1] * ids_len, grades, ['TRUE'] * ids_len))
00398
00399     if cls is not None:
00400         for lab_name in cls:
00401             grades = (str(grade) for grade in cls[lab_name].values)
00402             grades_tupples += list(zip(class_ids, [lab_name] * ids_len, [-1] * ids_len, grades, ['TRUE'] * ids_len))
00403
00404     with lite.connect(db_name) as con:
00405         cur = con.cursor()
00406         cur.executemany('INSERT OR REPLACE INTO grades\
00407                     (class_id, lab, attempt, grade, pass_fail) VALUES (?, ?, ?, ?, ?)', grades_tupples)
00408         con.commit()
00409
00410
00411 def gen_filenotfound_resp(lab_id, stud_path, corr_file, grader, att=None, next_date=None, db_name='./grades.sqlite3'):
00412     resp_text = 'file with name "{}" was not found.</br>'.format(corr_file)
00413     file_found = os.listdir(stud_path)
00414     potential_files = list()
00415     for file in file_found:
00416         if file not in ['grade.txt', 'penalty.txt', 'responce.txt', 'tech_info.txt', ]:
00417             potential_files.append(file)
00418     if potential_files:
00419         resp_text += '\nNext files|folders were found:</br>\n'
```

```
00420     for file in potential_files:
00421         if os.path.isdir(os.path.join(stud_path, file)):
00422             resp_text += file + ' - directory.</br>\n'
00423         else:
00424             resp_text += file + ' - regular file.</br>\n'
00425 
00426     if att and att < 4 and next_date:
00427         resp_text += 'Please submit your file by next due date ({}).</br>\n'.format(next_date)
00428 
00429     if not os.path.isfile(db_name):
00430         raise Exception("DB not found")
00431     with lite.connect(db_name) as con:
00432         cur = con.cursor()
00433         cur.execute("UPDATE grades SET graded=strftime('%s','now'), pass_fail=FALSE, grader_comment=?, grader=? WHERE id=?", (resp_text,
      grader, lab_id))
00434         con.commit()
00435 
00436 
00437 def get_resp_and_grade(grade_id, db_name='./grades.sqlite3'):
00438     with lite.connect(db_name) as con:
00439         cur = con.cursor()
00440         result = cur.execute("SELECT grade, grader_comment, extra_comment, graded FROM grades WHERE id=?", (grade_id,))
00441         grade, resp, uresp, graded = result.fetchone()
00442 
00443     return grade, resp, uresp, graded
00444 
00445 
00446 def get_prev_resp(grade_id, class_id, lab_id, db_name='./grades.sqlite3'):
00447     with lite.connect(db_name) as con:
00448         cur = con.cursor()
00449         result = cur.execute("SELECT grader_comment, extra_comment FROM grades WHERE class_id=? AND lab=? AND id<?", (class_id, lab_id,
      grade_id))
00450         res = result.fetchall()
00451     if len(res) == 0:
00452         return ''
00453     else:
00454         gresp, uresp = zip(*res)
00455         return '\n'.join(('{} :\n{}'.format(gresp[i], uresp[i]) for i in range(len(gresp))))
00456 
00457 
00458 def save_a_grade_to_db(grade_id, grade, grader_comment, extra_comment, grader_name, graded=True, pass_fail=True, db_name='./grades.sqlite3'):
00459     pass
00460 
00461 
00462 # def get_submissions_to_grade(lab_id, att, db_name='./grades.sqlite3'):
00463 #     if not os.path.isfile(db_name):
00464 #         raise Exception("DB not found")
00465 #     with lite.connect(db_name) as con:
00466 #         cur = con.cursor()
00467 #         result = cur.execute("SELECT id, FROM grades where lab=lab_id attempt=att and graded is NULL")
00468 #         try:
00469 #             lab_id, lab_type, lab_num = zip(*result.fetchall())
00470 #         except Exception as e:
00471 #             print(e)
00472 #             return None, None, None
00473 #     return lab_id, lab_type, lab_num
00474 
00475 
00476 def init_new_lab(stud_id, lab_name, att, submitted, lab_path, db_name='./grades.sqlite3'):
00477     if not os.path.isfile(db_name):
00478         raise Exception("DB not found")
00479     with lite.connect(db_name) as con:
00480         cur = con.cursor()
00481         cur.execute('INSERT INTO grades (class_id, lab, attempt, submitted, lab_path) VALUES (?, ?, ?, ?, ?)', (stud_id, lab_name, att,
      submitted, lab_path))
00482         con.commit()
00483 
00484 
00485 def get_lab_names(db_name='./grades.sqlite3'):
00486     """
00487 
00488     :param db_name:
00489     :return:
00490     """
00491     with lite.connect(db_name) as con:
00492         cur = con.cursor()
00493         result = cur.execute("SELECT id, type, num FROM lab_names")
00494         try:
00495             lab_id, lab_type, lab_num = zip(*result.fetchall())
00496         except Exception as e:
00497             print(e)
```

```
00498             return None, None, None
00499     return lab_id, lab_type, lab_num
00500
00501
00502 def update_lab_submissions_paths(db_name, repository_root, year, semester):
00503     import fnmatch
00504     import glob
00505     # import_previous_grades_into_db(year, semester, db_name, repository_root+'grades.xlsx')
00506     lab_id, lab_type, lab_num = get_lab_names()
00507     if lab_id is None or lab_type is None or lab_num is None:
00508         raise Exception("Error during lab type/num import: ")
00509     class_dict = get_ids_in_class_by_year_semester(year, semester, db_name)
00510     total_labs = len(lab_type)
00511
00512     all_dirs = list()
00513     for lab_iter in range(total_labs):
00514         for attempt in range(1, 5):  # class rule - 4 attempts
00515             full_lab_name = repository_root + lab_type[lab_iter] + '_Lab_' + str(lab_num[lab_iter]) + '_' + str(attempt) + '/'
00516             print('Processing ', full_lab_name)
00517             for stud_id in class_dict.keys():
00518                 found_dir = glob.glob(full_lab_name+stud_id+'*')
00519                 if found_dir:
00520                     # since it is initial pass, we do not set pass/fail. It will be set later with grade and comment.
00521                     all_dirs.append((class_dict[stud_id], lab_id[lab_iter], attempt, 'FALSE', found_dir[-1]))
00522
00523     with lite.connect(db_name) as con:
00524         cur = con.cursor()
00525         cur.executemany('INSERT OR REPLACE INTO grades (class_id, lab, attempt, pass_fail, lab_path)'
00526                         ' VALUES (?, ?, ?, ?, ?)', all_dirs)
00527         con.commit()
00528
00529
00530 def get_empty_grades_by_lid(lab_id, att, db_name='./grades.sqlite3'):
00531     with lite.connect(db_name) as con:
00532         cur = con.cursor()
00533         result = cur.execute("SELECT submitted, class_id, id, lab_path FROM grades WHERE lab=? AND attempt=? AND graded is NULL", (lab_id,
      att))
00534         try:
00535             subm, class_id, lab_id, lab_path = zip(*result.fetchall())
00536         except Exception as e:
00537             # print(e)
00538             return None, None, None, None
00539
00540     return subm, class_id, lab_id, lab_path
00541
00542
00543 def get_all_grades_by_lid(lab_id, att, db_name='./grades.sqlite3'):
00544     with lite.connect(db_name) as con:
00545         cur = con.cursor()
00546         result = cur.execute("SELECT submitted, class_id, id, lab_path FROM grades WHERE lab=? AND attempt=? ", (lab_id, att))
00547         try:
00548             subm, class_id, lab_id, lab_path = zip(*result.fetchall())
00549         except Exception as e:
00550             print(e)
00551             return None, None
00552
00553     return subm, class_id, lab_id, lab_path
00554
00555
00556 def reconstruct_grades_and_comments(db_name='./grades.sqlite3'):
00557     lab_id, lab_path = get_empty_grades(db_name)
00558     updated_grades = list()
00559     for l_iter in range(len(lab_path)):
00560         lpath = lab_path[l_iter]
00561         submition_t = int(lpath.split('-')[-1])
00562         try:
00563             with open(lpath+'/grade.txt', 'r') as gfile:
00564                 cur_grade = int(gfile.readline().strip())
00565         except Exception as e:
00566             print("Error during grade file reading :", e)
00567             cur_grade = 0
00568         try:
00569             cur_t_graded = int(os.path.getmtime(lpath + '/grade.txt'))
00570         except Exception as e:
00571             print("Error during grade file statistics retrieval: ", e)
00572             cur_t_graded = None
00573
00574         pass_fail = 'TRUE' if cur_grade else 'FALSE'
00575         try:
00576             with open(lpath+'/responce.txt', 'r') as rfile:
00577                 cur_resp = rfile.readlines()
```

```
00578                 if type(cur_resp) == list:
00579                     cur_resp = ' '.join(cur_resp)
00580         except Exception as e:
00581             print("Error during grade file reading", e)
00582             cur_resp = 'NULL'
00583         updated_grades.append((submition_t, cur_grade, cur_t_graded, pass_fail, cur_resp, lab_id[l_iter]))
00584 
00585 
00586     with lite.connect(db_name) as con:
00587         cur = con.cursor()
00588         cur.executemany('UPDATE grades SET submitted=?, grade=?, graded=?, pass_fail=?, grader_comment=? '
00589                         'WHERE id=?', updated_grades)
00590         con.commit()
00591 
00592     with lite.connect(db_name) as con:
00593         cur = con.cursor()
00594         cur.execute('VACUUM;')
00595         con.commit()
00596 
00597 
00598 def generate_final_grades(db_name, year, semester):
00599     ids = get_ids_in_class_by_year_semester(year, semester, db_name)
00600     with lite.connect(db_name) as con:
00601         cur = con.cursor()
00602 
00603         labs = list()
00604         for sid in ids.values():  # using JOIN here will add too much extra data
00605             result = cur.execute('SELECT lab, MAX(grade * (select percent from penalties where id=GRADES.attempt)/100) '
00606                                  'FROM GRADES WHERE class_id=? and attempt > 0 group by lab order by lab', (str(sid),))
00607             labs.append(result.fetchall() )
00608 
00609         stud_info = list()
00610         for sid in ids.keys():
00611             result = cur.execute('SELECT first_name, second_name FROM students WHERE pipeline_id=?', (str(sid),))
00612             stud_info.append(result.fetchall() )
00613 
00614     df_stud_info = pd.DataFrame(dict(zip(ids.keys(), stud_info)))
00615     df_grades = pd.DataFrame(dict(zip(ids.keys(), labs)))
00616     # id_list = list(ids.keys())
00617     # a = id_list[list(ids.values()).index(class_id)]
00618 
00619 
00620 def get_max_grade_for_lab(lid, year, semester, db_name='./grades.sqlite3'):
00621     with lite.connect(db_name) as con:
00622         cur = con.cursor()
00623         result = cur.execute('SELECT e.pipeline_id as pipid, IFNULL(MAX(k.final_grade), 0) as grade '
00624                              'FROM class e '
00625                              'LEFT OUTER JOIN '
00626                              '  (SELECT d.pipeline_id, c.grade*f.percent/100 AS final_grade '
00627                              '   FROM grades c '
00628                              '     JOIN class d ON d.id = c.class_id '
00629                              '     JOIN penalties f ON f.id = c.attempt '
00630                              '   WHERE c.lab = ? ) k '
00631                              'ON e.pipeline_id = k.pipeline_id '
00632                              'WHERE year=? AND semester=? '
00633                              'GROUP BY e.pipeline_id '
00634                              'ORDER BY e.pipeline_id ', (lid, int(year), int(semester)))
00635         return result.fetchall()
00636 
00637 
00638 def get_grades_by_lab_and_att(lid, att, db_name='./grades.sqlite3'):
00639     with lite.connect(db_name, detect_types=lite.PARSE_COLNAMES) as con:
00640         cur = con.cursor()
00641         result = cur.execute('select a.due_date_{0} as due_date, a.imported_{0} as import_date, '
00642                              'b.type, b.num, b.max_grade, '
00643                              'c.id as grade_id, c.submitted, c.graded, c.grade, c.pass_fail, c.grader_comment, c.extra_comment, c.grader,
       c.lab_path, '
00644                              'd.pipeline_id, e.first_name, e.second_name, f.percent, c.grade*f.percent/100 as final_grade '
00645                              'from lab_schedule a '
00646                              'join lab_names b on a.lab_id=b.id '
00647                              'join grades c on c.lab=a.id '
00648                              'join class d on d.id=c.class_id '
00649                              'join students e on e.pipeline_id=d.pipeline_id '
00650                              'join penalties f on f.id=c.attempt '
00651                              'where c.attempt={0} AND a.id=? ORDER BY d.pipeline_id'.format(int(att)), (lid,))
00652         info_tup = result.fetchall()
00653         info_desc = result.description
00654     return info_tup, info_desc
00655 
00656 
00657 def get_lab_filename(lab_id, db_name='./grades.sqlite3'):
```

```
00658     with lite.connect(db_name) as con:
00659         cur = con.cursor()
00660 
00661         result = cur.execute('SELECT mandatory_files FROM lab_names WHERE id=? ', (str(lab_id),))
00662         return result.fetchall()[0]
00663     return None
00664 
00665 
00666 def get_lab_max_value(lab_id, db_name='./grades.sqlite3'):
00667     with lite.connect(db_name) as con:
00668         cur = con.cursor()
00669 
00670         result = cur.execute('SELECT max_grade FROM lab_names WHERE id=? ', (str(lab_id),))
00671         return int(result.fetchone()[0])
00672     return None
00673 
00674 
00675 def get_full_path(paths, local):
00676     import os
00677     return os.path.expanduser(paths[1]) + str(local[1]) + "_" + str(local[2])
00678 
00679 
00680 def sync_files(self=None):
00681     import subprocess
00682     import os
00683 
00684     paths, local = settings_db_read_settings()
00685     full_path = get_full_path(paths, local) + "/server_sync/"
00686     lab_ids, lab_types, lab_nums = get_lab_names()
00687     lab_names = []
00688     for i in range(len(lab_types)):
00689         lab_names.append(lab_types[i] + '_Lab_' + str(lab_nums[i]))
00690 
00691     if not os.path.isdir(full_path):
00692         os.makedirs(full_path)
00693         for lab_name in lab_names:
00694             os.makedirs(full_path + lab_name)
00695 
00696     proc_arr = []
00697     for lab_name in lab_names:
00698         command = local[4] + ' ' + os.path.expanduser(paths[2] + lab_name) + '/*.zip' + ' ' + full_path + lab_name + '/'
00699         try:
00700             proc_arr.append(subprocess.Popen(os.path.expandvars(command), stdout=subprocess.PIPE, shell=True))
00701             proc_arr[-1].communicate()
00702         except Exception as e:
00703             print('Error in rsync: ', e)
00704         # output, error = process.communicate()
00705         # print(output)
00706         # print(error)
00707 
00708     for proc_elem in proc_arr:
00709         proc_elem.wait()
00710 
00711 
00712 def export_pdf(self=None):
00713     import subprocess
00714     import os
00715 
00716     paths, local = settings_db_read_settings()
00717     lab_ids, lab_types, lab_nums = get_lab_names()
00718     lab_names = []
00719     for i in range(len(lab_types)):
00720         lab_names.append(lab_types[i] + '_Lab_' + str(lab_nums[i]))
00721 
00722     full_path = get_full_path(paths, local) + "/"
00723     for lab_name in lab_names:
00724         nums_to_sync = '_{'
00725         i = 1
00726         while os.path.isdir(full_path + lab_name + '_' + str(i) + '/Answers'):
00727             nums_to_sync += str(i) + ','
00728             i += 1
00729         if i == 1:
00730             continue
00731         nums_to_sync = nums_to_sync[0:-1] + '}'
00732         # for case when we have only one directory to sync
00733         if len(nums_to_sync) == 4:
00734             nums_to_sync = '_1'
00735         if len(nums_to_sync) > 1:
00736             command = local[4] + ' ' + full_path + lab_name + nums_to_sync + '/Answers/*.pdf ' + os.path.expanduser(paths[2]) + lab_name + '/'
00737             process = subprocess.Popen(os.path.expandvars(command), stdout=subprocess.PIPE, shell=True)
00738             process.communicate()
```

```
00739             # print(output)
00740             # print(error)
00741
00742
00743 def save_grade_and_report(grade_id, grade, report, user_comment, grader, db_name='./grades.sqlite3'):
00744     if not os.path.isfile(db_name):
00745         raise Exception("DB not found")
00746     with lite.connect(db_name) as con:
00747         cur = con.cursor()
00748         cur.execute("UPDATE grades SET graded=strftime('%s','now'), pass_fail=TRUE, grade=?, grader_comment=?, extra_comment=?, grader=? WHERE
      id=?", (grade, report, user_comment, grader, grade_id))
00749         con.commit()
00750
00751
00752 def commit_gen_report(grade_id, db_name='./grades.sqlite3'):
00753     if not os.path.isfile(db_name):
00754         raise Exception("DB not found")
00755     with lite.connect(db_name) as con:
00756         cur = con.cursor()
00757         cur.execute("UPDATE grades SET report_generated=strftime('%s','now') WHERE id=?", (grade_id,))
00758         con.commit()
00759
00760
00761
00762 def get_lab_id(ltype, lab_num):
00763     lab_ids, lab_types, lab_nums = get_lab_names()
00764     for i, lid in enumerate(lab_ids):
00765         if lab_types[i] == ltype and lab_num == lab_nums[i]:
00766             return lid
00767     return None
00768
00769
00770 def register_lab_in_semester(ltype, lab_num, year, semester, due_dates, db_name='./grades.sqlite3'):
00771     lid = get_lab_id(ltype, int(lab_num))
00772     # TODO: add a check so you do not insert lab twice
00773     if lid is None:
00774         raise Exception('No such lab')
00775     if not os.path.isfile(db_name):
00776         raise Exception("DB not found")
00777     with lite.connect(db_name) as con:
00778         cur = con.cursor()
00779         cur.execute('INSERT OR REPLACE INTO lab_schedule (lab_id, year, semester, due_date_1, due_date_2, due_date_3, due_date_4) VALUES (?, ?,
      ?, ?, ?, ?, ?)', (lid, year, semester, due_dates[0], due_dates[1], due_dates[2], due_dates[3]))
00780         con.commit()
00781
00782 def get_labid_in_schedule(lid, year, semester, db_name='./grades.sqlite3'):
00783     with lite.connect(db_name) as con:
00784         cur = con.cursor()
00785         result = cur.execute('SELECT id FROM lab_schedule WHERE lab_id=? AND year=? AND semester=?', (lid, year, semester))
00786         fetched_red = result.fetchone()
00787     return int(fetched_red[0]) if fetched_red is not None else None
00788
00789
00790 def get_due_date_by_labid(lid_sem, att=None, db_name='./grades.sqlite3'):
00791     with lite.connect(db_name) as con:
00792         cur = con.cursor()
00793         if att:
00794             result = cur.execute('SELECT due_date_{} FROM lab_schedule WHERE id=?'.format(int(att)), (lid_sem,))
00795         else:
00796             result = cur.execute('SELECT due_date_1, due_date_2, due_date_3, due_date_4 FROM lab_schedule WHERE id=?', (lid_sem,))
00797         return result.fetchone()
00798     return None
00799
00800
00801 def get_import_dates_by_labid(lid_sem, att=None, db_name='./grades.sqlite3'):
00802     with lite.connect(db_name) as con:
00803         cur = con.cursor()
00804         if att:
00805             result = cur.execute('SELECT imported_{} FROM lab_schedule WHERE id=?'.format(int(att)), (lid_sem,))
00806         else:
00807             result = cur.execute('SELECT imported_1, imported_2, imported_3, imported_4 FROM lab_schedule WHERE id=?', (lid_sem,))
00808         return result.fetchone()
00809     return None
00810
00811
00812 # save_grade_and_report(self.grade_ids[self.cur_idx], self.final_grade, self.user_comment, self.grader)
00813 def gen_report(lid_sem, att=None, db_name='./grades.sqlite3'):
00814     if not os.path.isfile(db_name):
00815         raise Exception("DB not found")
00816     with lite.connect(db_name) as con:
00817         cur = con.cursor()
```

```
00818         cur.execute("UPDATE lab_schedule SET imported_{}=strftime('%s','now') WHERE id=?".format(att), (lid_sem,))
00819         con.commit()
00820
00821
00822 def get_pipids_in_class_by_year_semester(year, semester, db_name='./grades.sqlite3'):
00823     if not os.path.isfile(db_name):
00824         raise Exception("DB not found")
00825     with lite.connect(db_name) as con:
00826         cur = con.cursor()
00827         result = cur.execute('SELECT pipeline_id FROM class WHERE year=? AND semester=?', (year, semester))
00828         all_ids = result.fetchall()
00829     return [elem[0] for elem in all_ids]
00830
00831
00832
00833 if __name__ == '__main__':
00834     settings_db_create()
```

8.7 generate.py File Reference

Namespaces

- generate

Functions

- def generate.convert_to_pdf (html_file, func_type)
- def generate.create_html_pdf_report2 (lab_dict)

  Creates nice html report for submitted labs and converts it to pdf format.
- def generate.create_html_pdf_zero_report (filename, stud_name, top_part, bot_part)
- def generate.create_not_submitted (stud_id, lab_type, lab_num, dir_name)
- def generate.generate_answers3 (lid, att, year, semester, db_name='./grades.sqlite3')
- def generate.time_to_str_with_tz (in_time)

8.8 generate.py

```
00001 import os
00002 import shutil
00003 from dateutil import tz
00004 from datetime import datetime
00005 from PyQt5.QtCore import QLocale
00006 from PyQt5.QtCore import QDateTime, Qt
00007 import sqlite3 as lite
00008 import multiprocessing as mp
00009 from db_init import get_pipids_in_class_by_year_semester, commit_gen_report, get_grades_by_lab_and_att, get_max_grade_for_lab
00010 QLocale.setDefault(QLocale(QLocale.English))
00011
00012
00013 def convert_to_pdf(html_file, func_type):
00014     """
00015     Provides different ways to generate pdf report.
00016     :param html_file: report in html format.
00017     :param func_type: selects the function used to generate pdf.
00018     :return: nothing, pdf is generated instead.
00019     """
00020     if func_type == "wkhtmltopdf":  # old way
00021         from subprocess import call
00022         call(["wkhtmltopdf", "-q", html_file, html_file[:-4] + 'pdf'])
00023     elif func_type == "pdfkit":  # best margins
00024         import pdfkit
00025         options = {
00026             'page-size': 'A4',
00027             'margin-top': '0.0in',
00028             'margin-right': '0.0in',
00029             'margin-bottom': '0.0in',
00030             'margin-left': '0.0in',
00031         }
00032         pdfkit.from_url(html_file, html_file[:-4] + 'pdf', options=options)
00033     elif func_type == 'weasyprint':  # potentially the fastest
00034         # if string is passed as param, but has margins problem
00035         from weasyprint import HTML
00036         with open(html_file, 'r') as html_in_file:
00037             cont = html_in_file.readlines()
00038         str_file = ''.join(cont)
00039         pdf = HTML(string=str_file)
00040         pdf.write_pdf(html_file[:-4] + 'pdf')
00041
00042
00043 # def create_html_pdf_report(joined_path, stud_name, cur_dir, grade, max_points, penalty,
```

