

❗ **Your top-level group ADVANCE is over the 5 user limit and has been placed in a read-only state.**

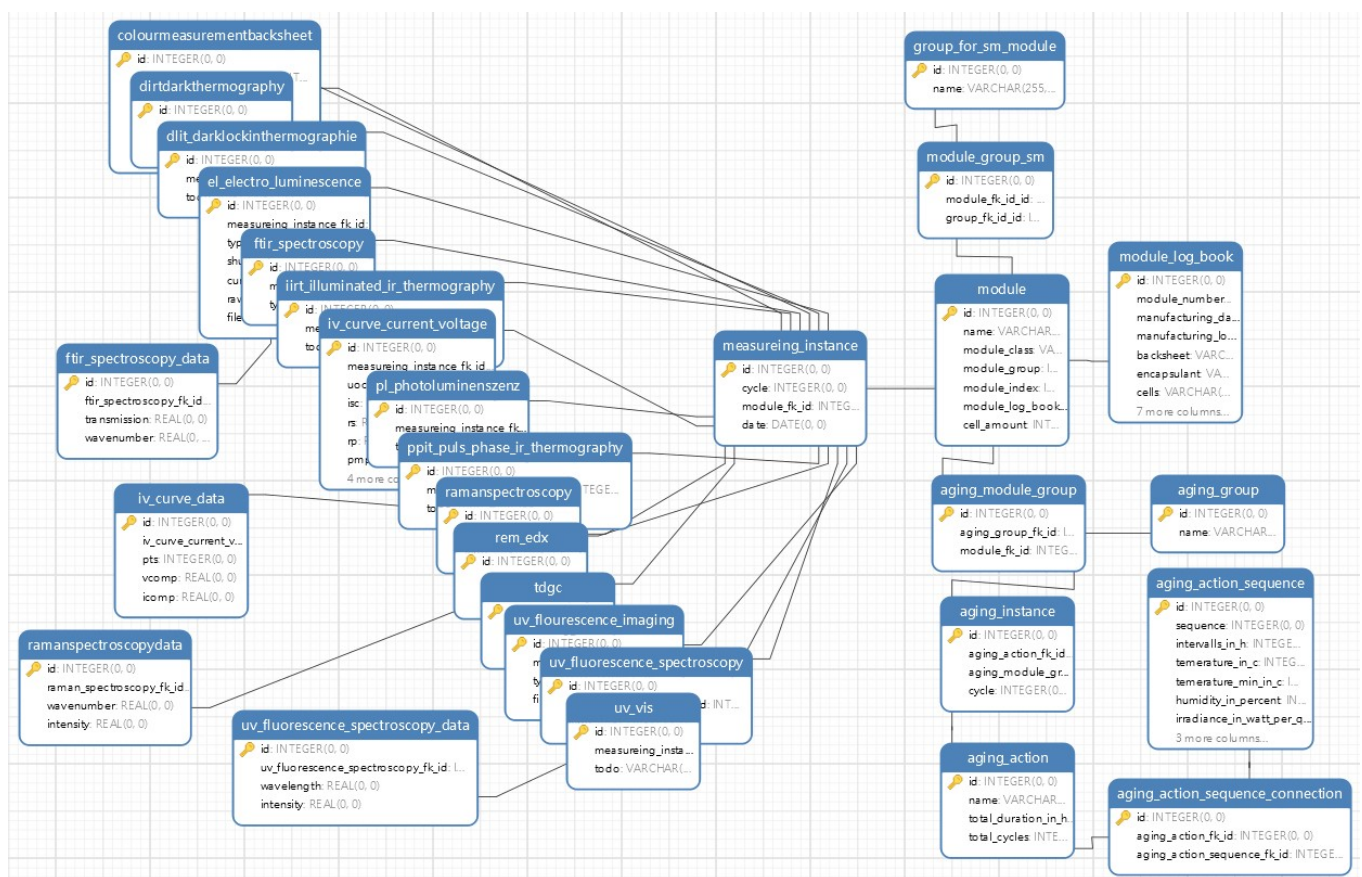
To remove the [read-only](#) state and regain write access, ask your top-level group owner(s) to reduce the number of users in your top-level group to 5 users or less, or to upgrade to a paid tier which do not have user limits.