```
00044 #                           final_score, top_part, bot_part, generated_time):
00045 #     """
00046 #     Creates nice html report for submitted labs and converts it to pdf format.
00047 #     TODO: use latex instead of ugly html.
00048 #     :param joined_path: working directory
00049 #     :param stud_name: full student name(first, last)
00050 #     :param cur_dir: directory with all labs(usually same as joined_path)
00051 #     :param grade: what grade to assign.
00052 #     :param max_points: max possible grade for this lab.
00053 #     :param penalty: usually for resubmission, like 90%, 70%...
00054 #     :param final_score: final grade = grade * penalty
00055 #     :param top_part: predefined top part of html document
00056 #     :param bot_part: predefined bottom part of html document
00057 #     :param generated_time: some extra statistics for curious students.
00058 #     :return: nothing, pdf is generated instead.
00059 #     """
00060 #     with open(joined_path + '-returned.html', 'w') as stud_report:
00061 #         stud_report.writelines(top_part)
00062 #         stud_report.write('<p>Grading directory : ' + cur_dir + ' </br>')
00063 #
00064 #         with open(joined_path + '/tech_info.txt', 'r') as tech_file:
00065 #             stud_report.writelines(tech_file.readlines())
00066 #
00067 #         stud_report.write('</p><p><i>Dear ' + stud_name + ', ')
00068 #
00069 #         with open(joined_path + '/responce.txt', 'r') as resp_file:
00070 #             stud_report.writelines(resp_file.readlines())
00071 #
00072 #         stud_report.write("</i></p>\n"
00073 #                           "<p>According to the comment above, next grade was assigned: "
00074 #                           "%d of %d <br/>\n \
00075 #                           Your final grade is %d*%.1f=<b>%d</b> of %d <br/>\n"
00076 #                           % (grade, max_points, grade, penalty, final_score, max_points))
00077 #         stud_report.write('This report was generated {} </p>'.format(generated_time))
00078 #         # TODO add current date/time
00079 #         stud_report.writelines(bot_part)
00080 #
00081 #     convert_to_pdf(joined_path + '-returned.html', "pdfkit")
00082 #     os.remove(joined_path + '-returned.html')
00083 #
00084
00085 def create_html_pdf_report2(lab_dict):
00086     """
00087     Creates nice html report for submitted labs and converts it to pdf format.
00088
00089     :return: nothing, pdf is generated instead.
00090     """
00091     with open('./answer.top', 'r') as partial_html:
00092         top_part = partial_html.readlines()
00093
00094     with open('./answer.bottom', 'r') as partial_html:
00095         bot_part = partial_html.readlines()
00096
00097     with open(lab_dict['lab_path'] + '-returned.html', 'w') as stud_report:
00098         stud_report.writelines(top_part)
00099
00100         stud_report.write('<p>Grading directory : {} </br>'.format(lab_dict['lab_path'].split('/')[-1]))
00101         stud_report.write('Due date was {} <br/>'.format(time_to_str_with_tz(lab_dict['due_date'])))
00102         stud_report.write('File was submited at {} <br/>'.format(time_to_str_with_tz(lab_dict['submitted'])))
00103         stud_report.write('I imported your file at {} <br/>'.format(time_to_str_with_tz(lab_dict['import_date'])))
00104         if lab_dict['graded'] is not None:
00105             stud_report.write('I graded your lab at {} <br/>'.format(time_to_str_with_tz(lab_dict['graded'])))
00106         else:
00107             stud_report.write('I did not grad your lab or grade timestamp was not set.<br/>')
00108         stud_report.write('Lab type : \'{}\' and it\'s number is \'{}\' <br/>'.format(lab_dict['type'], lab_dict['num']))
00109         stud_report.write('</p><p><i>Dear {} {}, '.format(lab_dict['first_name'], lab_dict['second_name']))
00110         if lab_dict['grader_comment'] is None or len(lab_dict['grader_comment']) < 2:
00111             stud_report.write('There were no comments.')
00112         else:
00113             stud_report.write(lab_dict['grader_comment'])
00114         if lab_dict['extra_comment'] is not None and len(lab_dict['extra_comment']) > 0:
00115             stud_report.write('<br/>\nExtra comment: {}'.format(lab_dict['extra_comment']))
00116
00117         stud_report.write("</i></p>\n"
00118                           "<p>According to the comment above, next grade was assigned: {} of {} <br/>\n"
00119                           " Your final grade is computed as {}*{:.1f}=<b>{}</b> of {} <br/>\n"
00120                           "".format(lab_dict['final_grade'], lab_dict['max_grade'], lab_dict['grade'], lab_dict['percent']/100,
      lab_dict['final_grade'], lab_dict['max_grade']))
00121         if lab_dict['grade'] == 0:
00122             stud_report.write('<br/>Don\'t forget to resubmit it by {} <br/><br/>\n'.format(time_to_str_with_tz(lab_dict['due_date'] +
      604800)))  # one extra week
```

```
00123         stud_report.write('This report was generated {} </p>\n'.format(QDateTime.currentDateTime().toString(Qt.DefaultLocaleLongDate)))
00124
00125         stud_report.writelines(bot_part)
00126
00127     convert_to_pdf(lab_dict['lab_path'] + '-returned.html', "pdfkit")
00128     os.remove(lab_dict['lab_path'] + '-returned.html')
00129
00130
00131 def create_html_pdf_zero_report(filename, stud_name, top_part, bot_part):
00132     """
00133     Creates nice html report for nonsubmitted labs and converts it to pdf format.
00134     :param filename: filename with correct naming(zeroes instead of timestamp)
00135     :param stud_name: full student name(first, last)
00136     :param top_part: predefined top part of html document
00137     :param bot_part: predefined bottom part of html document
00138     :return:
00139     """
00140     with open(filename, 'w') as zeroes_file:
00141         zeroes_file.writelines(top_part)
00142         zeroes_file.write(stud_name + ' : You did not submit your lab. :(</p>\n')
00143         zeroes_file.write("<p>According to comments above, next grade was assigned : 0 </p>")
00144         zeroes_file.write("<p>Please submit your file before the next due date.")
00145         zeroes_file.writelines(bot_part)
00146     convert_to_pdf(filename, "pdfkit")
00147     os.remove(filename)
00148
00149
00150 # def generate_answers(resubmit_num, dir_name, lab_type, lab_num, year, semester, grader_name):
00151 #     """
00152 #     general function that figures out max points, filenames, etc
00153 #     and calls generate function with appropriate parameters
00154 #     :param resubmit_num: resubmission attempt
00155 #     :param dir_name: working dir
00156 #     :param lab_type: open or closed lab
00157 #     :param lab_num: just lab identifier
00158 #     :param year: used wit semester to identify correct class list
00159 #     :param semester: used wit year to identify correct class list
00160 #     :param grader_name: name that will be displayed in the report
00161 #     :return:
00162 #     """
00163 #     students = {}
00164 #     # select
00165 #
00166 #     ids = get_pipids_in_class_by_year_semester(year, semester, 'grades.sqlite3')
00167 #     with lite.connect('grades.sqlite3') as con:
00168 #         cur = con.cursor()
00169 #
00170 #         for sid in ids.keys():
00171 #             result = cur.execute('SELECT first_name, second_name FROM students WHERE pipeline_id=?', (str(sid),))
00172 #             students[sid] = " ".join(result.fetchall()[0])
00173 #
00174 #     if not students:
00175 #         with open('students_list1.txt', 'r') as stud_list_file:
00176 #             temp_arr = stud_list_file.readlines()
00177 #             for line in temp_arr:
00178 #                 sid, name = line.split('%')
00179 #                 students[sid.strip()] = name.strip()
00180 #         del temp_arr
00181 #
00182 #
00183 #     if lab_type == 'Closed':
00184 #         max_points = 10
00185 #         type_for_name = 'CL'
00186 #     elif lab_type == 'Open':
00187 #         max_points = 20
00188 #         type_for_name = 'OL'
00189 #     else:
00190 #         raise Exception('Unknown lab type')
00191 #
00192 #     if resubmit_num == 1:
00193 #         penalty = 1.0
00194 #     elif resubmit_num == 2:
00195 #         penalty = 0.9
00196 #     elif resubmit_num == 3:
00197 #         penalty = 0.7
00198 #     elif resubmit_num == 4:
00199 #         penalty = 0.5
00200 #     else:
00201 #         penalty = 0.0
00202 #
00203 #     generated_time = QDateTime.currentDateTime().toString(Qt.DefaultLocaleLongDate)
```

```
00204 #
00205 #     print('This is ', type_for_name, ' lab, so max points is ', max_points)
00206 #
00207 #     try:
00208 #         shutil.rmtree(dir_name + 'Answers', ignore_errors=True)
00209 #         os.remove(dir_name + "grades.csv")
00210 #         os.remove(dir_name + "grades_for_" + type_for_name + "lab_num.csv")
00211 #     except Exception as e:
00212 #         print('Exception during dir preparetion : ', e)
00213 #
00214 #     dirs = os.walk(dir_name).__next__()[1]
00215 #
00216 #     with open('./answer.top', 'r') as partial_html:
00217 #         top_part = partial_html.readlines()
00218 #
00219 #     with open('./answer.bottom', 'r') as partial_html:
00220 #         bot_part = partial_html.readlines()
00221 #
00222 #     grades = list()
00223 #     for cur_dir in dirs:
00224 #         student_id = cur_dir.split('-')[0]
00225 #         joined_path = os.path.join(dir_name, cur_dir)
00226 #         with open(joined_path + '/grade.txt', 'r') as grade_file:
00227 #             grade = grade_file.readlines()
00228 #
00229 #         grade = int(grade[0].strip())
00230 #         final_score = grade * penalty
00231 #         grades.append((student_id, final_score))
00232 #         create_html_pdf_report(joined_path, students[student_id], cur_dir, grade,
00233 #                                max_points, penalty, final_score, top_part, bot_part, generated_time)
00234 #
00235 #     submitted = [x.split('-')[0] for x in dirs]
00236 #
00237 #     zeroes = list()
00238 #     for student in students:
00239 #         if student not in submitted:
00240 #             grades.append((student, 0))
00241 #             zeroes.append(student)
00242 #
00243 #     if resubmit_num == 1:
00244 #         for student_id in zeroes:
00245 #             filename = '%s/%s-%s%d-0000000000-returned' % \
00246 #                        (dir_name, student_id, type_for_name, lab_num)
00247 #             create_html_pdf_zero_report(filename+'.html', students[student_id], top_part, bot_part)
00248 #
00249 #     with open(dir_name + '/' + 'grades.csv', 'w') as grades_file:
00250 #         for grade in sorted(grades):
00251 #             grades_file.write("%s, %f \n" % grade)
00252 #
00253 #     os.mkdir(dir_name + '/Answers')
00254 #     files = os.walk(dir_name).__next__()[2]
00255 #     for file in files:
00256 #         if file[-3:] == 'pdf':
00257 #             shutil.move(dir_name + '/' + file, dir_name + '/Answers/' + file)
00258 #
00259 #     print('Done')
00260 #
00261 #
00262 # def generate_answers2(resubmit_num, dir_name, lab_type, lab_num, year, semester, grader_name):
00263 #     """
00264 #     general function that figures out max points, filenames, etc
00265 #     and calls generate function with appropriate parameters
00266 #     :param resubmit_num: resubmission attempt
00267 #     :param dir_name: working dir
00268 #     :param lab_type: open or closed lab
00269 #     :param lab_num: just lab identifier
00270 #     :param year: used wit semester to identify correct class list
00271 #     :param semester: used wit year to identify correct class list
00272 #     :param grader_name: name that will be displayed in the report
00273 #     :return:
00274 #     """
00275 #     students = {}
00276 #     # select
00277 #     import sqlite3 as lite
00278 #     from db_init import get_ids_in_class_by_year_semester
00279 #     ids = get_ids_in_class_by_year_semester(year, semester, 'grades.sqlite3')
00280 #     with lite.connect('grades.sqlite3') as con:
00281 #         cur = con.cursor()
00282 #
00283 #         for sid in ids.keys():
00284 #             result = cur.execute('SELECT first_name, second_name FROM students WHERE pipeline_id=?', (str(sid),))
```

```
00285 #             students[sid] = " ".join(result.fetchall()[0])
00286 #
00287 #     if not students:
00288 #         with open('students_list1.txt', 'r') as stud_list_file:
00289 #             temp_arr = stud_list_file.readlines()
00290 #             for line in temp_arr:
00291 #                 sid, name = line.split('%')
00292 #                 students[sid.strip()] = name.strip()
00293 #         del temp_arr
00294 #
00295 #
00296 #     if lab_type == 'Closed':
00297 #         max_points = 10
00298 #         type_for_name = 'CL'
00299 #     elif lab_type == 'Open':
00300 #         max_points = 20
00301 #         type_for_name = 'OL'
00302 #     else:
00303 #         raise Exception('Unknown lab type')
00304 #
00305 #     if resubmit_num == 1:
00306 #         penalty = 1.0
00307 #     elif resubmit_num == 2:
00308 #         penalty = 0.9
00309 #     elif resubmit_num == 3:
00310 #         penalty = 0.7
00311 #     elif resubmit_num == 4:
00312 #         penalty = 0.5
00313 #     else:
00314 #         penalty = 0.0
00315 #
00316 #     generated_time = QDateTime.currentDateTime().toString(Qt.DefaultLocaleLongDate)
00317 #
00318 #     print('This is ', type_for_name, ' lab, so max points is ', max_points)
00319 #
00320 #     try:
00321 #         shutil.rmtree(dir_name + 'Answers', ignore_errors=True)
00322 #         os.remove(dir_name + "grades.csv")
00323 #         os.remove(dir_name + "grades_for_" + type_for_name + "lab_num.csv")
00324 #     except Exception as e:
00325 #         print('Exception during dir preparetion : ', e)
00326 #
00327 #     dirs = os.walk(dir_name).__next__()[1]
00328 #
00329 #     with open('./answer.top', 'r') as partial_html:
00330 #         top_part = partial_html.readlines()
00331 #
00332 #     with open('./answer.bottom', 'r') as partial_html:
00333 #         bot_part = partial_html.readlines()
00334 #
00335 #     grades = list()
00336 #     for cur_dir in dirs:
00337 #         student_id = cur_dir.split('-')[0]
00338 #         joined_path = os.path.join(dir_name, cur_dir)
00339 #         with open(joined_path + '/grade.txt', 'r') as grade_file:
00340 #             grade = grade_file.readlines()
00341 #
00342 #         grade = int(grade[0].strip())
00343 #         final_score = grade * penalty
00344 #         grades.append((student_id, final_score))
00345 #         create_html_pdf_report(joined_path, students[student_id], cur_dir, grade,
00346 #                                max_points, penalty, final_score, top_part, bot_part, generated_time)
00347 #
00348 #     submitted = [x.split('-')[0] for x in dirs]
00349 #
00350 #     zeroes = list()
00351 #     for student in students:
00352 #         if student not in submitted:
00353 #             grades.append((student, 0))
00354 #             zeroes.append(student)
00355 #
00356 #     if resubmit_num == 1:
00357 #         for student_id in zeroes:
00358 #             filename = '%s/%s-%s%d-0000000000-returned' % \
00359 #                        (dir_name, student_id, type_for_name, lab_num)
00360 #             create_html_pdf_zero_report(filename+'.html', students[student_id], top_part, bot_part)
00361 #
00362 #     with open(dir_name + '/' + 'grades.csv', 'w') as grades_file:
00363 #         for grade in sorted(grades):
00364 #             grades_file.write("%s, %f \n" % grade)
00365 #
```

```
00366 #     os.mkdir(dir_name + '/Answers')
00367 #     files = os.walk(dir_name).__next__()[2]
00368 #     for file in files:
00369 #         if file[-3:] == 'pdf':
00370 #             shutil.move(dir_name + '/' + file, dir_name + '/Answers/' + file)
00371 #
00372 #     print('Done')
00373
00374 #
00375 # def create_a_report(lab_dict):
00376 #
00377 #     create_html_pdf_report2( lab_dict)
00378
00379
00380 def create_not_submitted(stud_id, lab_type, lab_num, dir_name):
00381     with open('./answer.top', 'r') as partial_html:
00382         top_part = partial_html.readlines()
00383
00384     with open('./answer.bottom', 'r') as partial_html:
00385         bot_part = partial_html.readlines()
00386     filename = '%s/%s-%s%d-0000000000-returned' % \
00387                (dir_name, stud_id, lab_type, lab_num)
00388     create_html_pdf_zero_report(filename + '.html', stud_id, top_part, bot_part)
00389
00390
00391 def generate_answers3(lid, att, year, semester, db_name='./grades.sqlite3'):
00392     all_ids = get_pipids_in_class_by_year_semester(year, semester)
00393     info_tup, info_desc = get_grades_by_lab_and_att(lid, att)
00394     col_names = [elem[0] for elem in info_desc]
00395     main_list = list()
00396     for tup in info_tup:
00397         a = dict()
00398         for i, elem in enumerate(tup):
00399             a[col_names[i]] = elem
00400         main_list.append(a)
00401     graded_students = [elem['pipeline_id'] for elem in main_list]
00402     grades = [elem['final_grade'] for elem in main_list]
00403     grade_dict = dict(zip(graded_students, grades))
00404     lab_type = main_list[0]['type']
00405     lab_num = main_list[0]['num']
00406     dir_name = main_list[0]['lab_path']
00407     dir_name = dir_name[:dir_name.rfind('/')]
00408     correctd_lab_type = 'CL' if lab_type == 'Closed' else 'OL'
00409
00410     # for elem in main_list:
00411     #     create_a_report(elem)
00412     #
00413     # for elem in main_list:
00414     #     commit_gen_report(elem['grade_id'])
00415
00416     not_subm_ids = [stud_id for stud_id in all_ids if stud_id not in graded_students]
00417
00418     if len(main_list) + len(not_subm_ids) == 0:
00419         return
00420
00421     ans_dir = os.path.join(dir_name, 'Answers')
00422     if os.path.exists(ans_dir):
00423         shutil.rmtree(ans_dir, ignore_errors=True)
00424     gr_file = os.path.join(dir_name, 'grades.csv')
00425     if os.path.exists(gr_file):
00426         os.remove(gr_file)
00427     gr_long_file = os.path.join(dir_name, "grades_for_{}lab_num.csv".format(correctd_lab_type))
00428     if os.path.exists(gr_long_file):
00429         os.remove(gr_long_file)
00430     files_to_rem = (os.path.join(dir_name, file) for file in (el for el in os.walk(dir_name).__next__()[2] if el[-3:] in ['pdf', 'html']))
00431
00432     with mp.Pool() as pool:
00433         pool.map(os.remove, files_to_rem)
00434         r1 = pool.map_async(create_html_pdf_report2, main_list)
00435         r2 = pool.map_async(commit_gen_report, (elem['grade_id'] for elem in main_list))
00436         if att == 1:
00437             pool.starmap(create_not_submitted, ((stud_id, correctd_lab_type, lab_num, dir_name) for stud_id in not_subm_ids))
00438         r1.wait()
00439         r2.wait()
00440
00441     with open(os.path.join(dir_name, '{}_lab_{}_grades.csv'.format(lab_num, lab_type)), 'w') as grades_file:
00442         grades_file.write("{1} Lab {0}, {1} Lab {0}\n".format(lab_num, lab_type))
00443         for stud_id in all_ids:
00444             if stud_id not in not_subm_ids:
00445                 grades_file.write("{:s}, {:d}\n".format(stud_id, int(grade_dict[stud_id])))
00446             else:
```

```
00447                 grades_file.write("{:s}, {:d}\n".format(stud_id, 0))
00448
00449
00450     best_grade_list = get_max_grade_for_lab(lid, year, semester)
00451     with open(os.path.join(dir_name, '{}_lab_{}_grades_best_so_far.csv'.format(lab_num, lab_type)), 'w') as grades_file:
00452         grades_file.write("{1} Lab {0}, {1} Lab {0}\n".format(lab_num, lab_type))
00453         for stud_tup in best_grade_list:
00454             grades_file.write('{}, {}\n'.format(stud_tup[0], stud_tup[1]))
00455
00456     # for elem in main_list:
00457     #     create_html_pdf_report2(elem)
00458     # for elem in main_list:
00459     #     commit_gen_report(elem['grade_id'])
00460
00461     # if att == 1:  # we do not form report for second attempt since most people are happy with previous grade
00462         # for stud_id in not_subm_ids:
00463         #     create_not_submitted(stud_id, correctd_lab_type, lab_num, dir_name)
00464
00465     os.mkdir(os.path.join(dir_name, 'Answers'))
00466     files = os.walk(dir_name).__next__()[2]
00467     for file in files:
00468         if file[-3:] == 'pdf':
00469             shutil.move(os.path.join(dir_name, file), os.path.join(dir_name, 'Answers/{}'.format(file)))
00470
00471     print('Done')
00472
00473
00474 def time_to_str_with_tz(in_time):
00475     return datetime.utcfromtimestamp(in_time).replace(tzinfo=tz.tzutc()).astimezone(tz.tzlocal()).strftime('%m-%d-%Y %H:%M')
00476 # if __name__ == '__main__':
00477 #     # generate_answers(3, 'Open_Lab_3_3', 'Open', 3)
```

8.9 main.py File Reference

Classes

- class main.CircFile
- class main.CircFile.circ_type
- class main.CircFile.PinType
- class main.Grader
- class main.UiMainWindow1
- class main.Ui_Create_settings_dialog

  Creates window that provides user with convenient way of changing settings that are stored in sqlite3 db.

- class main.SimpleDialog

  Wrapper class for very simple Ok|Cancel dialog.

- class main.Ui_manage_labs1
- class main.Ui_Create_dates_dialog1

Namespaces

- main

Functions

- def main.read_settings (db_name='settings.sqlite3')
- def main.get_grading_period (lid, cur_only=False)

Variables

- string main.MAIN_FILE_NAME = ''
- string main.MAIN_FILE_NAME_OVERRIDE = ''
- string main.styleData
- main.app = QtWidgets.QApplication(sys.argv)
- main.MainWindow = QtWidgets.QMainWindow()
- main.ui = UiMainWindow1()

8.10 main.py

```
00001 #! /usr/bin/env python3
00002 # -*- coding: utf-8 -*-
00003
00004 from collections import Counter
00005 import xml.etree.ElementTree as Etree
00006
00007 import sys
```

```
00008 import os
00009 import subprocess
00010 # import signal
00011 from pathlib import Path
00012 import numpy as np
00013 import sqlite3 as lite
00014 import zipfile
00015 from dateutil import tz
00016 from datetime import datetime
00017 import datetime
00018 import difflib
00019 import math
00020
00021 from PyQt5 import QtCore, QtWidgets, QtGui
00022 from PyQt5.QtCore import QDateTime, QLocale, QTimeZone
00023 from PyQt5.QtWidgets import QFileDialog
00024
00025 from main_window import Ui_mainWindow
00026 # from dates_window import Ui_dates_window
00027 from create_dates_diag import Ui_Create_dates_dialog
00028 from settings import Ui_Settings
00029 from manage_labs import Ui_manage_labs
00030 from db_init import *
00031 from simple_dialog import Ui_Dialog
00032 from generate import *
00033 import xml.etree.ElementTree as ET
00034
00035
00036 QLocale.setDefault(QLocale(QLocale.English))
00037
00038 MAIN_FILE_NAME = ”  # filename is selected automatically as most common. Change it only if it does not work.
00039 MAIN_FILE_NAME_OVERRIDE = ”
00040
00041 styleData = """
00042 /* https://stackoverflow.com/questions/22332106/python-qtgui-qprogressbar-color */
00043 QProgressBar
00044 {
00045     border: 1px solid grey;
00046     border-radius: 5px;
00047     text-align: center;
00048     font-weight: bold;
00049 }
00050 QProgressBar::chunk
00051 {
00052     background-color: #d7801a;
00053     width: 2.15px;
00054     margin: 0.5px;
00055 }
00056 """
00057
00058
00059 def read_settings(db_name = 'settings.sqlite3' ):
00060     """
00061     Queries settings db and sets paths
00062     :return: path to logisim, path to current semester labs
00063     """
00064     import os.path
00065
00066     if os.path.exists(db_name):
00067         with lite.connect(db_name) as con:
00068             cur = con.cursor()
00069             try:
00070                 cur.execute('SELECT * FROM PATHS')
00071                 result = cur.fetchone()
00072                 for row in result:
00073                     print(row)
00074                 logisim_path = result[0][0]
00075                 working_dir = result[0][1]
00076                 # since import is not implemented - ignore import path: import_path = result[0][2]
00077                 return logisim_path, working_dir
00078             except Exception as e:
00079                 print('Was not able to get results from settings DB: ', e)
00080     return None
00081
00082
00083 class CircFile:
00084     # not used yet
00085     class circ_type:
00086         def __init__(self, name):
00087             self.name = name
00088             self.input_pins = list()
```

```
00089             self.output_pins = list()
00090
00091     class PinType:
00092         def __init__(self, name, iotype, facing=None):
00093             self.name = name
00094             self.type = iotype
00095             self.facing = facing
00096
00097     def __init__(self, filename):
00098         self.filename = filename
00099         self.subtract = 0
00100         self.final_grade = 10
00101         self.__all_circuits = list()
00102
00103     def __get_parsed_circuits(self):
00104         """
00105
00106         :return:
00107         """
00108         tree = Etree.parse(self.filename)
00109         # self.log_update('Successfully openned ' + self.filename)
00110         root = tree.getroot()
00111         arr = list()
00112         for child in root:
00113             # print(child.tag)
00114             if child.tag == 'circuit':
00115                 arr.append(child)
00116         self.__all_circuits = arr
00117
00118     def get_parsed_pins(self):
00119         """
00120
00121         :return:
00122         """
00123         self.__get_parsed_circuits()
00124         arr = self.__all_circuits
00125         all_pins = list()
00126         for elem in arr:
00127             pins = list()
00128             for child in elem.findall('comp'):
00129                 if child.get('name') == 'Pin':
00130                     pins.append(child)
00131                     # print(child.tag, child.attrib)
00132             all_pins.append(pins)
00133
00134         clean_data = list()
00135         if all_pins:
00136             for pins in all_pins:  # Although this looks like an error - it is not,
00137                 # there is only one iteration. This code will be extended later
00138                 # as I had in my older scripts to grade all PLDs.
00139                 clean_data = list()
00140                 for pin in pins:
00141                     name = '0'
00142                     io_type = '0'
00143                     facing = ''
00144                     for elem in list(pin):
00145                         if elem.get('name') in ['output', 'input', 'tristate']:
00146                             io_type = elem.get('name')
00147                         elif elem.get('name') == 'label':
00148                             name = elem.get('val')
00149                         elif elem.get('name') == 'facing':
00150                             facing = elem.get('val')
00151                     clean_data.append(self.PinType(name, io_type, facing))
00152         else:
00153             raise Exception('Error in pin parsing(all_pins)')
00154
00155         output_pins = list()
00156         input_pins = list()
00157         other_pins = list()
00158
00159         if clean_data:
00160             for pin in clean_data:
00161                 if pin.type == 'output':
00162                     output_pins.append(pin)
00163                 elif pin.type == 'input' or pin.type == 'tristate':
00164                     input_pins.append(pin)
00165                 else:
00166                     other_pins.append(pin)
00167         else:
00168             raise Exception('Error in pin parsing(clean data)')
00169
```

```
00170         return input_pins, output_pins, other_pins
00171
00172
00173     def get_parsed_pins2(self, what_to_grade):
00174
00175         tree = ET.parse(self.filename)
00176         root = tree.getroot()
00177         arr=list()
00178         for child in root:
00179             # print(child.tag)
00180             if child.tag == 'circuit':
00181                 arr.append(child)
00182             # if child.attrib["name"] == what_to_grade:
00183             #     a = child
00184             #     b =1
00185
00186         all_circs = list()
00187         good_arr = list()
00188         for node in arr:
00189             if node.get('name').upper() in what_to_grade:
00190                 good_arr.append(node)
00191                 circ_instance = self.circ_type(node.get('name'))
00192                 all_circs.append(circ_instance)
00193                 # print(list(node)[0].items()[0][1])
00194
00195         all_pins = list()
00196         for elem in good_arr:
00197             pins = list()
00198             for child in elem.findall('comp'):
00199                 if child.get('name') == 'Pin':
00200                     pins.append(child)
00201                     # print(child.tag, child.attrib)
00202             all_pins.append(pins)
00203
00204
00205         clean_all_pins = list()
00206         for pins in all_pins:
00207             clean_data = list()
00208             for pin in pins:
00209                 name = '0'
00210                 type = '0'
00211                 for elem in list(pin):
00212                     if elem.get('name') in ['output', 'input', 'tristate']:
00213                         type = elem.get('name')
00214                     elif elem.get('name') == 'label':
00215                         name = elem.get('val')
00216                 clean_data.append(self.PinType(name, type))
00217             clean_all_pins.append(clean_data)
00218         for i in range(len(clean_all_pins)):
00219             for pin in clean_all_pins[i]:
00220                 if pin.type == 'output':
00221                     all_circs[i].output_pins.append(pin.name)
00222                 else:
00223                     all_circs[i].input_pins.append(pin.name)
00224         return all_circs
00225
00226
00227 class Grader:
00228     def __init__(self, working_directory, grader='Ivan'):
00229         self.__from_date = 0
00230         self.to_date = 0
00231         self.attempt = 0
00232         self.timestamps = list()
00233         self.stud_ids = list()
00234         self.stud_id = ''
00235         self.submitted = 0
00236         self.input_correct = False
00237         self.output_correct = False
00238         self.lab_max_grade = 0
00239         self.subtract = 0
00240         self.__wrong_clicked = False
00241         self.final_grade = 0
00242         self.__possible_answers_dict = {}
00243         self.global_log = ''
00244         self.previous_responses = ''
00245         self.__message_to_all = ''
00246         self.__graded_idlist = list()
00247         self.file_list = list()
00248         self.resp_text = 'I did not find any errors. Good job!\n'
00249         self.user_comment = ''
00250         self.cur_idx = 0
```

```
00251         self.working_dir = working_directory
00252         self.input_suggestion = set(",)
00253         self.resp_len = 38
00254         self.logisim_pid = -1
00255         self.circ_file_name = MAIN_FILE_NAME
00256         self.lab_type = ”
00257         self.lab_num = 0
00258         self.time = 0
00259         self.circ_obj_ref = None
00260         self.tot_elem = 0
00261         self.lab_id = ”
00262         self.grader = grader
00263 
00264     def open_dir(self):
00265         """
00266         Opens directory with labs for grading.
00267         :return: nothing.
00268         """
00269         # TODO check behaviour when directory is wrong.
00270         # if len(self.working_directory) < 3:
00271         #     wdir = './'
00272         # else:
00273         #     wdir = self.working_directory
00274 
00275 
00276         root, dirs, files = os.walk(self.working_dir).__next__()
00277         files.sort()
00278         # check_file = files[0]  # not used at this time
00279         # if len(files) < 1:
00280         #     raise Exception("No due files ? Extra files in working directory ?")
00281         # due_file = files[1] # TODO: change this to a better design. - Already changed
00282 
00283         self.lab_type = self.working_dir.split('/')[-2].split('_')[0]
00284         self.lab_num = int(self.working_dir.split('/')[-2].split('_')[2])
00285         self.attempt = int(self.working_dir.split('/')[-2].split('_')[3])
00286 
00287         if self.lab_type == 'Closed':
00288             self.lab_id = 'CLA{}'.format(self.lab_num)
00289             # self.lab_max_grade = 10
00290         else:  # Open
00291             # self.lab_max_grade = 20
00292             self.lab_id = 'OLA{}'.format(self.lab_num)
00293 
00294         self.lab_max_grade = get_lab_max_value(self.lab_id)
00295 
00296         # self.time = int(due_file[6:])
00297 
00298         # dirs.sort()  # sort list of submitted labs
00299         # if dirs[0] == 'Answers':
00300         #     dirs.pop(0)
00301 
00302         self.circ_file_name = get_lab_filename(self.lab_id)[0]
00303         self.year, self.semester = self.working_dir.split('/')[-3].split('_')
00304         self.lid = get_labid_in_schedule(get_lab_id(self.lab_type, self.lab_num), self.year, self.semester)
00305         self.timestamps, self.stud_ids, self.grade_ids, self.lab_paths = get_empty_grades_by_lid(self.lid, self.attempt)
00306 
00307         atime = get_grading_period(self.lid, cur_only=True)
00308         self.time_from = atime[1]
00309         self.time_to = atime[2]
00310         self.time_cur = atime[3]
00311 
00312         self.time_from_qt = QDateTime.fromSecsSinceEpoch(self.time_from)
00313         self.time_to_qt = QDateTime.fromSecsSinceEpoch(self.time_to)
00314         self.time_cur_qt = QDateTime.fromSecsSinceEpoch(self.time_cur)
00315 
00316         if self.lab_num > 8 and self.lab_type == 'Closed':
00317             if self.lab_num == 9:
00318                 self.what_to_grade = ['PC_BUS', 'AR_LD', 'PC_LD', 'PC_INC', 'DR_LD', 'DR_BUS']
00319             elif self.lab_num == 10:
00320                 self.what_to_grade = ["R_LD", "R_BUS", "S_LD", "ACC_CLR", "ACC_LD", "ACC_BUS", "ALU_SEL"]
00321             elif self.lab_num == 11:
00322                 self.what_to_grade = ["Z_LD", "OUTR_LD", "RAM_RW", "RAM_EN", "IR_LD", "SC_CLR"]
00323             circ = CircFile('/home/vanya/Documents/3130_labs/2018_2/PLDs.circ')
00324             self.all_my_circuits = circ.get_parsed_pins2(self.what_to_grade)
00325 
00326         if self.lab_paths is not None and len(self.lab_paths) > 0:
00327             self.timestamps, self.stud_ids, self.grade_ids, self.lab_paths = self.check_files()
00328 
00329         if self.lab_paths is None or len(self.lab_paths) == 0:  # if there are no ungraded labs - display all labs
00330             self.timestamps, self.stud_ids, self.grade_ids, self.lab_paths = get_all_grades_by_lid(self.lid, self.attempt)
00331 
```

```
00332         # self.grades = [self.lab_max_grade]*len(self.grade_ids)
00333         # self.stud_ids = dirs
00334         # self.stud_ids = list()
00335         # self.timestamps = list()
00336         # # directory_list = list()
00337         # for name in dirs:
00338         #     self.file_list.append(os.path.join(root, name))
00339         #     temp_arr = name.split('-')
00340         #     self.stud_ids.append(temp_arr[0])
00341         #     self.timestamps.append(int(temp_arr[2]))
00342 
00343         # for file in self.file_list:
00344         #     print(file)
00345 
00346     def check_files(self):
00347         paths_with_files_list = list()
00348         good_ids = list()
00349         good_sids = list()
00350         good_tss = list()
00351 
00352         for i, stud_path in enumerate(self.lab_paths):
00353             cur_path = os.path.join(stud_path, self.circ_file_name)
00354             if os.path.exists(cur_path):
00355                 paths_with_files_list.append(stud_path)
00356                 good_ids.append(self.grade_ids[i])
00357                 good_sids.append(self.stud_ids[i])
00358                 good_tss.append(self.timestamps[i])
00359                 if self.lab_num > 8 and self.lab_type == 'Closed':
00360                     self.precheck_PLDs(i)
00361             else:
00362                 if self.attempt > 1:
00363                     next_date = time_to_str_with_tz(self.time_to + self.time_to - self.time_from)
00364                 else:
00365                     next_date = time_to_str_with_tz(self.time_to + 604800)  # 604800 - one week in unix time, this line needs corrections for case when you skip a week
00366                 gen_filenotfound_resp(self.grade_ids[i], stud_path, self.circ_file_name, self.grader, self.attempt, next_date)
00367         # self.grade_ids = good_ids
00368         # self.stud_ids = good_sids
00369         # self.timestamps = good_tss
00370         return good_tss, good_sids, good_ids, paths_with_files_list
00371 
00372     def get_stud_circ_ind(self, student_circuits, circ_to_grade):
00373         for stud_circ in student_circuits:
00374             if stud_circ.name.upper() == circ_to_grade.upper():
00375                 return student_circuits.index(stud_circ)
00376         for stud_circ in student_circuits:
00377             print(stud_circ.name.upper())
00378         return -1
00379 
00380     def precheck_PLDs(self, stud_ind):
00381         file = os.path.join(self.lab_paths[stud_ind], self.circ_file_name)
00382 
00383         student_circuits = CircFile(file).get_parsed_pins2(self.what_to_grade)
00384         errors = 0
00385 
00386         out_str = '<br> Next part was generated by automatic grader that I wrote several years ago.' \
00387                   'If you are not agree with something or suspect an error - please send me a message.<br>With this grading approach you cat get nonzero grade ' \
00388                   'even if not everything was correct.<br>'
00389         for circ_to_grade in self.what_to_grade:
00390             for good_circ in self.all_my_circuits:
00391                 if good_circ.name.upper() == circ_to_grade.upper():
00392                     cur_ind = self.get_stud_circ_ind(student_circuits, circ_to_grade)
00393                     out_str += '<br>'
00394                     if cur_ind == -1:
00395                         out_str += '<font color="red">{}  NOT FOUND!<br> </font>'.format(circ_to_grade)
00396                         errors += 1
00397                     else:
00398                         check_pins = student_circuits[cur_ind].input_pins
00399                         for i in range(len(check_pins)):
00400                             if check_pins[i][0].lower() != 'c':
00401                                 # print(check_pins[i])
00402                                 if len(check_pins[i][1:]) > 0:
00403                                     try:
00404                                         pos = None
00405                                         for ch in check_pins[i]:
00406                                             if not ch.isalpha():
00407                                                 pos = check_pins[i].index(ch)
00408                                                 break
00409                                         num = int(check_pins[i][pos:])
00410                                     except Exception as e:
```

```
00411                                     print(e)
00412                                     continue
00413                                 check_pins[i] = check_pins[i][0:1] + str(num)
00414                         student_circuits_sorted = sorted(check_pins)
00415                         good_circ_sorted = sorted(good_circ.input_pins)
00416                         sm = difflib.SequenceMatcher(None, student_circuits_sorted, good_circ_sorted)
00417                         res_ratio = sm.ratio()
00418 
00419                         if res_ratio > 0.99:
00420                             out_str += '<font color="green"> {} : PERFECT MATCH!<br> </font>'.format(circ_to_grade)
00421                         elif res_ratio > 0.15:
00422                             out_str += '{} :Great news : you match ratio is {:.1%} (>75%)<br>{} : <b>FOUND</b> {}  <br>{} : <b>EXPECTED</b> {} <br>'\
00423                                 .format(circ_to_grade, res_ratio, circ_to_grade, ' '.join(student_circuits_sorted), circ_to_grade, ' '.join(good_circ_sorted))
00424                             errors += 1
00425                         else:
00426                             out_str += '<font color="red">{} Bad news : you match ratio is only {:.1f}% - this means that you have to ' \
00427                                        'significantly change your circuit. <br>  Please send me a message if you need some advice.<br> ' \
00428                                        '</font>'.format(circ_to_grade, res_ratio)
00429                             errors += 1
00430 
00431         final_grade = math.ceil(10 * (len(self.what_to_grade) - errors) / len(self.what_to_grade))
00432         # out_str += '<br> Bad grade confidence: ' + conf + ' (this is for Ivan)<br>' + '<br> Next part will be typed manually: <br>'
00433         save_grade_and_report(self.grade_ids[stud_ind], final_grade, out_str, None, self.grader)
00434         return final_grade, out_str
00435 
00436 
00437     def get_stud_id(self):
00438         """
00439         Just a simple getter.
00440         :return:
00441         """
00442         return self.stud_id
00443 
00444     def log_update(self, log_event):
00445         """
00446         Saves events into a string.
00447         Later this string is displayed in a separate tab.
00448         :param log_event: what happened.
00449         :return: nothing
00450         """
00451         self.global_log += str(self.stud_id) + ': ' + str(log_event) + '\n'
00452 
00453     def get_parsed_pins(self):
00454         """
00455         High level function that obtains in|out pins and check their facing.
00456         :return: nothing.
00457         """
00458         try:
00459             input_pins, output_pins, other_pins = self.circ_obj_ref.get_parsed_pins()
00460             if other_pins:
00461                 self.log_update('I was not able to recognize ' + str(len(other_pins)) + " pins.")
00462             self.input_correct = True
00463             self.output_correct = True
00464             if not self.check_pins_facing(pins=input_pins, corr_facing='east'):
00465                 self.subtract += 1
00466                 self.input_correct = False
00467             if not self.check_pins_facing(pins=output_pins, corr_facing='west'):
00468                 self.subtract += 1
00469                 self.output_correct = False
00470         except Exception as e:  # TODO check for FileNotFoundError and assign ZERO
00471             print(e)
00472             # self.log_update(sys.exc_info()[0])
00473             # print(sys.exc_info()[0])
00474             raise
00475         # self.log_update('Done checking: ' + self.filename)
00476 
00477 
00478     # noinspection PyMethodMayBeStatic
00479     def check_pins_facing(self, pins, corr_facing):
00480         """
00481         Low level pin facing checker.
00482         :param pins: structured list of pins.
00483         :param corr_facing: nothing to add.
00484         :return: True if facing is correct, False otherwise
00485         """
00486         for pin in pins:
00487             if pin.facing != corr_facing and pin.facing != '':
00488                 return False
00489         return True
```

```
00490
00491     def check_file(self):
00492         """
00493         Opens circ file, tryes to parse it and to generate grade according to the pin facing.
00494         This check is too simple and most likely will be updated later.
00495         :return: nothing
00496         """
00497         file = os.path.join(self.file_list[self.cur_idx], MAIN_FILE_NAME)
00498
00499         circ_obj = CircFile(file)
00500         self.circ_obj_ref = circ_obj
00501         self.subtract = 0
00502         try:
00503             self.get_parsed_pins()
00504
00505             self.log_update('Pins successfully parsed.')
00506             self.final_grade = self.lab_max_grade - self.subtract
00507             self.generate_response()
00508         except Exception as e:
00509             print(e)
00510             self.log_update(sys.exc_info()[0])
00511
00512     def check_circ_exist(self):
00513         """
00514         Checks whether file exists with specified name.
00515         If not generates report which contains all submitted elements.
00516         :return: True is file exists, False otherwise
00517         """
00518         if not os.path.isfile(self.file_list[self.cur_idx] + '/' + self.circ_file_name):
00519             self.resp_text = 'File was not found'
00520             file_found = os.listdir(self.file_list[self.cur_idx])
00521             potential_files = list()
00522             for file in file_found:
00523                 if file not in ['grade.txt', 'penalty.txt', 'responce.txt', 'tech_info.txt', ]:
00524                     potential_files.append(file)
00525             if potential_files:
00526                 self.resp_text += '\nNext files|folders were found:\n'
00527             for file in potential_files:
00528                 if os.path.isdir(self.file_list[self.cur_idx] + '/' + file):
00529                     self.resp_text += file + ' - directory.\n'
00530                 else:
00531                     self.resp_text += file + ' - regular file.\n'
00532             self.resp_len = len(self.resp_text)
00533             self.final_grade = 0
00534             return False
00535         return True
00536
00537     def read_resp(self):
00538         """
00539         Reads response generated by either import scripts or by grader.
00540         Usually is stored in response.txt. Later may be transferred into DB.
00541         :return: nothing.
00542         """
00543         self.submitted = self.timestamps[self.cur_idx]
00544         try:
00545             with open(os.path.join(self.file_list[self.cur_idx], 'responce.txt'), 'r') as resp_file:
00546                 a = resp_file.readlines()
00547                 self.resp_text = ''.join(a)
00548                 self.resp_len = len(self.resp_text)
00549         except Exception as e:
00550             print(e)
00551             self.log_update(sys.exc_info()[0])
00552
00553         try:
00554             with open(os.path.join(self.file_list[self.cur_idx], 'grade.txt'), 'r') as grade_file:
00555                 self.final_grade = int(grade_file.readline())
00556         except Exception as e:
00557             print(e)
00558             self.log_update(sys.exc_info()[0])
00559
00560         # self.read_prev_resp()
00561
00562     def read_resp2(self):
00563         self.final_grade, self.resp_text, self.user_comment, graded = get_resp_and_grade(self.grade_ids[self.cur_idx])
00564         if graded is None:
00565             self.final_grade = self.lab_max_grade
00566             self.resp_text = 'I did not find any errors. Good job!'
00567         # self.resp_text = '' if self.resp_text is None else self.resp_text
00568         self.resp_len = len(self.resp_text)
00569         return graded
00570
```