queries

Last edited by [Karl Knoebl](#) 3 years ago

- [DB Tables Figure](#)
- [MPP over aging cycles for given module](#)
- [Color values over aging cycles for given module group](#)
- [IR Spectra \(backsheet\) over aging cycles for given module](#)
- [Export EL pictures over aging cycles for given module](#)
- [UVFSP \(Encapsulant\) over aging cycles for given module - Example](#)

DB Tables Figure



MPP over aging cycles for given module

```
def get_ageing(module_name):
    """
    :param module_name: eg: INFS003_031
    :return:
    query the aging action name from the database
    """
    # variant 1 ++++++
    ++++++
    aging_module_group = (Aging_Module_Group
                          .select()
                          .join(Module)
                          .where(Module.name == module_name)
                          .objects()
                          .get())
```

```

    )

    a_instances = (Aging_Instance
        .select()
        .where(Aging_Instance.aging_module_group_fk == aging_module_group.aging_group_fk_id)
        .order_by(Aging_Instance.cycle.asc())
        .objects()
    )

    action = (Aging_Action
        .select()
    # aging action should not change that is why simply the last element is picked -> [-1]
        .where(Aging_Action.id == a_instances[-1].aging_action_fk)
        .get())
    print(action.name)
    return action.name

# variant 2 ++++++
++++
    aging_action = (Aging_Action
        .select(Aging_Action.name)
        .join(Aging_Instance, on=(Aging_Action.id == Aging_Instance.aging_action_fk))
        .join(Aging_Module_Group, on=(Aging_Instance.aging_module_group_fk == Aging_Module_Group.aging_group_fk_id))
        .join(Module, on=(Aging_Module_Group.module_fk == Module.id))
        .where(Module.name == module_name)
        .get()
    )

    print(aging_action.name)
    return aging_action.name

def getMPP(module_name):
    """
    :param module_name: eg: INFS003_031
    :return: a list of cycles and according mpps ordered by the cycle
    """
    pmpp_list = []
    cycle_list = []
    measurement_instances = (Measureing_Instance
        .select()
        .join(Module)
        .where(Module.name == module_name)
        .order_by(Measureing_Instance.cycle))

    for measurement_instance in measurement_instances:
        iv_curve = IV_Curve_Current_Voltage.select() \
            .where(IV_Curve_Current_Voltage.measureing_instance_fk == measurement_instance).get()
        pmpp_list.append(iv_curve.pmpp)
        cycle_list.append(measurement_instance.cycle)
    return cycle_list, pmpp_list

def printMPP():
    """
    calls the database
    create a graph from cycles and MPPs
    :return:
    """
    cycles_19, mpps_19 = getMPP('INFS003_031')
    cycles_20, mpps_20 = getMPP('INFS003_032')
    cycles_21, mpps_21 = getMPP('INFS003_033')

    fig, ax = plt.subplots()