```
00571     def read_prev_resp2(self):
00572         self.previous_responses = get_prev_resp(self.grade_ids[self.cur_idx], self.stud_ids[self.cur_idx], self.lid)
00573 
00574     def read_prev_resp(self):
00575         """
00576         In case we are working with resubmission,
00577         this function will try to get previous responses.
00578         :return: nothing.
00579         """
00580         if self.attempt > 1:
00581             self.previous_responses = ''  # TODO find same name in folder name
00582             prev_att = int(self.working_dir[-2:-1])
00583             for i in range(prev_att-1, 0, -1):
00584                 prev_working_dir = self.working_dir[:-2] + str(i) + '/'
00585                 for file in os.listdir(prev_working_dir):
00586                     if file.__contains__(self.stud_id):
00587                         # print(file)
00588                         try:
00589                             with open(prev_working_dir + file + '/responce.txt', 'r') as resp_file:
00590                                 self.previous_responses += str(i) + 'th submission :\n\t' \
00591                                                            + '\n'.join(resp_file.readlines())
00592                         except Exception as e:
00593                             print('Error in read prev responce: ', e)
00594 
00595     def next_circ(self):
00596         """
00597         Opens next circuit
00598         :return: current index
00599         """
00600         self.cur_idx += 1
00601         # self.check_file(self.cur_idx)
00602         self.user_comment = ''
00603         graded = self.read_resp2()
00604         # if graded:
00605         self.read_prev_resp2()
00606         # if self.check_circ_exist():
00607         #     self.read_resp()
00608         self.stud_id = self.stud_ids[self.cur_idx]
00609         # try:
00610         #     self.read_prev_resp()
00611         # except Exception as e:
00612         #     print('Error during attempt to read prev resp when opening next circuit: ', e)
00613         #     # TODO add handler
00614         return self.cur_idx
00615 
00616     def prev_circ(self):
00617         """
00618         Opens previous circuit
00619         :return: current index
00620         """
00621         self.cur_idx -= 1
00622         # self.check_file(self.cur_idx)
00623         self.user_comment = ''
00624         graded = self.read_resp2()
00625         if graded:
00626             self.read_prev_resp2()
00627         # if self.check_circ_exist():
00628         #     self.read_resp()
00629         self.stud_id = self.stud_ids[self.cur_idx]
00630         # try:
00631         #     self.read_prev_resp()
00632         # except Exception as e:
00633         #     print('Error during attempt to read prev resp when opening prev circuit: ', e)
00634         #     # TODO add handler
00635         return self.cur_idx
00636 
00637     def check_wrong(self):
00638         """
00639         Funciton bound to 'Wrong' button(checkbox). Marks lab as 'wrong'.
00640         :return: nothing
00641         """
00642         self.final_grade = 0
00643         self.resp_text = 'your lab was marked as wrong. You should fix errors listed below and resubmit it.'
00644         self.resp_len = len(self.resp_text)
00645 
00646     def save_grade(self):
00647         """
00648         Function bound to 'Save grade' button. Saves grade into 'grade.txt' file
00649         :return: nothing.
00650         """
00651         file = os.path.join(self.lab_paths[self.cur_idx], 'grade.txt')
```

```
00652         with open(file, 'w') as grade_file:
00653             grade_file.write(str(self.final_grade))
00654 
00655         self.log_update('Grade saved')
00656 
00657     def save_responce(self):
00658         """
00659         Function bound to 'Save responce' button.
00660         Saves current (auto and manual) responce into 'responce.txt'.
00661         :return: nothing.
00662         """
00663         file = os.path.join(self.lab_paths[self.cur_idx], 'responce.txt')
00664         with open(file, 'w') as resp_file:
00665             resp_file.write(self.resp_text)
00666             if self.user_comment:
00667                 resp_file.write('\nAdditional comment: ' + self.user_comment + '\n')
00668         self.log_update('Responce saved')
00669 
00670     def save_all(self):
00671         """
00672         Function bound to 'Save all' button.
00673         Saves both grade and response by calling appropriate functions.
00674         :return: nothing.
00675         """
00676         self.save_grade()
00677         self.save_responce()
00678 
00679 
00680     def save_all2(self):
00681         """
00682         Same as save_all but uses db to save grade
00683         :return:
00684         """
00685         save_grade_and_report(self.grade_ids[self.cur_idx], self.final_grade, self.resp_text, self.user_comment, self.grader)
00686 
00687     def generate_response(self):
00688         """
00689         Regenerates the responce.
00690         :return: nothing.
00691         """
00692         self.resp_text = ''
00693         self.user_comment = ''
00694         if self.input_correct and self.output_correct:
00695             self.resp_text = 'I did not find any errors. Good job!'
00696         else:
00697             if not self.input_correct:
00698                 self.resp_text += 'Your input pins have wrong orientation.\n'
00699 
00700             if not self.output_correct:
00701                 self.resp_text += 'Your output pins have wrong orientation.\n'
00702         self.resp_len = len(self.resp_text)
00703 
00704     def add_to_common_answers(self, typed):
00705         """
00706         Function bound to FocusOut input handler.
00707         Adds whatever is typed into popular answers.
00708         :param typed: Text from input field
00709         :return: nothing.
00710         """
00711         self.input_suggestion.add(typed)
00712 
00713 
00714 class UiMainWindow1(Ui_mainWindow):
00715     """
00716 
00717     """
00718 
00719     def __init__(self):
00720         Ui_mainWindow.__init__(self)
00721         self.grader_ref = None
00722         self.cal_window = None
00723         self.working_dir = None
00724 
00725 
00726 
00727     def disable_fields(self):
00728         """
00729         disables UI elements. Usually followed by 'enable_fields'
00730         :return: nothing
00731         """
00732         self.checkB_input_pin_status.setDisabled(True)
```

```
00733         self.checkB_output_pin_status.setDisabled(True)
00734         # self.input_response_browser.setDisabled(True)
00735         self.checkB_wrong.setDisabled(True)
00736 
00737         # self.input_subtract.setDisabled(True)
00738         self.but_regrade.setDisabled(True)
00739         self.popular_answers.setDisabled(True)
00740         self.input_final_grade.setDisabled(True)
00741         self.checkB_wrong.setChecked(False)
00742         self.check_autosave.setDisabled(True)
00743         self.input_current_id.setText('')
00744 
00745     def enable_fields(self):
00746         """
00747         enables UI elements. Usually follows 'disable_fields'
00748         :return: nothing
00749         """
00750         self.checkB_input_pin_status.setEnabled(True)
00751         self.checkB_output_pin_status.setEnabled(True)
00752         # self.input_response_browser.setEnabled(True)
00753         self.checkB_wrong.setEnabled(True)
00754         self.input_final_grade.setEnabled(True)
00755         self.check_autosave.setEnabled(True)
00756 
00757         # self.input_subtract.setEnabled(True)
00758         # self.but_regrade.setEnabled(True)
00759         self.popular_answers.setEnabled(True)
00760 
00761     def load_dir(self):
00762         """
00763         Resets UI when directory to grade is loaded.
00764         :return:
00765         """
00766         # activate elements
00767         cur_year, cur_sem = self.grader_ref.working_dir.split('/')[-3].split('_')
00768         self.class_id_to_id = get_ids_in_class_by_year_semester(cur_year, cur_sem)[1]
00769         self.but_begin.setDisabled(True)
00770         self.but_begin.repaint()
00771         self.progressBar.setEnabled(True)
00772 
00773         self.disable_fields()
00774 
00775         self.grader_ref.tot_elem = len(self.grader_ref.lab_paths)
00776         if self.grader_ref.tot_elem > 1:
00777             self.but_next.setEnabled(True)
00778 
00779         self.progressBar.setMaximum(self.grader_ref.tot_elem)
00780         self.progressBar.setValue(0)
00781         self.popular_answers.clear()
00782 
00783         # self.grader_ref.check_file(0)
00784         # self.grader_ref.stud_id = self.grader_ref.stud_ids[self.grader_ref.cur_idx]
00785         self.grader_ref.cur_idx = -1
00786         # graded = self.grader_ref.read_resp2()
00787         # if graded:
00788         #     self.grader_ref.read_prev_resp2()
00789         self.next_circ()
00790         # self.grader_ref.read_resp()
00791         # self.grader_ref.read_prev_resp()
00792         # self.show_stat()
00793         # self.check_file()
00794         # self.input_current_id.setPlainText(self.grader_ref.get_stud_id())
00795 
00796         self.enable_fields()
00797         self.input_response_browser_user.setEnabled(True)
00798         self.but_regrade.setText('GRADE')
00799         self.but_save_all.setEnabled(True)
00800         self.but_save_response.setEnabled(True)
00801         self.check_autosave.setEnabled(True)
00802         self.but_reset.setEnabled(True)
00803 
00804     def my_open_file(self):
00805         """
00806         Creates Grader instance and stores it in local reference
00807         Determines filename by selecting filename used by majority of students.
00808         Displays selected filename in UI element, so grader can see it.
00809         :return:
00810         """
00811         working_dir = self.input_file_location.text()
00812         # self.input_response_browser.clear()
00813         # self.input_response_browser_user.clear()
```

```
00814         self.input_response_browser.setPlainText('I did not find any errors. Good job!')
00815         grader_name = settings_db_read_settings()[1][0]
00816         self.current_tz = QDateTime.currentDateTime().timeZoneAbbreviation()
00817 
00818         try:
00819             my_grader = Grader(working_dir, grader_name)
00820             my_grader.open_dir()
00821 
00822             self.grader_ref = my_grader
00823 
00824             self.input_max_pos_grade.setText(str(my_grader.lab_max_grade))
00825             self.input_attempt.setText(str(my_grader.attempt))
00826             self.dateTimeEdit_from.setDateTime(my_grader.time_from_qt)
00827             self.dateTimeEdit_to.setDateTime(my_grader.time_to_qt)
00828             self.grader_ref.add_to_common_answers("")  # helps to remove all text in user comment section
00829             # QDateTime.currentDateTime().timeZone()
00830             # global MAIN_FILE_NAME, MAIN_FILE_NAME_OVERRIDE
00831 
00832             # MAIN_FILE_NAME = get_lab_filename(my_grader.lab_id)[0]
00833             # if not MAIN_FILE_NAME:
00834             #     # Old way, I was determining filename as the most common submitted file.
00835             #     if not MAIN_FILE_NAME_OVERRIDE:
00836             #         a = []
00837             #         for root, dirs, files in os.walk(working_dir):
00838             #             for file in files:
00839             #                 if file.endswith(".circ"):
00840             #                     a.append(file)
00841             #         a = np.array(a)
00842             #
00843             #         MAIN_FILE_NAME = Counter(a.flat).most_common(1)[0][0]
00844             #     else:
00845             #         MAIN_FILE_NAME = MAIN_FILE_NAME_OVERRIDE
00846             #     # Now I can just read it from DB
00847 
00848             # self.grader_ref.circ_file_name = MAIN_FILE_NAME
00849             self.filename_lineEdit.setText(self.grader_ref.circ_file_name.split('.')[0])
00850             # self.reset_grade_resp()
00851             self.but_save_all.setChecked(False)
00852 
00853             self.but_create_report.setEnabled(True)
00854             self.but_begin.setEnabled(True)
00855 
00856         except Exception as e:  # TODO add log error
00857             print('Error in open_file : ', e)
00858             print(sys.exc_info()[0])
00859 
00860     def show_stat(self):
00861         """
00862         Displays current and old responses.
00863         Resets many UI elements for a next student.
00864         :return:
00865         """
00866         self.input_prev_response.setPlainText(self.grader_ref.previous_responses)
00867         file_path = os.path.join(self.grader_ref.lab_paths[self.grader_ref.cur_idx], self.grader_ref.circ_file_name)
00868         if not Path(file_path).is_file():
00869             self.kill_logisim()
00870             self.grader_ref.final_grade = 0
00871             self.input_response_browser.setPlainText('File does not exist.')
00872             self.grader_ref.final_grade = 0
00873         else:
00874             if self.but_regrade.text() == '&GRADE' or self.but_regrade.text() == 'GRADE':
00875                 try:
00876                     self.run_logisim(file_path)
00877                 except Exception as e:
00878                     print('Error in run_logisim: ', e)
00879                     print(sys.exc_info()[0])
00880 
00881         self.input_current_id.setText(self.class_id_to_id[self.grader_ref.get_stud_id()])
00882         self.dateTimeEdit_submitted.setDateTime(QDateTime.fromSecsSinceEpoch(self.grader_ref.timestamps[self.grader_ref.cur_idx]))
00883         self.input_subtract.setText("")
00884         self.input_final_grade.setText(str(self.grader_ref.final_grade))
00885         self.input_log_browser.setText(self.grader_ref.global_log)
00886         self.input_response_browser.setPlainText(self.grader_ref.resp_text)
00887         self.input_response_browser_user.setPlainText(self.grader_ref.user_comment)
00888         self.checkB_input_pin_status.setChecked(False)
00889         self.checkB_output_pin_status.setChecked(False)
00890         self.popular_answers.setCurrentIndex(-1)
00891 
00892     def check_file(self):
00893         """
00894         Sets UI elements related to autonomous pin check to states
```

```
00895         set by lab checker.
00896         Not useful anymore.
00897         :return: nothing.
00898         """
00899         self.input_subtract.setText(str(self.grader_ref.subtract))
00900         self.input_final_grade.setText(str(self.grader_ref.final_grade))
00901 
00902         self.input_log_browser.setText(self.grader_ref.global_log)
00903         # self.input_log_browser.append(self.grader_ref.global_log)
00904 
00905         if self.grader_ref.input_correct:
00906             self.checkB_input_pin_status.setChecked(True)
00907         if self.grader_ref.output_correct:
00908             self.checkB_output_pin_status.setChecked(True)
00909 
00910         # self.but_save_response.setDisabled(True)
00911         # self.but_save_all.setDisabled(True)
00912 
00913         # self.but_edit_done.setDisabled(True)
00914         try:
00915             # self.grader_ref.generate_response()  #TODO this overwrites File not found.
00916             self.input_response_browser.setPlainText(self.grader_ref.resp_text)
00917             # self.but_edit_done.setEnabled(True)
00918             # self.but_save_response.setEnabled(True)
00919             # self.but_save_all.setEnabled(True)
00920         except Exception as e:
00921             print('Error in generate response:', e)
00922 
00923     def next_circ(self):
00924         """
00925         Function bound to 'Next' button.
00926         Saves prev response and grade if 'autosave' is on.
00927         Opens next file
00928         Checks whether next file exists and sets UI appropriately.
00929         :return:
00930         """
00931         self.disable_fields()
00932         self.but_regrade.setText('GRADE')
00933         if self.check_autosave.isChecked() and self.grader_ref.cur_idx >= 0:
00934             self.save_all()
00935         # else:
00936         #     self.check_autosave.setDisabled(True)
00937         next_idx = self.grader_ref.next_circ()
00938         # self.check_file()
00939         self.show_stat()
00940         if next_idx >= self.grader_ref.tot_elem-1:
00941             self.but_next.setDisabled(True)
00942         if next_idx == 1:
00943             self.but_prev.setEnabled(True)
00944 
00945         self.progressBar.setValue(next_idx)
00946         self.enable_fields()
00947 
00948     def prev_circ(self):
00949         """
00950         Function bound to 'Prev' button.
00951         Saves prev response and grade if 'autosave' is on.
00952         Opens prev file
00953         Checks whether prev file exists and sets UI appropriately.
00954         :return:
00955         """
00956         self.disable_fields()
00957         self.but_regrade.setText('GRADE')
00958         next_idx = self.grader_ref.prev_circ()
00959         # self.check_file()
00960         self.show_stat()
00961         if next_idx <= self.grader_ref.tot_elem-1:
00962             self.but_next.setEnabled(True)
00963         if next_idx == 0:
00964             self.but_prev.setDisabled(True)
00965 
00966         self.progressBar.setValue(next_idx)
00967         self.enable_fields()
00968 
00969     def check_wrong(self):
00970         """
00971         Function bound to 'WRONG' button.
00972         Marks lab as wrong.
00973         :return: nothing
00974         """
00975         if self.checkB_wrong.isEnabled():
```

```
00976             self.grader_ref.check_wrong()
00977             self.input_final_grade.setText(str(self.grader_ref.final_grade))
00978             self.grader_ref.log_update('Lab was marked as wrong manually. Zero was assigned to final grade.')
00979             self.input_response_browser.setPlainText(self.grader_ref.resp_text)
00980             self.checkB_wrong.setDisabled(True)
00981
00982     def regrade(self):
00983         """
00984         Resets lab values.
00985         :return:
00986         """
00987         self.disable_fields()
00988         self.but_regrade.setText('regrade')
00989         # if self.lab_num > 8 and self.lab_type == 'Closed':
00990         #    self.precheck_PLDs(i, cur_path)
00991         self.show_stat()
00992         # self.grader_ref.check_file()
00993         # if self.grader_ref.check_circ_exist():
00994         #     self.check_file()
00995         self.input_response_browser.setPlainText(self.grader_ref.resp_text)
00996         self.enable_fields()
00997
00998     def reset_grade_resp(self):
00999         """
01000         Resets grade values.
01001         :return:
01002         """
01003         self.disable_fields()
01004         self.show_stat()
01005         # self.grader_ref.check_file()
01006         # if self.grader_ref.check_circ_exist():
01007         if self.grader_ref.lab_num > 8 and self.grader_ref.lab_type == 'Closed':
01008            self.grader_ref.final_grade, report = self.grader_ref.precheck_PLDs(self.grader_ref.cur_idx)
01009            self.input_response_browser.setPlainText(report)
01010         else:
01011             self.grader_ref.final_grade = self.grader_ref.lab_max_grade
01012             self.input_response_browser.setPlainText('I did not find any errors. Good job!')
01013
01014         self.input_final_grade.setText(str(self.grader_ref.final_grade))
01015         self.enable_fields()
01016
01017     def update_popular_answers(self):
01018         """
01019         In case length of the internal set structure changed, refills UI drop down list
01020         with new  values.
01021         :return:
01022         """
01023         if len(self.popular_answers) != len(self.grader_ref.input_suggestion):
01024             self.popular_answers.clear()
01025             self.popular_answers.addItems(self.grader_ref.input_suggestion)
01026             # for item in self.grader_ref.input_suggestion:
01027
01028     def save_grade(self):
01029         """
01030         Function bound to 'Save grade' button.
01031         Calls function that saves current grade.
01032         :return:
01033         """
01034         self.grader_ref.save_grade()
01035
01036     def save_response(self):
01037         """
01038         Function bound to 'Save response' button.
01039         Calls functions that save current commment.
01040         :return:
01041         """
01042         self.grader_ref.resp_text = self.input_response_browser.toPlainText()
01043         self.grader_ref.user_comment = self.input_response_browser_user.toPlainText()
01044         self.grader_ref.save_responce()
01045
01046     def save_all(self):
01047         """
01048         Function bound to 'Save all' button.
01049         Simply calls other 'save' functions.
01050         :return:
01051         """
01052         self.grader_ref.save_grade()
01053         # self.grader_ref.save_responce()
01054         self.save_response()
01055         self.grader_ref.save_all2()
01056
```

```
01057     def track_final_grade(self):
01058         """
01059         Saves manual grade changes into log.
01060         :return:
01061         """
01062         grade = self.input_final_grade.text()
01063         self.grader_ref.log_update('Manual grade change from : ' + str(self.grader_ref.final_grade))
01064         self.input_log_browser.setText(self.grader_ref.global_log)
01065         self.grader_ref.final_grade = int(grade)
01066         self.grader_ref.log_update('Manual grade change to: ' + str(grade))
01067         self.input_log_browser.setText(self.grader_ref.global_log)
01068 
01069     def setupUi(self, main_window):
01070         """
01071         Adds extra functionality to the UI generated by Qt Designer
01072         and converted to the python file.
01073         :return: nohting
01074         """
01075         super().setupUi(main_window)
01076 
01077         self.bind_functions()
01078         self.sync_params_to_settings()
01079 
01080         from pathlib import Path
01081         settings_location = str(Path(os.path.expandvars(os.path.expanduser('./settings.sqlite3'))).absolute())
01082         if os.path.isfile(settings_location):
01083             paths, local = settings_db_read_settings()
01084             try:
01085                 if len(os.walk(get_full_path(paths, local) + "/server_sync/").__next__()[1]) > 0:
01086                     if not self.manage_labs_but.isEnabled():
01087                         self.manage_labs_but.setEnabled(True)
01088                     if not self.but_file_open.isEnabled():
01089                         self.but_file_open.setEnabled(True)
01090                         self.input_file_location.setEnabled(True)
01091             except Exception as e:
01092                 print("Most likely you did not fill all the settings: ", e)
01093 
01094 
01095     def sync_params_to_settings(self):
01096         """
01097         Once returned from settings tab - updates grading path
01098         :return: Nothing
01099         """
01100         paths, local = settings_db_read_settings()
01101         working_dir = ''
01102         if paths and len(paths) == 4:
01103             self.logisim_path = paths[0]
01104             if len(paths[1]) > 0:
01105                 working_dir = paths[1]
01106             else:
01107                 working_dir = './'
01108         if local and len(local) >= 4:
01109             self.grader_name = local[0]
01110             working_dir += str(local[1])
01111             working_dir += '_' + local[2] + '/'
01112             self.set_style_checkbox.setChecked(bool(local[3]))
01113 
01114         if len(working_dir) > 0:
01115             self.input_file_location.setText(os.path.expanduser(working_dir))
01116 
01117 
01118     def bind_functions(self):
01119         """
01120         All bindings happen here.
01121         :return: nothing.
01122         """
01123         self.but_file_open.clicked.connect(self.my_open_file)
01124         self.but_begin.clicked.connect(self.load_dir)
01125         self.but_next.clicked.connect(self.next_circ)
01126         self.but_prev.clicked.connect(self.prev_circ)
01127         self.checkB_wrong.clicked.connect(self.check_wrong)
01128         #  self.but_regrade.clicked.connect(self.regrade)
01129         self.but_save_all.clicked.connect(self.save_all)
01130         self.but_save_response.clicked.connect(self.save_response)
01131         self.input_final_grade.textEdited.connect(self.track_final_grade)
01132         #  self.but_edit_done.clicked.connect(self.resp_edit_done)
01133         #  self.popular_answers.activated.connect(self.select_saved_answer)
01134         # self.but_create_report.setEnabled(True)  # Debug
01135         self.but_create_report.clicked.connect(self.generate_reports)
01136         # self.new_window_but.clicked.connect(self.open_dates_dialog)
01137         #  self.input_response_browser_user.focusInEvent(self, self.memorize_user_comment)
```

```
01138         #  self.custom_but_test.right_clicked[int].connect(self.dummy_d)
01139         self.input_file_location.dclicked.connect(self.open_file_diag)
01140         self.input_response_browser_user.focus_lost.connect(self.memorize_user_comment)
01141         self.popular_answers.currentIndexChanged.connect(self.update_user_comment_from_popular_answers)
01142         self.set_style_checkbox.stateChanged.connect(self.change_win_style)
01143         self.but_reset.clicked.connect(self.reset_grade_resp)
01144         self.settings_but.clicked.connect(self.open_settings_dialog)
01145         self.manage_labs_but.clicked.connect(self.open_manage_labs_diag)
01146         # self.sync_but.clicked.connect(self.sync_files)
01147 
01148 
01149 
01150     def change_win_style(self):
01151         """
01152         Adds nice style to the progress bar.
01153         style is defined above as global var.
01154         :return: nothing.
01155         """
01156         if self.set_style_checkbox.isChecked():
01157             self.progressBar.setStyleSheet(styleData)
01158         else:
01159             self.progressBar.setStyleSheet('')
01160 
01161     # noinspection PyMethodMayBeStatic
01162     def dummy_d_1(self):
01163         print('dummy_1 activated')
01164 
01165     def update_user_comment_from_popular_answers(self):
01166         """
01167         Updates user answer with selected popular answer.
01168         :return: nothing.
01169         """
01170         if self.popular_answers.hasFocus():
01171             self.input_response_browser_user.setPlainText(self.popular_answers.currentText())
01172 
01173     def open_file_diag(self):
01174         """
01175         Function bound to the doubleclick even handler of input field.
01176         Creates file dialog to select correct lab directory.
01177         :return: nothing.
01178         """
01179         obtained_dir = QFileDialog.getExistingDirectory(caption='Select directory with lab',
01180                                                         directory=self.input_file_location.text())
01181         if len(obtained_dir) > 1:
01182             self.input_file_location.setText(obtained_dir+'/')
01183 
01184     def memorize_user_comment(self):
01185         """
01186         If comment is already in popular answers - selects it,
01187         otherwise adds it to the popular answers.
01188         :return: nothing.
01189         """
01190         typed = self.input_response_browser_user.toPlainText()
01191         if hasattr(self, 'grader_ref') and typed:
01192             try:
01193                 index = self.popular_answers.findText(self.input_response_browser_user.toPlainText(),
01194                                                       QtCore.Qt.MatchFixedString)
01195                 if index >= 0:
01196                     self.popular_answers.setCurrentIndex(index)
01197                 else:
01198                     self.grader_ref.add_to_common_answers(typed)
01199                     self.update_popular_answers()
01200                     index = self.popular_answers.findText(self.input_response_browser_user.toPlainText(),
01201                                                           QtCore.Qt.MatchFixedString)
01202                     try:
01203                         self.popular_answers.setCurrentIndex(index)
01204                     except Exception as e:
01205                         print('Failed to select proper index: ', e)
01206                         raise
01207             except Exception as e:
01208                 print('failed to add popular answer: ', e)
01209 
01210     def kill_logisim(self):
01211         """
01212         Just kills logisim process by saved pid.
01213         :return: nothing.
01214         """
01215         try:
01216             self.grader_ref.logisim_pid.kill()
01217         except Exception as e:
01218             print("was not able to kill : ", e)
```

```
01219
01220     def run_logisim(self, filename):
01221         """
01222         Opens logisim in a separate process.
01223         Path is hardcoded, but will be changed once I have 'settings' window
01224         and|or database.
01225         :return: nothing.
01226         """
01227
01228         command = 'java -jar ' + self.logisim_path + 'logisim-generic-2.7.1.jar {}'.format(filename)
01229         # command_with_file = command + os.path.join(self.grader_ref.file_list[self.grader_ref.cur_idx], MAIN_FILE_NAME)
01230         #  if self.grader_ref.logisim_pid.pid > 0:
01231         self.kill_logisim()
01232         self.grader_ref.logisim_pid = subprocess.Popen(command, shell=True)
01233
01234     def generate_reports(self):
01235         """
01236         Function bound to 'Create reports' button.
01237         Calls generate_answers procedure from generate.py
01238         :return: nothing.
01239         """
01240         self.but_create_report.setDisabled(True)
01241         self.but_create_report.setText('Generating..')
01242         self.but_create_report.repaint()
01243         # from generate import generate_answers
01244         # (resubmit_num, dir_name, lab_type, lab_num)
01245         if hasattr(self, 'grader_ref'):
01246             loc_settings = settings_db_read_settings()[1]
01247             generate_answers3(self.grader_ref.lid, self.grader_ref.attempt, self.grader_ref.year, self.grader_ref.semester)
01248             # generate_answers(self.grader_ref.attempt, self.grader_ref.working_dir, self.grader_ref.lab_type, self.grader_ref.lab_num,
    loc_settings[1], loc_settings[2], self.grader_name)
01249             # generate_answers2(self.grader_ref.attempt, self.grader_ref.working_dir, self.grader_ref.lab_type, self.grader_ref.lab_num,
    loc_settings[1], loc_settings[2], self.grader_name)
01250             self.but_create_report.setEnabled(True)
01251             self.but_create_report.setText('Create reports')
01252
01253
01254
01255     # def open_dates_dialog(self):
01256     #     """
01257     #     Function bound to 'Create due dates' button.
01258     #     Creates new window, saves selected dates into files, but calling 'due_date_creator'
01259     #     :return: nothing.
01260     #     """
01261     #     self.new_window_but.setDisabled(True)
01262     #     self.cal_window = QtWidgets.QDialog()
01263     #     dui = Ui_Create_dates_dialog1()
01264     #     dui.setupUi(self.cal_window)
01265     #     # self.cal_window.finished.connect(self.check_new_win_result)
01266     #     self.cal_window.show()
01267     #     accepted = self.cal_window.exec_()
01268     #     if accepted:
01269     #         due_dates = list()
01270     #         due_dates.append(dui.init_subm_date_time.dateTime().toTime_t())
01271     #         due_dates.append(dui.first_subm_date_time.dateTime().toTime_t())
01272     #         due_dates.append(dui.second_subm_date_time.dateTime().toTime_t())
01273     #         due_dates.append(dui.third_subm_date_time.dateTime().toTime_t())
01274     #         due_location = dui.lab_path.text()
01275     #         self.due_date_creator(due_location, due_dates)
01276     #     self.new_window_but.setEnabled(True)
01277
01278     def open_settings_dialog(self):
01279         """
01280         Function bound to 'Open settings button'
01281         Creates new window to edit settings stored in the settings database.
01282         :return: nothing.
01283         """
01284         self.settings_but.setDisabled(True)
01285         self.settings_but.repaint()
01286         self.settings_window = QtWidgets.QDialog()
01287         dui = Ui_Create_settings_dialog()
01288         dui.setupUi(self.settings_window)
01289
01290         self.centralwidget.setDisabled(True)
01291         self.centralwidget.repaint()
01292
01293         self.settings_window.show()
01294         self.settings_window.exec_()
01295
01296         self.sync_params_to_settings()
01297         self.centralwidget.setEnabled(True)
```

```
01298
01299         self.settings_but.setEnabled(True)
01300
01301         if not self.manage_labs_but.isEnabled():
01302             from pathlib import Path
01303             settings_location = str(Path(os.path.expandvars(os.path.expanduser('./settings.sqlite3'))).absolute())
01304             if os.path.isfile(settings_location):
01305                 self.manage_labs_but.setEnabled(True)
01306
01307
01308     def open_manage_labs_diag(self):
01309         """
01310         Function bound to 'Open mangage labs button'
01311         Creates new window to mangage labs(import, export).
01312         :return: nothing
01313         """
01314         self.manage_labs_but.setDisabled(True)
01315         self.manage_labs_but.repaint()
01316         self.centralwidget.setDisabled(True)
01317         self.centralwidget.repaint()
01318         self.manage_labs_window = QtWidgets.QDialog()
01319         dui = Ui_manage_labs1()
01320         dui.setupUi(self.manage_labs_window)
01321
01322         self.manage_labs_window.show()
01323         self.manage_labs_window.exec_()
01324
01325         self.centralwidget.setEnabled(True)
01326         self.manage_labs_but.setEnabled(True)
01327
01328         if not self.but_file_open.isEnabled():
01329             paths, local = settings_db_read_settings()
01330             # if there are some labs in server sync directory:
01331             if len(os.walk(get_full_path(paths, local) + "/server_sync/").__next__()[1]) > 0:
01332                 self.but_file_open.setEnabled(True)
01333                 self.input_file_location.setEnabled(True)
01334
01335
01336 class Ui_Create_settings_dialog(Ui_Settings):
01337     """
01338     Creates window that provides user with convenient way of changing settings that are stored in sqlite3 db.
01339     """
01340
01341     def bind_functions(self):
01342         """
01343         Place where all the bindings happen.
01344         :return: nothing.
01345         """
01346         self.buttonBox.button(self.buttonBox.Reset).clicked.connect(self.update_user_input_with_paths)
01347         self.buttonBox.button(self.buttonBox.RestoreDefaults).clicked.connect(self.set_default_user_input_with_paths)
01348         self.buttonBox.button(self.buttonBox.Apply).clicked.connect(self.create_or_update_settings_db)
01349         self.buttonBox.button(self.buttonBox.Ok).clicked.connect(self.create_or_update_settings_db)
01350
01351         self.import_stuents_btn.clicked.connect(self.import_students)
01352
01353         # TODO: make 'personal' events and update only fields that have been changed
01354         self.input_logisim_path.textChanged.connect(self.set_apply_restet_active)
01355         self.input_local_stor.textChanged.connect(self.set_apply_restet_active)
01356         self.input_rem_stor.textChanged.connect(self.set_apply_restet_active)
01357         self.input_grader_name.textChanged.connect(self.set_apply_restet_active)
01358         self.spin_year.valueChanged.connect(self.set_apply_restet_active)
01359         self.semester_comboBox.currentIndexChanged.connect(self.set_apply_restet_active)
01360         self.style_checkBox.stateChanged.connect(self.set_apply_restet_active)
01361         self.sync_command.textChanged.connect(self.set_apply_restet_active)
01362         self.input_grades_db.textChanged.connect(self.set_apply_restet_active)
01363
01364     def setupUi(self, Settings):
01365         """
01366         Sets initial parameters for all needed fields
01367         :return: Nothing
01368         """
01369         super().setupUi(Settings)
01370         self.buttonBox.button(self.buttonBox.Reset).setDisabled(True)
01371         self.buttonBox.button(self.buttonBox.Apply).setDisabled(True)
01372         self.bind_functions()
01373         self.update_user_input_with_paths()
01374
01375     def update_user_input_with_paths(self):
01376         """
01377         Reads settings parameters from DB and sets appropriate fields with obtained values
01378         Warning: dependa on a number of settings obtained from read_settings_data
```

```
01379         :return: Nothing
01380         """
01381         paths, local = self.read_settings_data()
01382         if paths and len(paths) >= 4:
01383             self.input_logisim_path.setText(paths[0])
01384             self.input_local_stor.setText(paths[1])
01385             self.input_rem_stor.setText(paths[2])
01386             self.input_grades_db.setText(paths[3])
01387             self.groupBox_user.setEnabled(True)
01388 
01389         if local and len(local) >= 4:
01390             self.input_grader_name.setText(local[0])
01391             self.spin_year.setValue(local[1])
01392             self.semester_comboBox.setCurrentIndex(int(local[2]))
01393             self.style_checkBox.setChecked(bool(local[3]))
01394             self.sync_command.setText(local[4])
01395 
01396         if (paths and len(paths) >= 4 ) and (local and len(local) >= 4):
01397             self.spin_year.setEnabled(True)
01398             self.semester_comboBox.setEnabled(True)
01399             self.style_checkBox.setEnabled(True)
01400             self.input_grader_name.setEnabled(True)
01401             self.sync_command.setEnabled(True)
01402         # if (local and len(local) > 5) or len(paths):
01403         #         print('Obtained more settings than expected. Please check Ui_Create_settings_dialog.')
01404 
01405         self.buttonBox.button(self.buttonBox.Reset).setDisabled(True)
01406         self.buttonBox.button(self.buttonBox.Apply).setDisabled(True)
01407 
01408     def set_default_user_input_with_paths(self):
01409         """
01410         Sets predefined values to the most important fields.
01411         Additionally enables Apply and Reset buttons
01412         :return:
01413         """
01414         self.input_logisim_path.setText(" /Downloads/")
01415         self.input_local_stor.setText(" /Documents/3130_labs/")
01416         self.input_grades_db.setText(" /Documents/3130_labs/grades.sqlite3")
01417         self.input_rem_stor.setText("")  # impossible to predict
01418         self.groupBox_user.setEnabled(True)
01419         self.buttonBox.button(self.buttonBox.Reset).setEnabled(True)
01420         self.buttonBox.button(self.buttonBox.Apply).setEnabled(True)
01421 
01422     def read_settings_data(self):
01423         """
01424         reads settings from settings DB by calling function supplied by db_init.py
01425         :return: None for error, list of fields for success
01426         """
01427         self.buttonBox.button(self.buttonBox.Reset).setEnabled(True)
01428         return settings_db_read_settings()
01429 
01430     def create_or_update_settings_db(self):
01431         """
01432         Checks if settings DB file exists.
01433         If NO - attempts to create one (depends on user's choice)
01434         If YES - reads paramenters with function supplied by db_init.py
01435         In future it will contain checks for grades.db
01436         :return:
01437         """
01438         from pathlib import Path
01439         settings_location = str(Path(os.path.expandvars(os.path.expanduser('./settings.sqlite3'))).absolute())
01440         if not os.path.isfile(settings_location):
01441             if self.open_simple_dialog("Do you want to create settings database ?"):
01442                 if not settings_db_create(force=True):
01443                     raise Exception('Was not able to create SETTINGS db.')
01444         if len(self.input_local_stor.text()) > 0:
01445             if self.input_local_stor.text()[-1] != '/':
01446                 self.input_local_stor.setText(self.input_local_stor.text() + '/')
01447         if len(self.input_rem_stor.text()) > 0:
01448             if self.input_rem_stor.text()[-1] != '/':
01449                 self.input_rem_stor.setText(self.input_rem_stor.text() + '/')
01450         if len(self.input_logisim_path.text()) > 0:
01451             if self.input_logisim_path.text()[-1] != '/':
01452                 self.input_logisim_path.setText(self.input_logisim_path.text() + '/')
01453 
01454 
01455         paths = (self.input_logisim_path.text(), self.input_local_stor.text(), self.input_rem_stor.text(),
01456                  self.input_grades_db.text())
01457         if os.path.isfile(settings_location):
01458             local = (self.input_grader_name.text(), int(self.spin_year.text()),
01459                      self.semester_comboBox.currentIndex(), self.style_checkBox.checkState(), self.sync_command.text())
```

```
01460             if len(self.input_local_stor.text()) > 0:
01461                 local_stor = str(Path(os.path.expanduser(os.path.expandvars(self.input_local_stor.text()))).absolute())
01462                 if local_stor[-1] != '/':
01463                     local_stor += '/'
01464                 if not os.path.isdir(local_stor):
01465                     os.mkdir(local_stor)
01466                 local_grading_path = local_stor + self.spin_year.text() + '_' +\
01467                                      str(self.semester_comboBox.currentIndex())
01468                 if not os.path.isdir(local_grading_path):
01469                     os.mkdir(local_grading_path)
01470             update_settings(paths, local)
01471 
01472 
01473         grades_location = str(Path(os.path.expandvars(os.path.expanduser(self.input_grades_db.text()))).absolute())
01474         if len(self.input_grades_db.text()) > 1 and not os.path.isfile(grades_location):
01475             if self.open_simple_dialog("Do you want to create GRADES database ?"):
01476                 print('Before grades creation.')
01477                 if not grades_db_create(grades_location, force=True):
01478                     raise Exception('Was not able to create GRADES db.')
01479 
01480         if os.path.isfile(settings_location) and os.path.isfile(grades_location):
01481             self.buttonBox.button(self.buttonBox.Apply).setDisabled(True)
01482             self.buttonBox.button(self.buttonBox.Apply).repaint()
01483             self.buttonBox.button(self.buttonBox.Reset).setDisabled(True)
01484             self.buttonBox.button(self.buttonBox.Reset).repaint()
01485             if not self.groupBox_user.isEnabled():
01486                 self.groupBox_user.setEnabled(True)
01487             if not self.input_logisim_path.isEnabled():
01488                 self.input_logisim_path.setEnabled(True)
01489                 self.label_logisim_path.setEnabled(True)
01490             if not self.input_local_stor.isEnabled():
01491                 self.input_local_stor.setEnabled(True)
01492                 self.label_local_stor.setEnabled(True)
01493             if not self.input_rem_stor.isEnabled():
01494                 self.input_rem_stor.setEnabled(True)
01495                 self.label_rem_stor.setEnabled(True)
01496             if not self.spin_year.isEnabled():
01497                 self.spin_year.setEnabled(True)
01498             if not self.semester_comboBox.isEnabled():
01499                 self.semester_comboBox.setEnabled(True)
01500             if not self.style_checkBox.isEnabled():
01501                 self.style_checkBox.setEnabled(True)
01502             if not self.input_grader_name.isEnabled():
01503                 self.input_grader_name.setEnabled(True)
01504             if not self.sync_command.isEnabled():
01505                 self.sync_command.setEnabled(True)
01506 
01507         # if len(self.input_local_stor.text()) > 1:
01508         #     full_path = Path(self.input_local_stor.text()).absolute()
01509         #     if not os.path.exists(full_path) or not os.path.isdir(full_path):
01510         #         os.makedirs(full_path)
01511 
01512     def import_students(self):
01513         """
01514         creates dialog with selector for students file to parse and input into the db
01515         :return: Nothing
01516         """
01517         self.import_stuents_btn.setEnabled(False)
01518         stud_file = QFileDialog.getOpenFileName(caption="Select file with students' info", directory='.', filter="Text files (*.txt)")
01519         if len(stud_file[0]) > 3:
01520             load_student_list_into_grades_db(self.input_grades_db.text(), self.spin_year.value(), self.semester_comboBox.currentIndex(),
    filename=stud_file[0])
01521 
01522 
01523         self.import_stuents_btn.setEnabled(True)
01524 
01525     def set_apply_restet_active(self):
01526         """
01527         Enables reset and apply buttons
01528         :return: Nothing
01529         """
01530         self.buttonBox.button(self.buttonBox.Reset).setEnabled(True)
01531         self.buttonBox.button(self.buttonBox.Apply).setEnabled(True)
01532 
01533     def __dummy(self):
01534         print('dummy exec')
01535 
01536     def open_simple_dialog(self, phrase):
01537         """
01538         Creates simple Ok|Cancel dialog and returns user's choice
01539         :param phrase: Phrase to ask the user
```

```
01540         :return: True for Ok, False otherwise
01541         """
01542         self.simple_diag = QtWidgets.QDialog()
01543         dui = SimpleDialog()
01544         dui.setupUi(self.simple_diag, phrase)
01545 
01546         self.buttonBox.setDisabled(True)
01547         self.buttonBox.repaint()
01548 
01549         self.simple_diag.setWindowTitle('Settings confirmation')
01550         self.simple_diag.show()
01551 
01552         result = self.simple_diag.exec_()
01553 
01554         self.buttonBox.setEnabled(True)
01555 
01556         return result
01557 
01558 
01559 class SimpleDialog(Ui_Dialog):
01560     """
01561     Wrapper class for very simple Ok|Cancel dialog
01562     """
01563     def setupUi(self, Dialog, phrase):
01564         """
01565         :param phrase: Phrase to display
01566         :return: Nothing
01567         """
01568         super().setupUi(Dialog)
01569         self.label_main_question.setText(phrase)
01570 
01571 
01572 class Ui_manage_labs1(Ui_manage_labs):
01573     srv_sync_path = None
01574     selected_path = None
01575     selected_lab_name = None
01576     zip_files_len = None
01577 
01578     def bind_functions(self):
01579         self.labs_select_comboBox.currentIndexChanged.connect(self.update_status_bar)
01580         self.import_but.clicked.connect(self.import_lab)
01581         self.create_due_dates_but.clicked.connect(self.open_dates_dialog)
01582         # self.sync_but.clicked.connect(lambda i: self.sync_but.setDisabled(True))
01583         self.sync_but.clicked.connect(self.sync_files)
01584         self.export_but.clicked.connect(self.export_pdfs)
01585 
01586     def setupUi(self, manage_labs):
01587         super().setupUi(manage_labs)
01588         self.bind_functions()
01589         self.set_local_vars()
01590 
01591         try:
01592             self.scan_for_labs()
01593             if self.labs_select_comboBox.count() > 0:
01594                 self.labs_select_comboBox.setEnabled(True)
01595                 self.import_but.setEnabled(True)
01596                 self.create_due_dates_but.setEnabled(True)
01597                 self.export_but.setEnabled(True)
01598         except Exception as e:
01599             print('Error in manage labs. Probably your grading path was not set properly: ', e)
01600 
01601 
01602     def set_local_vars(self):
01603         pass
01604 
01605     def update_status_bar(self, force=False):
01606         # no need to scan files in background, but only when user selects it intentionally, or if it is first run
01607         if self.labs_select_comboBox.hasFocus() or force:
01608             self.selected_lab_name = self.labs_select_comboBox.currentText()
01609             self.selected_path = self.srv_sync_path + self.selected_lab_name + '/'
01610             zip_pdf_files = [f for f in os.listdir(self.selected_path) if '.zip' in f or '.pdf' in f]
01611 
01612             self.pdf_files_len = len([f for f in zip_pdf_files if f.split('.')[1] == 'pdf'])
01613             self.zip_files_len = len([f for f in zip_pdf_files if f.split('.')[1] == 'zip'])
01614 
01615             self.status_bar.setText("Contains " + str(self.zip_files_len) + ' zip files and ' + str(self.pdf_files_len) + ' pdf files.')
01616 
01617             if self.zip_files_len > 0 and not self.create_due_dates_but.isEnabled():
01618                 self.export_but.setEnabled(True)
01619                 self.import_but.setEnabled(True)
01620                 self.labs_select_comboBox.setEnabled(True)
```

```
01621
01622           # good_zip_files_size = len([f for f in zip_files if os.isfile(os.path.join(selected_path, f))])
01623
01624       def sync_files(self):
01625           self.sync_but.setDisabled(True)
01626           self.sync_but.setText('Synchronizing...')
01627           self.sync_but.repaint()
01628           self.status_bar.setText("Synchronizing...")
01629           self.status_bar.repaint()
01630           sync_files()
01631           self.status_bar.setText("Done.")
01632           self.sync_but.setText('Sync to local storage')
01633           self.sync_but.setEnabled(True)
01634
01635           sync_success = True  # there are no tools to check it at this point.
01636           if sync_success and not self.labs_select_comboBox.isEnabled():
01637               self.labs_select_comboBox.setEnabled(True)
01638               self.create_due_dates_but.setEnabled(True)
01639               self.scan_for_labs()
01640               #  TODO: There should be additional checks to enable import and export, but I do not have enough time to implement them.
01641               self.import_but.setEnabled(True)
01642               self.export_but.setEnabled(True)
01643
01644       def scan_for_labs(self):
01645           """
01646           Scans local repository for labs to import into grading path
01647           Offers creation of the due date for particular lab.
01648           :return: Nothing
01649           """
01650           paths, local = settings_db_read_settings()
01651           # self.local_path = paths[1] + str(local[1]) + '_' + str(local[2]) + '/'
01652           self.main_lab_path = get_full_path(paths, local)
01653           self.srv_sync_path = self.main_lab_path + "/server_sync/"
01654           dirs = os.walk(self.srv_sync_path).__next__()[1]
01655           if len(dirs) > 0:
01656               self.labs_select_comboBox.addItems(sorted(dirs))
01657               self.labs_select_comboBox.setCurrentIndex(0)
01658               self.labs_select_comboBox.setFocus(True)
01659               self.update_status_bar(force=True)
01660
01661
01662       def import_lab(self):
01663           if self.selected_path:
01664               self.import_but.setDisabled(True)
01665               self.import_but.setText('Importing..')
01666               self.import_but.repaint()
01667
01668               # due_file = self.check_for_due_dates(self.selected_path)
01669               if False:
01670               # if len(due_file) < 4:
01671                   self.status_bar.setText('Create due dates !')
01672                   self.import_but.setText('Import labs')
01673                   self.import_but.setEnabled(True)
01674                   return False
01675               else:
01676                   from shutil import copy2 as cp2
01677                   zip_files = [f for f in os.listdir(self.selected_path) if 'zip' in f]
01678                   real_zip_files_rev = sorted([f for f in zip_files if os.path.isfile(os.path.join(self.selected_path, f))], reverse=True)
01679
01680                   year, semester = self.main_lab_path.split('/')[-1].split('_')
01681                   ltype, _, lab_num = self.selected_lab_name.split('_')
01682                   lid = get_labid_in_schedule(get_lab_id(ltype, int(lab_num)), year, semester)
01683                   if lid is None:
01684                       self.status_bar.setText('Create due dates ! Lab is not initialised in lab_schedule')
01685                       self.import_but.setText('Import labs')
01686                       self.import_but.setEnabled(True)
01687                       return False
01688                   current_check, prev_due, next_due, current_timestamp = get_grading_period(lid)
01689
01690                   if current_check > 4:
01691                       self.status_bar.setText('This lab has no more resubmissions (graded 4 times).')
01692                       self.import_but.setText('Import labs')
01693                       self.import_but.setEnabled(True)
01694                       return False
01695
01696                   if current_timestamp < next_due:
01697                       # we cannot grade before the due date
01698                       self.status_bar.setText('Current date is less than next due date. It is too early to import.')
01699                       self.import_but.setText('Import labs')
01700                       self.import_but.setEnabled(True)
01701                       return False
```

```
01702
01703
01704
01705                 penalty_mess = ''
01706                 if current_check == 1:
01707                     penalty_mess = '100% - this is your max point(no resubmissions)'
01708                 elif current_check == 2:
01709                     penalty_mess = '90% - first resubmission'
01710                 elif current_check == 3:
01711                     penalty_mess = '70% - second resubmission'
01712                 elif current_check == 4:
01713                     penalty_mess = '50% - third resubmission'
01714
01715                 lab_type, _, lab_num = self.selected_lab_name.split('_')
01716                 lab_corr_name = lab_type[0] + 'LA' + lab_num
01717                 max_points = get_lab_max_value(lab_corr_name)
01718                 lab_filename = get_lab_filename(lab_corr_name)
01719
01720                 # temporary solution. path should be stored as local var
01721                 paths_to_grading_dir = self.main_lab_path + '/' + self.selected_lab_name + '_' + str(current_check) + '/'
01722
01723                 # proc_time = datetime.utcfromtimestamp(current_timestamp).strftime('%Y-%m-%d %H:%M:%S')
01724                 proc_time = time_to_str_with_tz(current_timestamp)
01725
01726                 # File manipulations goes below:
01727
01728                 if not os.path.isdir(paths_to_grading_dir):
01729                     os.makedirs(paths_to_grading_dir)
01730
01731                 cur_year, cur_sem = paths_to_grading_dir.split('/')[-3].split('_')
01732                 id_to_classId = get_ids_in_class_by_year_semester(cur_year, cur_sem)[0]
01733                 imported_files_counter = 0
01734
01735                 selected_files = []
01736                 for file in real_zip_files_rev:
01737                     parts = file.split('.')[0].split('-')
01738                     if int(parts[2]) > prev_due and int(parts[2]) <= next_due:
01739                         if len(selected_files) == 0:
01740                             selected_files.append(file)
01741                         elif selected_files[-1].split('.')[0].split('-')[0] != parts[0]:
01742                             selected_files.append(file)
01743
01744                 for file in reversed(selected_files):
01745                     zipped_file = zipfile.ZipFile(self.selected_path + file)
01746                     extraction_dir = paths_to_grading_dir + file.split('.')[0]
01747                     try:
01748                         zipped_file.extractall(paths_to_grading_dir + file.split('.')[0])
01749                     except Exception as e:
01750                         print(self.selected_path + file)
01751                         print(e)
01752                     finally:
01753                         zipped_file.close()
01754                     parts = file.split('.')[0].split('-')
01755                     subm_int = int(extraction_dir.split('-')[-1])
01756                     # subm_time = datetime.utcfromtimestamp(subm_int).replace(tzinfo=tz.tzutc()).astimezone(tz.tzlocal()).strftime('%Y-%m-%d
    %H:%M:%S')
01757                     subm_time = time_to_str_with_tz(subm_int)
01758                     # check for required files
01759                     if not lab_filename[0] or os.path.isfile(extraction_dir + '/' + lab_filename[0]):
01760                         lab_responce = 'I did not find any errors. Good job !'
01761                         cur_grade = max_points
01762                     else:
01763                         lab_responce = 'File "' + lab_filename[0] +'" was not found.\nThese files were found: ' +\
01764                                        " ".join(os.listdir(extraction_dir))
01765                         cur_grade = 0
01766
01767                     # This check is for a case when you graded the lab and trying to import it again.
01768                     # No existing files should be wiped
01769                     if not os.path.isfile(extraction_dir+'/penalty.txt'):
01770                         with open(extraction_dir+'/penalty.txt', 'w') as f:
01771                             f.write(penalty_mess)
01772
01773                     if not os.path.isfile(extraction_dir + '/grade.txt'):
01774                         with open(extraction_dir + '/grade.txt', 'w') as f:
01775                             f.write(str(cur_grade))
01776
01777                     if not os.path.isfile(extraction_dir + '/responce.txt'):
01778                         with open(extraction_dir + '/responce.txt', 'w') as f:
01779                             f.write(lab_responce)
01780
01781                     if not os.path.isfile(extraction_dir + '/tech_info.txt'):
```

```
01782                         with open(extraction_dir + '/tech_info.txt', 'w') as f:
01783                             f.writelines(['File was submited at %s<br/>\n' % subm_time,
01784                                           'I started processing your file at %s<br/>\n' % proc_time,
01785                                           "I found that your lab type is '%s' and it's number is %s <br/>" % (lab_type, lab_num),
01786                                           'So max points for this lab type is <u>%d</u><br/>' % max_points,
01787                                           'Theoretical max points: %s)' % penalty_mess])
01788 
01789                     init_new_lab(id_to_classId[parts[0]], lid, current_check, subm_int, extraction_dir)
01790                     imported_files_counter += 1
01791 
01792                 # cp2(self.selected_path + due_file[current_check-1], paths_to_grading_dir)
01793 
01794                 # check_filename = paths_to_grading_dir + 'check_' + str(current_check) + '_' + str(current_timestamp)
01795                 # with open(check_filename, 'w'): pass
01796                 gen_report(lid, att=current_check)
01797 
01798                 # cp2(check_filename, self.selected_path)
01799 
01800                 self.import_but.setEnabled(True)
01801                 self.import_but.setText('Import labs')
01802                 self.status_bar.setText("Imported " + str(imported_files_counter) + " files.")
01803                 return True
01804 
01805         return False
01806 
01807 
01808 
01809     def check_for_due_dates(self, dir):
01810         """
01811         It will be using old desing of due files.
01812         I am going to switch to DB due dates when the time comes.
01813         At this point DB has full support for it.
01814         :param dir:
01815         :return:
01816         """
01817         return sorted([f for f in os.listdir(dir) if 'due_' in f])
01818 
01819 
01820     def open_dates_dialog(self):
01821         """
01822         Function bound to 'Create due dates' button.
01823         Creates new window, saves selected dates into files, but calling 'due_date_creator'
01824         :return: nothing.
01825         """
01826         self.create_due_dates_but.setDisabled(True)
01827         self.create_due_dates_but.repaint()
01828         self.cal_window = QtWidgets.QDialog()
01829         dui = Ui_Create_dates_dialog1()
01830         dui.setupUi(self.cal_window, self.selected_lab_name)
01831         # self.cal_window.finished.connect(self.check_new_win_result)
01832         self.cal_window.show()
01833         accepted = self.cal_window.exec_()
01834         if accepted:
01835             due_dates = list()
01836             due_dates.append(dui.init_subm_date_time.dateTime().toTime_t())
01837             due_dates.append(dui.first_subm_date_time.dateTime().toTime_t())
01838             due_dates.append(dui.second_subm_date_time.dateTime().toTime_t())
01839             due_dates.append(dui.third_subm_date_time.dateTime().toTime_t())
01840             due_location = dui.lab_path.text()
01841             self.due_date_creator(due_location, due_dates)
01842             year, semester = self.main_lab_path.split('/')[-1].split('_')
01843             ltype, _, lab_num = self.selected_lab_name.split('_')
01844             register_lab_in_semester(ltype, lab_num, year, semester, due_dates)
01845         self.create_due_dates_but.setEnabled(True)
01846 
01847     # noinspection PyMethodMayBeStatic
01848     def due_date_creator(self, due_location, due_dates):
01849         """
01850         Saves due files into location specified by user.
01851         :param due_location: where to save.
01852         :param due_dates: list of dates to save.
01853         :return: nothing.
01854         """
01855         if len(due_location) > 1:
01856             i = 1
01857             for due_date in due_dates:
01858                 with open('%sdue_%d_%d' % (due_location, i, due_date), 'w'):
01859                     i += 1
01860         else:
01861             print('Location was not specified.')
01862 
```

```
01863     def export_pdfs(self):
01864         self.export_but.setDisabled(True)
01865         self.export_but.setText('Exporting..')
01866         self.export_but.repaint()
01867         export_pdf()
01868         self.export_but.setText('Export pdfs')
01869         self.export_but.setEnabled(True)
01870 
01871 
01872 def get_grading_period(lid, cur_only=False):
01873     # should comput correct grading period and return the due date in Unix timestamp format
01874     import time
01875     # due_timestamps = [int(f.split('_')[2]) for f in due_files]
01876 
01877     current_timestamp = int(time.time())
01878     due_timestamps1 = get_due_date_by_labid(lid)
01879     import_timestamps1 = get_import_dates_by_labid(lid)
01880     cur_check = len(due_timestamps1)
01881     for i, ts in enumerate(import_timestamps1):
01882         if ts is None:
01883             cur_check = i
01884             break
01885     i = 0
01886     if cur_check:
01887         while i < len(due_timestamps1) and import_timestamps1[i] is not None and due_timestamps1[i] < current_timestamp and due_timestamps1[i]
      < import_timestamps1[cur_check-1]:
01888             i += 1
01889 
01890     if cur_only:  # neede for CLA2-2
01891         i = max(0, i-1)
01892 
01893     if i == 0:
01894         from_time = 0
01895         to_time = due_timestamps1[i]
01896     elif i > len(due_timestamps1)-1:
01897         from_time = due_timestamps1[i-1]
01898         to_time = int(time.time())
01899     else:
01900         from_time = due_timestamps1[i - 1]
01901         to_time = due_timestamps1[i]
01902 
01903     cur_check_num = i+1
01904     # cur_check += 1
01905 
01906 
01907     #
01908     # check_files = [int(f.split('_')[2]) for f in os.listdir(dir) if 'check_' in f]
01909     # if len(check_files) > 0:
01910     #     if len(check_files) >= 4:
01911     #         cur_check_num = 0
01912     #         from_time = 0
01913     #         to_time = 0
01914     #     else:
01915     #         cur_check_num = len(check_files) + 1           # 1 + 1
01916     #         from_time = due_timestamps[cur_check_num - 2]  # 0 => after first due date
01917     #         to_time = due_timestamps[cur_check_num - 1]    # 1 => before second due date
01918     # else:
01919     #     from_time = 0
01920     #     to_time = due_timestamps[0]
01921     #     cur_check_num = 1
01922 
01923     return cur_check_num, from_time, to_time, current_timestamp
01924 
01925 
01926 class Ui_Create_dates_dialog1(Ui_Create_dates_dialog):
01927 
01928     def bind_functions(self):
01929         """
01930         Place where all the bindings happen.
01931         :return: Nothing
01932         """
01933         self.init_subm_date_time.dateTimeChanged.connect(self.date_select)
01934         # self.select_file_path.clicked.connect(self.open_file_diag)
01935         # self.lineEdit.left_clicked[int].connect(self.dummy_d)
01936         self.lab_path.dclicked.connect(self.open_file_diag)
01937 
01938     # noinspection PyMethodMayBeStatic
01939     def __dummy_d(self, nb):
01940         """
01941         It is here for testing new features
01942         :param nb:
```

```
01943         :return:
01944         """
01945         if nb == 1:
01946             print('Single left click ', nb)
01947         else:
01948             print('Double left click ', nb)
01949 
01950     # noinspection PyMethodMayBeStatic
01951     def __dummy_d_1(self):
01952         """
01953         It is here for testing new features
01954         :return:
01955         """
01956         print('Single left click ')
01957 
01958     def setupUi(self, Create_dates_dialog, selected_lab=''):
01959         """
01960         Initiates creation of the new window for
01961         due dates creation.
01962         Adds binded functions and sets some important variables (like current time).
01963         :param Create_dates_dialog - parent class generated with Qt5 Designer.
01964         :return: nothing.
01965         """
01966         super().setupUi(Create_dates_dialog)
01967         self.bind_functions()
01968         self.init_subm_date_time.setDateTime(QDateTime.currentDateTime())
01969         paths, local = settings_db_read_settings()
01970         good_path = get_full_path(paths, local) + '/server_sync/'
01971         try:
01972             good_path += selected_lab + '/'
01973         except Exception as e:
01974             print('Exception when tried to append selected folder from Manage labs. ', e)
01975         self.lab_path.setText(good_path)
01976 
01977     def date_select(self):
01978         """
01979         Automatically set next due dates.
01980         :return: nothing.
01981         """
01982         self.first_subm_date_time.setDateTime(self.init_subm_date_time.dateTime().addDays(7))
01983         self.second_subm_date_time.setDateTime(self.init_subm_date_time.dateTime().addDays(14))
01984         self.third_subm_date_time.setDateTime(self.init_subm_date_time.dateTime().addDays(21))
01985 
01986     def open_file_diag(self):
01987         """
01988         Creates File browser in a new window.
01989         :return: nothing.
01990         """
01991         obtained_dir = QFileDialog.getExistingDirectory(caption='Select where to create due files',
01992                                                         directory=self.lab_path.text())+'/'
01993         if len(obtained_dir) > 1:
01994             self.lab_path.setText(obtained_dir)
01995 
01996 
01997 if __name__ == "__main__":
01998     # generate_final_grades('./grades.sqlite3', 2018, 1)
01999     # reconstruct_grades_and_comments()
02000     # update_lab_submissions_paths('./grades.sqlite3', '/home/vanya/Documents/3130_labs/2018/', 2018, 1)
02001     # import_previous_grades_into_db('./grades.sqlite3', 'grades.xls', 2018, 1)
02002     # load_student_list_into_grades_db('./grades.sqlite3', 2018, '1')
02003 
02004     # sync_files()
02005 
02006     app = QtWidgets.QApplication(sys.argv)
02007     MainWindow = QtWidgets.QMainWindow()
02008     ui = UiMainWindow1()
02009     ui.setupUi(MainWindow)
02010     # MainWindow.setStyleSheet(styleData)
02011     MainWindow.show()
02012     sys.exit(app.exec_())
```

8.11 main_window.py File Reference

Classes

- class main_window.Ui_mainWindow

Namespaces

- main_window

8.12 main_window.py

```
00001 # -*- coding: utf-8 -*-
00002
00003 # Form implementation generated from reading ui file 'main_window.ui'
00004 #
00005 # Created by: PyQt5 UI code generator 5.12.dev1812231618
00006 #
00007 # WARNING! All changes made in this file will be lost!
00008
00009 from PyQt5 import QtCore, QtGui, QtWidgets
00010
00011 class Ui_mainWindow(object):
00012     def setupUi(self, mainWindow):
00013         mainWindow.setObjectName("mainWindow")
00014         mainWindow.setEnabled(True)
00015         mainWindow.resize(888, 584)
00016         icon = QtGui.QIcon()
00017         icon.addPixmap(QtGui.QPixmap("os_linux_1.ico"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
00018         mainWindow.setWindowIcon(icon)
00019         mainWindow.setAccessibleName("")
00020         self.centralwidget = QtWidgets.QWidget(mainWindow)
00021         self.centralwidget.setObjectName("centralwidget")
00022         self.verticalLayout_7 = QtWidgets.QVBoxLayout(self.centralwidget)
00023         self.verticalLayout_7.setObjectName("verticalLayout_7")
00024         self.horizontalLayout_12 = QtWidgets.QHBoxLayout()
00025         self.horizontalLayout_12.setObjectName("horizontalLayout_12")
00026         self.input_file_location = BetterLineEdit(self.centralwidget)
00027         self.input_file_location.setEnabled(False)
00028         self.input_file_location.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00029         self.input_file_location.setText("")
00030         self.input_file_location.setObjectName("input_file_location")
00031         self.horizontalLayout_12.addWidget(self.input_file_location)
00032         self.filename_lineEdit = QtWidgets.QLineEdit(self.centralwidget)
00033         self.filename_lineEdit.setMaximumSize(QtCore.QSize(90, 16777215))
00034         self.filename_lineEdit.setReadOnly(True)
00035         self.filename_lineEdit.setObjectName("filename_lineEdit")
00036         self.horizontalLayout_12.addWidget(self.filename_lineEdit)
00037         self.but_file_open = QtWidgets.QPushButton(self.centralwidget)
00038         self.but_file_open.setEnabled(False)
00039         self.but_file_open.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00040         self.but_file_open.setObjectName("but_file_open")
00041         self.horizontalLayout_12.addWidget(self.but_file_open)
00042         self.but_begin = QtWidgets.QPushButton(self.centralwidget)
00043         self.but_begin.setEnabled(False)
00044         self.but_begin.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00045         self.but_begin.setCheckable(False)
00046         self.but_begin.setAutoDefault(False)
00047         self.but_begin.setDefault(False)
00048         self.but_begin.setFlat(False)
00049         self.but_begin.setObjectName("but_begin")
00050         self.horizontalLayout_12.addWidget(self.but_begin)
00051         self.verticalLayout_7.addLayout(self.horizontalLayout_12)
00052         self.horizontalLayout_7 = QtWidgets.QHBoxLayout()
00053         self.horizontalLayout_7.setSpacing(6)
00054         self.horizontalLayout_7.setObjectName("horizontalLayout_7")
00055         self.verticalLayout = QtWidgets.QVBoxLayout()
00056         self.verticalLayout.setObjectName("verticalLayout")
00057         self.horizontalLayout = QtWidgets.QHBoxLayout()
00058         self.horizontalLayout.setObjectName("horizontalLayout")
00059         self.label_from = QtWidgets.QLabel(self.centralwidget)
00060         self.label_from.setObjectName("label_from")
00061         self.horizontalLayout.addWidget(self.label_from)
00062         self.dateTimeEdit_from = QtWidgets.QDateTimeEdit(self.centralwidget)
00063         self.dateTimeEdit_from.setEnabled(True)
00064         self.dateTimeEdit_from.setWrapping(False)
00065         self.dateTimeEdit_from.setReadOnly(True)
00066         self.dateTimeEdit_from.setAccelerated(False)
00067         self.dateTimeEdit_from.setCalendarPopup(True)
00068         self.dateTimeEdit_from.setObjectName("dateTimeEdit_from")
00069         self.horizontalLayout.addWidget(self.dateTimeEdit_from)
00070         self.verticalLayout.addLayout(self.horizontalLayout)
00071         self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
00072         self.horizontalLayout_2.setObjectName("horizontalLayout_2")
00073         self.label_submitted = QtWidgets.QLabel(self.centralwidget)
00074         self.label_submitted.setObjectName("label_submitted")
00075         self.horizontalLayout_2.addWidget(self.label_submitted)
00076         self.dateTimeEdit_submitted = QtWidgets.QDateTimeEdit(self.centralwidget)
00077         self.dateTimeEdit_submitted.setEnabled(True)
00078         self.dateTimeEdit_submitted.setWrapping(False)
00079         self.dateTimeEdit_submitted.setFrame(True)
```

```
00080         self.dateTimeEdit_submitted.setReadOnly(True)
00081         self.dateTimeEdit_submitted.setKeyboardTracking(False)
00082         self.dateTimeEdit_submitted.setCalendarPopup(True)
00083         self.dateTimeEdit_submitted.setObjectName("dateTimeEdit_submitted")
00084         self.horizontalLayout_2.addWidget(self.dateTimeEdit_submitted)
00085         self.verticalLayout.addLayout(self.horizontalLayout_2)
00086         self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
00087         self.horizontalLayout_3.setObjectName("horizontalLayout_3")
00088         self.label_to = QtWidgets.QLabel(self.centralwidget)
00089         self.label_to.setObjectName("label_to")
00090         self.horizontalLayout_3.addWidget(self.label_to)
00091         self.dateTimeEdit_to = QtWidgets.QDateTimeEdit(self.centralwidget)
00092         self.dateTimeEdit_to.setEnabled(True)
00093         self.dateTimeEdit_to.setReadOnly(True)
00094         self.dateTimeEdit_to.setCalendarPopup(True)
00095         self.dateTimeEdit_to.setObjectName("dateTimeEdit_to")
00096         self.horizontalLayout_3.addWidget(self.dateTimeEdit_to)
00097         self.verticalLayout.addLayout(self.horizontalLayout_3)
00098         self.horizontalLayout_7.addLayout(self.verticalLayout)
00099         self.verticalLayout_3 = QtWidgets.QVBoxLayout()
00100         self.verticalLayout_3.setObjectName("verticalLayout_3")
00101         self.horizontalLayout_8 = QtWidgets.QHBoxLayout()
00102         self.horizontalLayout_8.setObjectName("horizontalLayout_8")
00103         self.input_current_id = QtWidgets.QLineEdit(self.centralwidget)
00104         self.input_current_id.setEnabled(False)
00105         self.input_current_id.setMaximumSize(QtCore.QSize(60, 40))
00106         self.input_current_id.setReadOnly(True)
00107         self.input_current_id.setObjectName("input_current_id")
00108         self.horizontalLayout_8.addWidget(self.input_current_id)
00109         self.label_current_id = QtWidgets.QLabel(self.centralwidget)
00110         self.label_current_id.setObjectName("label_current_id")
00111         self.horizontalLayout_8.addWidget(self.label_current_id)
00112         self.verticalLayout_3.addLayout(self.horizontalLayout_8)
00113         self.horizontalLayout_9 = QtWidgets.QHBoxLayout()
00114         self.horizontalLayout_9.setObjectName("horizontalLayout_9")
00115         self.input_attempt = QtWidgets.QLineEdit(self.centralwidget)
00116         self.input_attempt.setEnabled(False)
00117         self.input_attempt.setMaximumSize(QtCore.QSize(40, 40))
00118         self.input_attempt.setReadOnly(True)
00119         self.input_attempt.setObjectName("input_attempt")
00120         self.horizontalLayout_9.addWidget(self.input_attempt)
00121         spacerItem = QtWidgets.QSpacerItem(20, 20, QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Minimum)
00122         self.horizontalLayout_9.addItem(spacerItem)
00123         self.label_attempt = QtWidgets.QLabel(self.centralwidget)
00124         self.label_attempt.setObjectName("label_attempt")
00125         self.horizontalLayout_9.addWidget(self.label_attempt)
00126         self.verticalLayout_3.addLayout(self.horizontalLayout_9)
00127         spacerItem1 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)
00128         self.verticalLayout_3.addItem(spacerItem1)
00129         self.horizontalLayout_7.addLayout(self.verticalLayout_3)
00130         self.verticalLayout_2 = QtWidgets.QVBoxLayout()
00131         self.verticalLayout_2.setObjectName("verticalLayout_2")
00132         self.horizontalLayout_6 = QtWidgets.QHBoxLayout()
00133         self.horizontalLayout_6.setObjectName("horizontalLayout_6")
00134         self.input_max_pos_grade = QtWidgets.QLineEdit(self.centralwidget)
00135         self.input_max_pos_grade.setEnabled(False)
00136         self.input_max_pos_grade.setMaximumSize(QtCore.QSize(40, 40))
00137         self.input_max_pos_grade.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00138         self.input_max_pos_grade.setText("")
00139         self.input_max_pos_grade.setReadOnly(True)
00140         self.input_max_pos_grade.setObjectName("input_max_pos_grade")
00141         self.horizontalLayout_6.addWidget(self.input_max_pos_grade)
00142         self.label_max_pos = QtWidgets.QLabel(self.centralwidget)
00143         self.label_max_pos.setEnabled(True)
00144         self.label_max_pos.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00145         self.label_max_pos.setObjectName("label_max_pos")
00146         self.horizontalLayout_6.addWidget(self.label_max_pos)
00147         self.verticalLayout_2.addLayout(self.horizontalLayout_6)
00148         self.horizontalLayout_4 = QtWidgets.QHBoxLayout()
00149         self.horizontalLayout_4.setObjectName("horizontalLayout_4")
00150         self.input_subtract = QtWidgets.QLineEdit(self.centralwidget)
00151         self.input_subtract.setEnabled(False)
00152         self.input_subtract.setMaximumSize(QtCore.QSize(40, 40))
00153         self.input_subtract.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00154         self.input_subtract.setReadOnly(True)
00155         self.input_subtract.setObjectName("input_subtract")
00156         self.horizontalLayout_4.addWidget(self.input_subtract)
00157         self.label_subtr = QtWidgets.QLabel(self.centralwidget)
00158         self.label_subtr.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00159         self.label_subtr.setObjectName("label_subtr")
00160         self.horizontalLayout_4.addWidget(self.label_subtr)
```

```
00161         self.verticalLayout_2.addLayout(self.horizontalLayout_4)
00162         self.horizontalLayout_5 = QtWidgets.QHBoxLayout()
00163         self.horizontalLayout_5.setObjectName("horizontalLayout_5")
00164         self.input_final_grade = QtWidgets.QLineEdit(self.centralwidget)
00165         self.input_final_grade.setEnabled(False)
00166         self.input_final_grade.setMaximumSize(QtCore.QSize(40, 40))
00167         self.input_final_grade.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00168         self.input_final_grade.setText("")
00169         self.input_final_grade.setReadOnly(True)
00170         self.input_final_grade.setObjectName("input_final_grade")
00171         self.horizontalLayout_5.addWidget(self.input_final_grade)
00172         self.label_final = QtWidgets.QLabel(self.centralwidget)
00173         self.label_final.setEnabled(True)
00174         self.label_final.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00175         self.label_final.setObjectName("label_final")
00176         self.horizontalLayout_5.addWidget(self.label_final)
00177         self.verticalLayout_2.addLayout(self.horizontalLayout_5)
00178         self.horizontalLayout_7.addLayout(self.verticalLayout_2)
00179         self.verticalLayout_4 = QtWidgets.QVBoxLayout()
00180         self.verticalLayout_4.setObjectName("verticalLayout_4")
00181         self.but_regrade = QtWidgets.QPushButton(self.centralwidget)
00182         self.but_regrade.setEnabled(False)
00183         self.but_regrade.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00184         self.but_regrade.setObjectName("but_regrade")
00185         self.verticalLayout_4.addWidget(self.but_regrade)
00186         self.checkB_input_pin_status = QtWidgets.QCheckBox(self.centralwidget)
00187         self.checkB_input_pin_status.setEnabled(False)
00188         self.checkB_input_pin_status.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00189         self.checkB_input_pin_status.setObjectName("checkB_input_pin_status")
00190         self.verticalLayout_4.addWidget(self.checkB_input_pin_status)
00191         self.checkB_output_pin_status = QtWidgets.QCheckBox(self.centralwidget)
00192         self.checkB_output_pin_status.setEnabled(False)
00193         self.checkB_output_pin_status.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00194         self.checkB_output_pin_status.setObjectName("checkB_output_pin_status")
00195         self.verticalLayout_4.addWidget(self.checkB_output_pin_status)
00196         self.horizontalLayout_7.addLayout(self.verticalLayout_4)
00197         self.verticalLayout_7.addLayout(self.horizontalLayout_7)
00198         self.horizontalLayout_10 = QtWidgets.QHBoxLayout()
00199         self.horizontalLayout_10.setSpacing(65)
00200         self.horizontalLayout_10.setObjectName("horizontalLayout_10")
00201         self.but_prev = QtWidgets.QPushButton(self.centralwidget)
00202         self.but_prev.setEnabled(False)
00203         self.but_prev.setMinimumSize(QtCore.QSize(60, 30))
00204         self.but_prev.setMaximumSize(QtCore.QSize(200, 16777215))
00205         self.but_prev.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00206         self.but_prev.setObjectName("but_prev")
00207         self.horizontalLayout_10.addWidget(self.but_prev)
00208         self.checkB_wrong = QtWidgets.QCheckBox(self.centralwidget)
00209         self.checkB_wrong.setEnabled(False)
00210         self.checkB_wrong.setMinimumSize(QtCore.QSize(80, 20))
00211         self.checkB_wrong.setMaximumSize(QtCore.QSize(75, 16777215))
00212         self.checkB_wrong.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00213         self.checkB_wrong.setObjectName("checkB_wrong")
00214         self.horizontalLayout_10.addWidget(self.checkB_wrong)
00215         self.but_reset = QtWidgets.QPushButton(self.centralwidget)
00216         self.but_reset.setEnabled(False)
00217         self.but_reset.setMinimumSize(QtCore.QSize(60, 20))
00218         self.but_reset.setMaximumSize(QtCore.QSize(90, 16777215))
00219         self.but_reset.setObjectName("but_reset")
00220         self.horizontalLayout_10.addWidget(self.but_reset)
00221         self.but_next = QtWidgets.QPushButton(self.centralwidget)
00222         self.but_next.setEnabled(False)
00223         self.but_next.setMinimumSize(QtCore.QSize(60, 30))
00224         self.but_next.setMaximumSize(QtCore.QSize(200, 16777215))
00225         self.but_next.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00226         self.but_next.setObjectName("but_next")
00227         self.horizontalLayout_10.addWidget(self.but_next)
00228         self.verticalLayout_7.addLayout(self.horizontalLayout_10)
00229         self.popular_answers = QtWidgets.QComboBox(self.centralwidget)
00230         self.popular_answers.setEnabled(False)
00231         self.popular_answers.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00232         self.popular_answers.setEditable(False)
00233         self.popular_answers.setCurrentText("")
00234         self.popular_answers.setObjectName("popular_answers")
00235         self.popular_answers.addItem("")
00236         self.popular_answers.setItemText(0, "")
00237         self.verticalLayout_7.addWidget(self.popular_answers)
00238         self.tabs_for_log_and_resp = QtWidgets.QTabWidget(self.centralwidget)
00239         self.tabs_for_log_and_resp.setEnabled(True)
00240         self.tabs_for_log_and_resp.setMinimumSize(QtCore.QSize(770, 30))
00241         self.tabs_for_log_and_resp.setMaximumSize(QtCore.QSize(20000, 3700))
```

```
00242         self.tabs_for_log_and_resp.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00243         self.tabs_for_log_and_resp.setTabShape(QtWidgets.QTabWidget.Rounded)
00244         self.tabs_for_log_and_resp.setObjectName("tabs_for_log_and_resp")
00245         self.response_tab = QtWidgets.QWidget()
00246         self.response_tab.setMinimumSize(QtCore.QSize(0, 180))
00247         self.response_tab.setMaximumSize(QtCore.QSize(16777215, 300))
00248         self.response_tab.setObjectName("response_tab")
00249         self.verticalLayout_9 = QtWidgets.QVBoxLayout(self.response_tab)
00250         self.verticalLayout_9.setObjectName("verticalLayout_9")
00251         self.splitter = QtWidgets.QSplitter(self.response_tab)
00252         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Expanding)
00253         sizePolicy.setHorizontalStretch(0)
00254         sizePolicy.setVerticalStretch(0)
00255         sizePolicy.setHeightForWidth(self.splitter.sizePolicy().hasHeightForWidth())
00256         self.splitter.setSizePolicy(sizePolicy)
00257         self.splitter.setOrientation(QtCore.Qt.Vertical)
00258         self.splitter.setObjectName("splitter")
00259         self.input_response_browser = QtWidgets.QPlainTextEdit(self.splitter)
00260         self.input_response_browser.setEnabled(True)
00261         self.input_response_browser.setMinimumSize(QtCore.QSize(0, 30))
00262         self.input_response_browser.setReadOnly(True)
00263         self.input_response_browser.setTextInteractionFlags(QtCore.Qt.TextSelectableByKeyboard|QtCore.Qt.TextSelectableByMouse)
00264         self.input_response_browser.setObjectName("input_response_browser")
00265         self.input_response_browser_user = BetterPlainTextEdit(self.splitter)
00266         self.input_response_browser_user.setEnabled(False)
00267         self.input_response_browser_user.setMinimumSize(QtCore.QSize(0, 30))
00268         self.input_response_browser_user.setObjectName("input_response_browser_user")
00269         self.verticalLayout_9.addWidget(self.splitter)
00270         self.tabs_for_log_and_resp.addTab(self.response_tab, "")
00271         self.tab_prev_resp = QtWidgets.QWidget()
00272         self.tab_prev_resp.setObjectName("tab_prev_resp")
00273         self.verticalLayout_5 = QtWidgets.QVBoxLayout(self.tab_prev_resp)
00274         self.verticalLayout_5.setObjectName("verticalLayout_5")
00275         self.input_prev_response = QtWidgets.QPlainTextEdit(self.tab_prev_resp)
00276         self.input_prev_response.setEnabled(True)
00277         self.input_prev_response.setTextInteractionFlags(QtCore.Qt.TextSelectableByKeyboard|QtCore.Qt.TextSelectableByMouse)
00278         self.input_prev_response.setObjectName("input_prev_response")
00279         self.verticalLayout_5.addWidget(self.input_prev_response)
00280         self.tabs_for_log_and_resp.addTab(self.tab_prev_resp, "")
00281         self.tab_message_to_all = QtWidgets.QWidget()
00282         self.tab_message_to_all.setObjectName("tab_message_to_all")
00283         self.verticalLayout_8 = QtWidgets.QVBoxLayout(self.tab_message_to_all)
00284         self.verticalLayout_8.setObjectName("verticalLayout_8")
00285         self.input_message_to_all = QtWidgets.QPlainTextEdit(self.tab_message_to_all)
00286         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Expanding)
00287         sizePolicy.setHorizontalStretch(0)
00288         sizePolicy.setVerticalStretch(0)
00289         sizePolicy.setHeightForWidth(self.input_message_to_all.sizePolicy().hasHeightForWidth())
00290         self.input_message_to_all.setSizePolicy(sizePolicy)
00291         self.input_message_to_all.setObjectName("input_message_to_all")
00292         self.verticalLayout_8.addWidget(self.input_message_to_all)
00293         self.tabs_for_log_and_resp.addTab(self.tab_message_to_all, "")
00294         self.log_tab = QtWidgets.QWidget()
00295         self.log_tab.setObjectName("log_tab")
00296         self.verticalLayout_6 = QtWidgets.QVBoxLayout(self.log_tab)
00297         self.verticalLayout_6.setObjectName("verticalLayout_6")
00298         self.input_log_browser = QtWidgets.QTextBrowser(self.log_tab)
00299         self.input_log_browser.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00300         self.input_log_browser.setObjectName("input_log_browser")
00301         self.verticalLayout_6.addWidget(self.input_log_browser)
00302         self.tabs_for_log_and_resp.addTab(self.log_tab, "")
00303         self.verticalLayout_7.addWidget(self.tabs_for_log_and_resp)
00304         self.horizontalLayout_11 = QtWidgets.QHBoxLayout()
00305         self.horizontalLayout_11.setObjectName("horizontalLayout_11")
00306         self.but_save_response = QtWidgets.QPushButton(self.centralwidget)
00307         self.but_save_response.setEnabled(False)
00308         self.but_save_response.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00309         self.but_save_response.setObjectName("but_save_response")
00310         self.horizontalLayout_11.addWidget(self.but_save_response)
00311         self.check_autosave = QtWidgets.QCheckBox(self.centralwidget)
00312         self.check_autosave.setEnabled(False)
00313         self.check_autosave.setObjectName("check_autosave")
00314         self.horizontalLayout_11.addWidget(self.check_autosave)
00315         self.manage_labs_but = QtWidgets.QPushButton(self.centralwidget)
00316         self.manage_labs_but.setEnabled(False)
00317         self.manage_labs_but.setObjectName("manage_labs_but")
00318         self.horizontalLayout_11.addWidget(self.manage_labs_but)
00319         self.set_style_checkbox = QtWidgets.QCheckBox(self.centralwidget)
00320         self.set_style_checkbox.setObjectName("set_style_checkbox")
00321         self.horizontalLayout_11.addWidget(self.set_style_checkbox)
00322         self.settings_but = QtWidgets.QToolButton(self.centralwidget)
```

```
00323         self.settings_but.setEnabled(True)
00324         self.settings_but.setObjectName("settings_but")
00325         self.horizontalLayout_11.addWidget(self.settings_but)
00326         self.but_save_all = QtWidgets.QPushButton(self.centralwidget)
00327         self.but_save_all.setEnabled(False)
00328         self.but_save_all.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00329         self.but_save_all.setObjectName("but_save_all")
00330         self.horizontalLayout_11.addWidget(self.but_save_all)
00331         self.but_create_report = QtWidgets.QPushButton(self.centralwidget)
00332         self.but_create_report.setEnabled(False)
00333         self.but_create_report.setObjectName("but_create_report")
00334         self.horizontalLayout_11.addWidget(self.but_create_report)
00335         self.verticalLayout_7.addLayout(self.horizontalLayout_11)
00336         self.progressBar = QtWidgets.QProgressBar(self.centralwidget)
00337         self.progressBar.setEnabled(True)
00338         self.progressBar.setAutoFillBackground(False)
00339         self.progressBar.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00340         self.progressBar.setProperty("value", 0)
00341         self.progressBar.setTextVisible(True)
00342         self.progressBar.setInvertedAppearance(False)
00343         self.progressBar.setObjectName("progressBar")
00344         self.verticalLayout_7.addWidget(self.progressBar)
00345         mainWindow.setCentralWidget(self.centralwidget)
00346 
00347         self.retranslateUi(mainWindow)
00348         self.tabs_for_log_and_resp.setCurrentIndex(0)
00349         QtCore.QMetaObject.connectSlotsByName(mainWindow)
00350 
00351     def retranslateUi(self, mainWindow):
00352         _translate = QtCore.QCoreApplication.translate
00353         mainWindow.setWindowTitle(_translate("mainWindow", "CSCI3130 grader"))
00354         self.input_file_location.setPlaceholderText(_translate("mainWindow", "Double click for path selection or paste|type path here"))
00355         self.but_file_open.setText(_translate("mainWindow", "Open"))
00356         self.but_begin.setText(_translate("mainWindow", "Begin"))
00357         self.label_from.setText(_translate("mainWindow", "From"))
00358         self.label_submitted.setText(_translate("mainWindow", "Submitted"))
00359         self.label_to.setText(_translate("mainWindow", "To"))
00360         self.label_current_id.setText(_translate("mainWindow", "current id"))
00361         self.label_attempt.setText(_translate("mainWindow", "attempt"))
00362         self.label_max_pos.setText(_translate("mainWindow", "lab max grade"))
00363         self.label_subtr.setText(_translate("mainWindow", "subtract"))
00364         self.label_final.setText(_translate("mainWindow", "final grade"))
00365         self.but_regrade.setText(_translate("mainWindow", "GRADE"))
00366         self.checkB_input_pin_status.setText(_translate("mainWindow", "Input direction"))
00367         self.checkB_output_pin_status.setText(_translate("mainWindow", "Output direction"))
00368         self.but_prev.setText(_translate("mainWindow", "prev"))
00369         self.checkB_wrong.setText(_translate("mainWindow", "WRONG"))
00370         self.but_reset.setText(_translate("mainWindow", "Reset"))
00371         self.but_next.setText(_translate("mainWindow", "next"))
00372         self.input_response_browser.setPlaceholderText(_translate("mainWindow", "Auto answer"))
00373         self.input_response_browser_user.setPlaceholderText(_translate("mainWindow", "User comment"))
00374         self.tabs_for_log_and_resp.setTabText(self.tabs_for_log_and_resp.indexOf(self.response_tab), _translate("mainWindow", "Response"))
00375         self.tabs_for_log_and_resp.setTabText(self.tabs_for_log_and_resp.indexOf(self.tab_prev_resp), _translate("mainWindow", "Previous
      Response"))
00376         self.tabs_for_log_and_resp.setTabText(self.tabs_for_log_and_resp.indexOf(self.tab_message_to_all), _translate("mainWindow", "Message to
      all"))
00377         self.tabs_for_log_and_resp.setTabText(self.tabs_for_log_and_resp.indexOf(self.log_tab), _translate("mainWindow", "Log"))
00378         self.but_save_response.setText(_translate("mainWindow", "save responce"))
00379         self.check_autosave.setText(_translate("mainWindow", "autosave"))
00380         self.manage_labs_but.setText(_translate("mainWindow", "Manage labs"))
00381         self.set_style_checkbox.setText(_translate("mainWindow", "style"))
00382         self.settings_but.setText(_translate("mainWindow", "Settings"))
00383         self.but_save_all.setText(_translate("mainWindow", "save all"))
00384         self.but_create_report.setText(_translate("mainWindow", "Create reports"))
00385         self.progressBar.setFormat(_translate("mainWindow", "%v/%m (%p%)"))
00386 
00387 from qt_class_improvements import BetterLineEdit, BetterPlainTextEdit
```

8.13 manage_labs.py File Reference

Classes

- class manage_labs.Ui_manage_labs

Namespaces

- manage_labs

8.14 manage_labs.py

```
00001 # -*- coding: utf-8 -*-
```

```
00002
00003 # Form implementation generated from reading ui file 'manage_labs.ui'
00004 #
00005 # Created by: PyQt5 UI code generator 5.12.dev1812231618
00006 #
00007 # WARNING! All changes made in this file will be lost!
00008
00009 from PyQt5 import QtCore, QtGui, QtWidgets
00010
00011 class Ui_manage_labs(object):
00012     def setupUi(self, manage_labs):
00013         manage_labs.setObjectName("manage_labs")
00014         manage_labs.resize(753, 90)
00015         manage_labs.setWindowFilePath("")
00016         self.verticalLayout = QtWidgets.QVBoxLayout(manage_labs)
00017         self.verticalLayout.setObjectName("verticalLayout")
00018         self.horizontalLayout = QtWidgets.QHBoxLayout()
00019         self.horizontalLayout.setObjectName("horizontalLayout")
00020         self.labs_select_comboBox = QtWidgets.QComboBox(manage_labs)
00021         self.labs_select_comboBox.setEnabled(False)
00022         self.labs_select_comboBox.setObjectName("labs_select_comboBox")
00023         self.horizontalLayout.addWidget(self.labs_select_comboBox)
00024         self.sync_but = QtWidgets.QPushButton(manage_labs)
00025         self.sync_but.setObjectName("sync_but")
00026         self.horizontalLayout.addWidget(self.sync_but)
00027         self.import_but = QtWidgets.QPushButton(manage_labs)
00028         self.import_but.setEnabled(False)
00029         self.import_but.setObjectName("import_but")
00030         self.horizontalLayout.addWidget(self.import_but)
00031         self.create_due_dates_but = QtWidgets.QPushButton(manage_labs)
00032         self.create_due_dates_but.setEnabled(False)
00033         self.create_due_dates_but.setObjectName("create_due_dates_but")
00034         self.horizontalLayout.addWidget(self.create_due_dates_but)
00035         self.export_but = QtWidgets.QPushButton(manage_labs)
00036         self.export_but.setEnabled(False)
00037         self.export_but.setObjectName("export_but")
00038         self.horizontalLayout.addWidget(self.export_but)
00039         self.verticalLayout.addLayout(self.horizontalLayout)
00040         self.status_bar = QtWidgets.QLineEdit(manage_labs)
00041         self.status_bar.setObjectName("status_bar")
00042         self.verticalLayout.addWidget(self.status_bar)
00043
00044         self.retranslateUi(manage_labs)
00045         QtCore.QMetaObject.connectSlotsByName(manage_labs)
00046
00047     def retranslateUi(self, manage_labs):
00048         _translate = QtCore.QCoreApplication.translate
00049         manage_labs.setWindowTitle(_translate("manage_labs", "Manage labs"))
00050         self.sync_but.setText(_translate("manage_labs", "Sync to local storage"))
00051         self.import_but.setText(_translate("manage_labs", "import labs"))
00052         self.create_due_dates_but.setText(_translate("manage_labs", "Create due dates"))
00053         self.export_but.setText(_translate("manage_labs", "Export pdfs"))
00054
```

8.15 mptest_mp.py File Reference

Namespaces

- mptest_mp

Functions

- def mptest_mp.f (x, y)

Variables

- list mptest_mp.b = [elem for elem in range(10)]
- int mptest_mp.c = 10
- mptest_mp.res = pool.starmap_async(f, ((elem, c) for elem in b))
- int mptest_mp.a = 55

8.16 mptest_mp.py

```
00001 import time, os, sys
00002 import multiprocessing as mp
00003 import random
00004
00005 def f(x, y):
00006     time.sleep(random.randint(1, 4))
```

```
00007     print(x, y)
00008     return x
00009
00010 if __name__ == '__main__':
00011     b = [elem for elem in range(10)]
00012     c = 10
00013
00014     with mp.Pool() as pool:
00015         res = pool.starmap_async(f, ((elem, c) for elem in b))
00016         # time.sleep(15)
00017         # a = res.get()
00018         res.wait()
00019         a = 55
00020         # for _ in res:
00021         #     pass
00022     # [print(elem) for elem in res._value]
00023     # [print(elem) for elem in res.get()]
00024     a = res.get(timeout=1)
00025     # res.get()
00026     a = 85
00027
00028     time.sleep(6)
00029     time.sleep(6)
00030     time.sleep(6)
```

8.17 qt_class_improvements.py File Reference

Classes

- class qt_class_improvements.BetterLineEdit
- class qt_class_improvements.BetterPlainTextEdit

Namespaces

- qt_class_improvements

8.18 qt_class_improvements.py

```
00001 """
00002 This file contains some improvements over standard QT5 UI elements.
00003 """
00004 from PyQt5 import QtWidgets, QtCore
00005
00006
00007 class BetterLineEdit(QtWidgets.QLineEdit):
00008     """
00009     QLineEdit extension: added mouse double click.
00010     I need this to open file browser by double click.
00011     """
00012     dclicked = QtCore.pyqtSignal()
00013
00014     def __init__(self, *args, **kwargs):
00015         QtWidgets.QLineEdit.__init__(self, *args, **kwargs)
00016
00017         self.installEventFilter(self)
00018
00019     def eventFilter(self, obj, event):
00020         """ typical way to add event handler """
00021         if event.type() == QtCore.QEvent.MouseButtonDblClick:
00022             self.dclicked.emit()
00023         return False
00024
00025
00026 class BetterPlainTextEdit(QtWidgets.QPlainTextEdit):
00027     """
00028     Overloaded QPlainTextEdit to track focus out.
00029     Needed to implement autosaving of user answer.
00030     """
00031     focus_lost = QtCore.pyqtSignal()
00032
00033     def __init__(self, *args, **kwargs):
00034         QtWidgets.QPlainTextEdit.__init__(self, *args, **kwargs)
00035
00036         self.installEventFilter(self)
00037
00038     def eventFilter(self, obj, event):
00039         """ typical way to add event handler """
00040         if event.type() == QtCore.QEvent.FocusOut:
00041             self.focus_lost.emit()
00042         return False
```

8.19 README.md File Reference

8.20 settings.py File Reference

Classes

- class settings.Ui_Settings

Namespaces

- settings

8.21 settings.py

```
00001 # -*- coding: utf-8 -*-
00002
00003 # Form implementation generated from reading ui file 'settings.ui'
00004 #
00005 # Created by: PyQt5 UI code generator 5.12.dev1812231618
00006 #
00007 # WARNING! All changes made in this file will be lost!
00008
00009 from PyQt5 import QtCore, QtGui, QtWidgets
00010
00011 class Ui_Settings(object):
00012     def setupUi(self, Settings):
00013         Settings.setObjectName("Settings")
00014         Settings.setEnabled(True)
00015         Settings.resize(800, 487)
00016         Settings.setMinimumSize(QtCore.QSize(600, 0))
00017         icon = QtGui.QIcon()
00018         icon.addPixmap(QtGui.QPixmap("os_linux_1.ico"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
00019         Settings.setWindowIcon(icon)
00020         Settings.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00021         self.verticalLayout = QtWidgets.QVBoxLayout(Settings)
00022         self.verticalLayout.setObjectName("verticalLayout")
00023         self.groupBox_db = QtWidgets.QGroupBox(Settings)
00024         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)
00025         sizePolicy.setHorizontalStretch(0)
00026         sizePolicy.setVerticalStretch(0)
00027         sizePolicy.setHeightForWidth(self.groupBox_db.sizePolicy().hasHeightForWidth())
00028         self.groupBox_db.setSizePolicy(sizePolicy)
00029         self.groupBox_db.setMinimumSize(QtCore.QSize(0, 0))
00030         self.groupBox_db.setAutoFillBackground(False)
00031         self.groupBox_db.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignTop)
00032         self.groupBox_db.setFlat(False)
00033         self.groupBox_db.setCheckable(False)
00034         self.groupBox_db.setObjectName("groupBox_db")
00035         self.formLayout = QtWidgets.QFormLayout(self.groupBox_db)
00036         self.formLayout.setObjectName("formLayout")
00037         self.label_settings_db = QtWidgets.QLabel(self.groupBox_db)
00038         self.label_settings_db.setMinimumSize(QtCore.QSize(110, 0))
00039         self.label_settings_db.setObjectName("label_settings_db")
00040         self.formLayout.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.label_settings_db)
00041         self.input_settings_db = QtWidgets.QLineEdit(self.groupBox_db)
00042         self.input_settings_db.setEnabled(False)
00043         self.input_settings_db.setMinimumSize(QtCore.QSize(550, 31))
00044         self.input_settings_db.setObjectName("input_settings_db")
00045         self.formLayout.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.input_settings_db)
00046         self.label_grades_db = QtWidgets.QLabel(self.groupBox_db)
00047         self.label_grades_db.setMinimumSize(QtCore.QSize(110, 0))
00048         self.label_grades_db.setObjectName("label_grades_db")
00049         self.formLayout.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.label_grades_db)
00050         self.input_grades_db = QtWidgets.QLineEdit(self.groupBox_db)
00051         self.input_grades_db.setMinimumSize(QtCore.QSize(550, 31))
00052         self.input_grades_db.setText("")
00053         self.input_grades_db.setObjectName("input_grades_db")
00054         self.formLayout.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.input_grades_db)
00055         self.verticalLayout.addWidget(self.groupBox_db)
00056         self.groupBox_user = QtWidgets.QGroupBox(Settings)
00057         self.groupBox_user.setEnabled(False)
00058         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)
00059         sizePolicy.setHorizontalStretch(0)
00060         sizePolicy.setVerticalStretch(0)
00061         sizePolicy.setHeightForWidth(self.groupBox_user.sizePolicy().hasHeightForWidth())
00062         self.groupBox_user.setSizePolicy(sizePolicy)
00063         self.groupBox_user.setMinimumSize(QtCore.QSize(0, 0))
00064         self.groupBox_user.setObjectName("groupBox_user")
00065         self.formLayout_2 = QtWidgets.QFormLayout(self.groupBox_user)
00066         self.formLayout_2.setObjectName("formLayout_2")
00067         self.label_logisim_path = QtWidgets.QLabel(self.groupBox_user)
```

```
00068         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)
00069         sizePolicy.setHorizontalStretch(0)
00070         sizePolicy.setVerticalStretch(0)
00071         sizePolicy.setHeightForWidth(self.label_logisim_path.sizePolicy().hasHeightForWidth())
00072         self.label_logisim_path.setSizePolicy(sizePolicy)
00073         self.label_logisim_path.setMinimumSize(QtCore.QSize(110, 0))
00074         self.label_logisim_path.setObjectName("label_logisim_path")
00075         self.formLayout_2.setWidget(0, QtWidgets.QFormLayout.LabelRole, self.label_logisim_path)
00076         self.input_logisim_path = QtWidgets.QLineEdit(self.groupBox_user)
00077         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Fixed)
00078         sizePolicy.setHorizontalStretch(0)
00079         sizePolicy.setVerticalStretch(0)
00080         sizePolicy.setHeightForWidth(self.input_logisim_path.sizePolicy().hasHeightForWidth())
00081         self.input_logisim_path.setSizePolicy(sizePolicy)
00082         self.input_logisim_path.setMinimumSize(QtCore.QSize(637, 31))
00083         self.input_logisim_path.setText("")
00084         self.input_logisim_path.setObjectName("input_logisim_path")
00085         self.formLayout_2.setWidget(0, QtWidgets.QFormLayout.FieldRole, self.input_logisim_path)
00086         self.label_local_stor = QtWidgets.QLabel(self.groupBox_user)
00087         self.label_local_stor.setMinimumSize(QtCore.QSize(110, 0))
00088         self.label_local_stor.setObjectName("label_local_stor")
00089         self.formLayout_2.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.label_local_stor)
00090         self.input_local_stor = QtWidgets.QLineEdit(self.groupBox_user)
00091         self.input_local_stor.setMinimumSize(QtCore.QSize(637, 31))
00092         self.input_local_stor.setText("")
00093         self.input_local_stor.setObjectName("input_local_stor")
00094         self.formLayout_2.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.input_local_stor)
00095         self.label_rem_stor = QtWidgets.QLabel(self.groupBox_user)
00096         self.label_rem_stor.setMinimumSize(QtCore.QSize(110, 0))
00097         self.label_rem_stor.setObjectName("label_rem_stor")
00098         self.formLayout_2.setWidget(2, QtWidgets.QFormLayout.LabelRole, self.label_rem_stor)
00099         self.input_rem_stor = QtWidgets.QLineEdit(self.groupBox_user)
00100         self.input_rem_stor.setMinimumSize(QtCore.QSize(637, 31))
00101         self.input_rem_stor.setInputMask("")
00102         self.input_rem_stor.setText("")
00103         self.input_rem_stor.setObjectName("input_rem_stor")
00104         self.formLayout_2.setWidget(2, QtWidgets.QFormLayout.FieldRole, self.input_rem_stor)
00105         self.verticalLayout.addWidget(self.groupBox_user)
00106         self.groupBox_local = QtWidgets.QGroupBox(Settings)
00107         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.MinimumExpanding)
00108         sizePolicy.setHorizontalStretch(0)
00109         sizePolicy.setVerticalStretch(0)
00110         sizePolicy.setHeightForWidth(self.groupBox_local.sizePolicy().hasHeightForWidth())
00111         self.groupBox_local.setSizePolicy(sizePolicy)
00112         self.groupBox_local.setMinimumSize(QtCore.QSize(0, 145))
00113         self.groupBox_local.setMaximumSize(QtCore.QSize(16777215, 300))
00114         self.groupBox_local.setFlat(False)
00115         self.groupBox_local.setCheckable(False)
00116         self.groupBox_local.setObjectName("groupBox_local")
00117         self.gridLayout = QtWidgets.QGridLayout(self.groupBox_local)
00118         self.gridLayout.setObjectName("gridLayout")
00119         self.spin_year = QtWidgets.QSpinBox(self.groupBox_local)
00120         self.spin_year.setEnabled(False)
00121         self.spin_year.setMinimumSize(QtCore.QSize(110, 31))
00122         self.spin_year.setMaximumSize(QtCore.QSize(110, 16777215))
00123         self.spin_year.setWrapping(True)
00124         self.spin_year.setReadOnly(False)
00125         self.spin_year.setButtonSymbols(QtWidgets.QAbstractSpinBox.PlusMinus)
00126         self.spin_year.setAccelerated(True)
00127         self.spin_year.setProperty("showGroupSeparator", False)
00128         self.spin_year.setMinimum(2012)
00129         self.spin_year.setMaximum(2026)
00130         self.spin_year.setProperty("value", 2018)
00131         self.spin_year.setObjectName("spin_year")
00132         self.gridLayout.addWidget(self.spin_year, 0, 1, 1, 1)
00133         self.label_grad_year = QtWidgets.QLabel(self.groupBox_local)
00134         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Fixed)
00135         sizePolicy.setHorizontalStretch(0)
00136         sizePolicy.setVerticalStretch(0)
00137         sizePolicy.setHeightForWidth(self.label_grad_year.sizePolicy().hasHeightForWidth())
00138         self.label_grad_year.setSizePolicy(sizePolicy)
00139         self.label_grad_year.setMinimumSize(QtCore.QSize(110, 31))
00140         self.label_grad_year.setMaximumSize(QtCore.QSize(110, 16777215))
00141         self.label_grad_year.setObjectName("label_grad_year")
00142         self.gridLayout.addWidget(self.label_grad_year, 0, 0, 1, 1)
00143         self.input_grader_name = QtWidgets.QLineEdit(self.groupBox_local)
00144         self.input_grader_name.setEnabled(False)
00145         self.input_grader_name.setMinimumSize(QtCore.QSize(110, 31))
00146         self.input_grader_name.setMaximumSize(QtCore.QSize(110, 16777215))
00147         self.input_grader_name.setObjectName("input_grader_name")
00148         self.gridLayout.addWidget(self.input_grader_name, 2, 1, 1, 1)
```

```
00149         self.label_semester = QtWidgets.QLabel(self.groupBox_local)
00150         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Fixed)
00151         sizePolicy.setHorizontalStretch(0)
00152         sizePolicy.setVerticalStretch(0)
00153         sizePolicy.setHeightForWidth(self.label_semester.sizePolicy().hasHeightForWidth())
00154         self.label_semester.setSizePolicy(sizePolicy)
00155         self.label_semester.setMinimumSize(QtCore.QSize(110, 31))
00156         self.label_semester.setMaximumSize(QtCore.QSize(110, 16777215))
00157         self.label_semester.setObjectName("label_semester")
00158         self.gridLayout.addWidget(self.label_semester, 0, 3, 1, 1)
00159         self.label_style = QtWidgets.QLabel(self.groupBox_local)
00160         self.label_style.setMinimumSize(QtCore.QSize(110, 31))
00161         self.label_style.setObjectName("label_style")
00162         self.gridLayout.addWidget(self.label_style, 1, 0, 1, 1)
00163         self.label_sync_comm = QtWidgets.QLabel(self.groupBox_local)
00164         self.label_sync_comm.setObjectName("label_sync_comm")
00165         self.gridLayout.addWidget(self.label_sync_comm, 2, 3, 1, 1)
00166         self.label_grader_name = QtWidgets.QLabel(self.groupBox_local)
00167         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Fixed)
00168         sizePolicy.setHorizontalStretch(0)
00169         sizePolicy.setVerticalStretch(0)
00170         sizePolicy.setHeightForWidth(self.label_grader_name.sizePolicy().hasHeightForWidth())
00171         self.label_grader_name.setSizePolicy(sizePolicy)
00172         self.label_grader_name.setMinimumSize(QtCore.QSize(110, 31))
00173         self.label_grader_name.setObjectName("label_grader_name")
00174         self.gridLayout.addWidget(self.label_grader_name, 2, 0, 1, 1)
00175         self.style_checkBox = QtWidgets.QCheckBox(self.groupBox_local)
00176         self.style_checkBox.setEnabled(False)
00177         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.MinimumExpanding, QtWidgets.QSizePolicy.Fixed)
00178         sizePolicy.setHorizontalStretch(0)
00179         sizePolicy.setVerticalStretch(0)
00180         sizePolicy.setHeightForWidth(self.style_checkBox.sizePolicy().hasHeightForWidth())
00181         self.style_checkBox.setSizePolicy(sizePolicy)
00182         self.style_checkBox.setMinimumSize(QtCore.QSize(0, 31))
00183         self.style_checkBox.setMaximumSize(QtCore.QSize(110, 16777215))
00184         self.style_checkBox.setLayoutDirection(QtCore.Qt.LeftToRight)
00185         self.style_checkBox.setText("")
00186         self.style_checkBox.setObjectName("style_checkBox")
00187         self.gridLayout.addWidget(self.style_checkBox, 1, 1, 1, 1)
00188         self.semester_comboBox = QtWidgets.QComboBox(self.groupBox_local)
00189         self.semester_comboBox.setEnabled(False)
00190         self.semester_comboBox.setMinimumSize(QtCore.QSize(110, 31))
00191         self.semester_comboBox.setMaximumSize(QtCore.QSize(110, 16777215))
00192         self.semester_comboBox.setMaxVisibleItems(3)
00193         self.semester_comboBox.setMaxCount(5)
00194         self.semester_comboBox.setObjectName("semester_comboBox")
00195         self.semester_comboBox.addItem("")
00196         self.semester_comboBox.addItem("")
00197         self.semester_comboBox.addItem("")
00198         self.gridLayout.addWidget(self.semester_comboBox, 0, 4, 1, 1)
00199         self.sync_command = QtWidgets.QLineEdit(self.groupBox_local)
00200         self.sync_command.setEnabled(False)
00201         self.sync_command.setMinimumSize(QtCore.QSize(0, 31))
00202         self.sync_command.setInputMask("")
00203         self.sync_command.setObjectName("sync_command")
00204         self.gridLayout.addWidget(self.sync_command, 2, 4, 1, 4)
00205         self.import_stuents_btn = QtWidgets.QPushButton(self.groupBox_local)
00206         self.import_stuents_btn.setObjectName("import_stuents_btn")
00207         self.gridLayout.addWidget(self.import_stuents_btn, 0, 6, 1, 1)
00208         spacerItem = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
00209         self.gridLayout.addItem(spacerItem, 0, 5, 1, 1)
00210         self.verticalLayout.addWidget(self.groupBox_local)
00211         self.buttonBox = QtWidgets.QDialogButtonBox(Settings)
00212         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Fixed)
00213         sizePolicy.setHorizontalStretch(0)
00214         sizePolicy.setVerticalStretch(0)
00215         sizePolicy.setHeightForWidth(self.buttonBox.sizePolicy().hasHeightForWidth())
00216         self.buttonBox.setSizePolicy(sizePolicy)
00217         self.buttonBox.setOrientation(QtCore.Qt.Horizontal)
00218
      self.buttonBox.setStandardButtons(QtWidgets.QDialogButtonBox.Apply|QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Ok|QtWidgets.QDialogButtonBox.Reset|Qt
00219         self.buttonBox.setObjectName("buttonBox")
00220         self.verticalLayout.addWidget(self.buttonBox)
00221
00222         self.retranslateUi(Settings)
00223         self.buttonBox.accepted.connect(Settings.accept)
00224         self.buttonBox.rejected.connect(Settings.reject)
00225         QtCore.QMetaObject.connectSlotsByName(Settings)
00226
00227     def retranslateUi(self, Settings):
00228         _translate = QtCore.QCoreApplication.translate
```

```
00229         Settings.setWindowTitle(_translate("Settings", "Settings"))
00230         self.groupBox_db.setTitle(_translate("Settings", "&Database paths:"))
00231         self.label_settings_db.setText(_translate("Settings", "Settings"))
00232         self.input_settings_db.setText(_translate("Settings", "./settings.sqlite3"))
00233         self.label_grades_db.setText(_translate("Settings", "Grades"))
00234         self.input_grades_db.setPlaceholderText(_translate("Settings", " /Documents/3130_labs/grades.sqlite3"))
00235         self.groupBox_user.setTitle(_translate("Settings", "User paths"))
00236         self.label_logisim_path.setText(_translate("Settings", "Logisim path"))
00237         self.input_logisim_path.setPlaceholderText(_translate("Settings", "path to logisim executable logisim.jar"))
00238         self.label_local_stor.setText(_translate("Settings", "Local lab storage"))
00239         self.input_local_stor.setPlaceholderText(_translate("Settings", "local directory that contains labs, reports, and other working
      files"))
00240         self.label_rem_stor.setText(_translate("Settings", "Remote lab storage"))
00241         self.input_rem_stor.setPlaceholderText(_translate("Settings", "sshfs mounted dir that points to submission directory on the remote
      server"))
00242         self.groupBox_local.setTitle(_translate("Settings", "&Local settings"))
00243         self.label_grad_year.setText(_translate("Settings", "Grading year"))
00244         self.label_semester.setText(_translate("Settings", "Grading semester"))
00245         self.label_style.setText(_translate("Settings", "Use styles"))
00246         self.label_sync_comm.setText(_translate("Settings", "Sync command"))
00247         self.label_grader_name.setText(_translate("Settings", "Grader name"))
00248         self.semester_comboBox.setItemText(0, _translate("Settings", "Spring"))
00249         self.semester_comboBox.setItemText(1, _translate("Settings", "Summer"))
00250         self.semester_comboBox.setItemText(2, _translate("Settings", "Fall"))
00251         self.sync_command.setPlaceholderText(_translate("Settings", "rsync -avz ? cp -v ? dd ... ?"))
00252         self.import_stuents_btn.setText(_translate("Settings", "Import_students"))
00253
```

8.22 simple_dialog.py File Reference

Classes

- class simple_dialog.Ui_Dialog

Namespaces

- simple_dialog

8.23 simple_dialog.py

```
00001 # -*- coding: utf-8 -*-
00002
00003 # Form implementation generated from reading ui file 'simple_dialog.ui'
00004 #
00005 # Created by: PyQt5 UI code generator 5.12.dev1812231618
00006 #
00007 # WARNING! All changes made in this file will be lost!
00008
00009 from PyQt5 import QtCore, QtGui, QtWidgets
00010
00011 class Ui_Dialog(object):
00012     def setupUi(self, Dialog):
00013         Dialog.setObjectName("Dialog")
00014         Dialog.resize(328, 76)
00015         icon = QtGui.QIcon()
00016         icon.addPixmap(QtGui.QPixmap("os_linux_1.ico"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
00017         Dialog.setWindowIcon(icon)
00018         Dialog.setLocale(QtCore.QLocale(QtCore.QLocale.English, QtCore.QLocale.UnitedStates))
00019         self.verticalLayout = QtWidgets.QVBoxLayout(Dialog)
00020         self.verticalLayout.setObjectName("verticalLayout")
00021         self.label_main_question = QtWidgets.QLabel(Dialog)
00022         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)
00023         sizePolicy.setHorizontalStretch(0)
00024         sizePolicy.setVerticalStretch(0)
00025         sizePolicy.setHeightForWidth(self.label_main_question.sizePolicy().hasHeightForWidth())
00026         self.label_main_question.setSizePolicy(sizePolicy)
00027         self.label_main_question.setAlignment(QtCore.Qt.AlignCenter)
00028         self.label_main_question.setObjectName("label_main_question")
00029         self.verticalLayout.addWidget(self.label_main_question)
00030         self.buttonBox_simple_dial = QtWidgets.QDialogButtonBox(Dialog)
00031         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)
00032         sizePolicy.setHorizontalStretch(0)
00033         sizePolicy.setVerticalStretch(0)
00034         sizePolicy.setHeightForWidth(self.buttonBox_simple_dial.sizePolicy().hasHeightForWidth())
00035         self.buttonBox_simple_dial.setSizePolicy(sizePolicy)
00036         self.buttonBox_simple_dial.setOrientation(QtCore.Qt.Horizontal)
00037         self.buttonBox_simple_dial.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets.QDialogButtonBox.Ok)
00038         self.buttonBox_simple_dial.setObjectName("buttonBox_simple_dial")
00039         self.verticalLayout.addWidget(self.buttonBox_simple_dial)
00040
```

```
00041         self.retranslateUi(Dialog)
00042         self.buttonBox_simple_dial.accepted.connect(Dialog.accept)
00043         self.buttonBox_simple_dial.rejected.connect(Dialog.reject)
00044         QtCore.QMetaObject.connectSlotsByName(Dialog)
00045 
00046     def retranslateUi(self, Dialog):
00047         _translate = QtCore.QCoreApplication.translate
00048         Dialog.setWindowTitle(_translate("Dialog", "Create database ?"))
00049         self.label_main_question.setText(_translate("Dialog", "Database will be created.  Confirm.."))
00050 
```

Index