```

```

fig.canvas.set_window_title('MPP')
ageing = get_ageing('INFS003_031')
ax.set_title('MPP (' + ageing + ')')

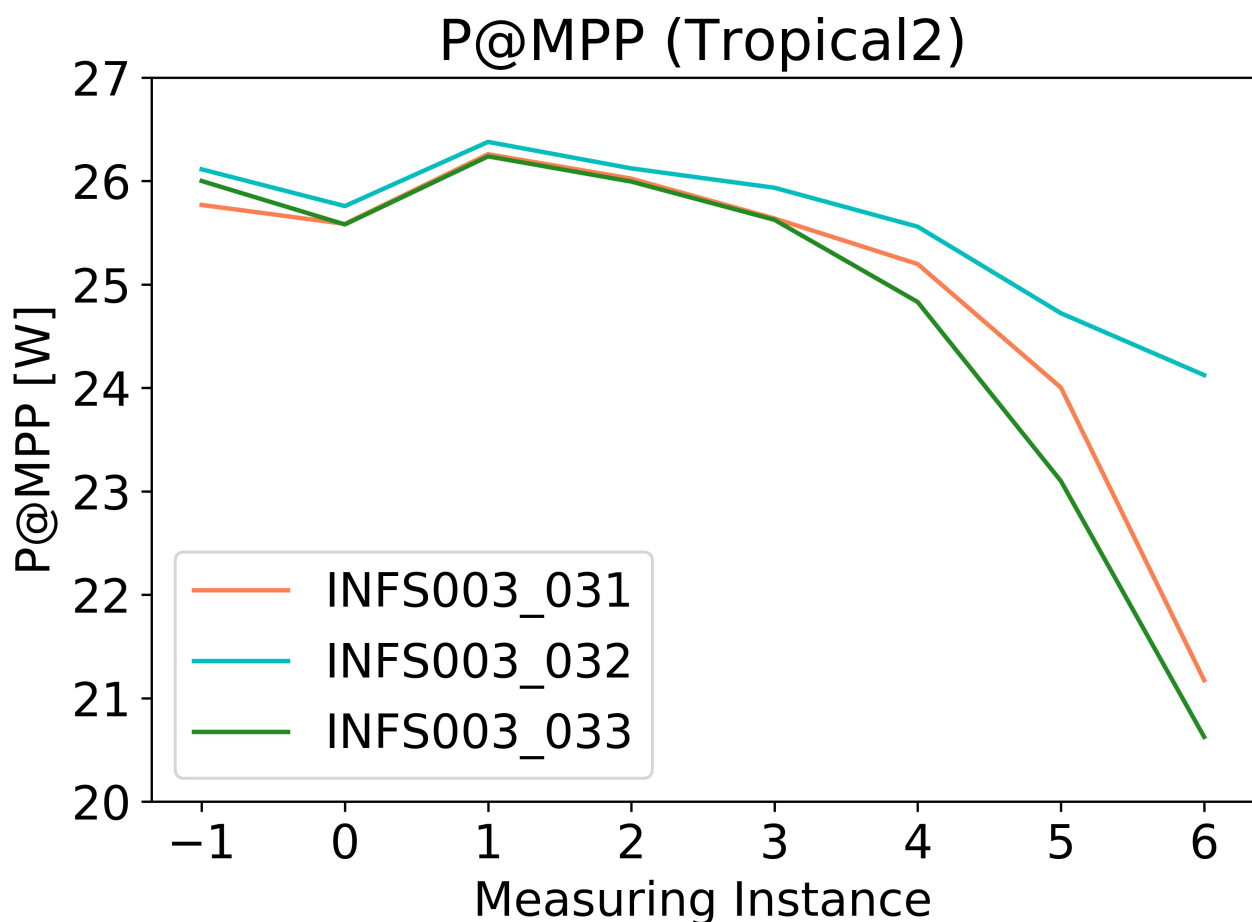
ax.plot(cycles_19, mpps_19, 'b', label='INFS003_031')
ax.plot(cycles_20, mpps_20, 'k', label='INFS003_032')
ax.plot(cycles_21, mpps_21, 'r', label='INFS003_033')
plt.ylabel('MPP [W]')
plt.xlabel('Measurement cycle')
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
ax.legend()
plt.show()

```

get_MPP() returns cycle numbers and mpp values for given module

sequence below configures and shows a plot

...set_major_locator... configures x-axis to be scaled as integers



Color values over aging cycles for given module group

```

def get_Color(module_name):
    """
    :param module_name: eg: INFS003_019
    :return:
    query deltaE values and cycles
    """
    delta_Es = []
    cycles = []
    measurement_instances = (Measureing_Instance
                             .select(Measureing_Instance, ColourMeasurementBacksheet)

```

```

        .join(Module)
        .join(ColourMeasurementBacksheet,
              on=(ColourMeasurementBacksheet.measureing_instance_fk == Measureing_Instance
e.id))

        .where(Module.name == module_name)
        .order_by(Measureing_Instance.cycle.desc())
        .objects())
for measurement_instance in measurement_instances:
    delta_Es.append(measurement_instance.delta_E)
    cycles.append(measurement_instance.cycle)
return delta_Es, cycles

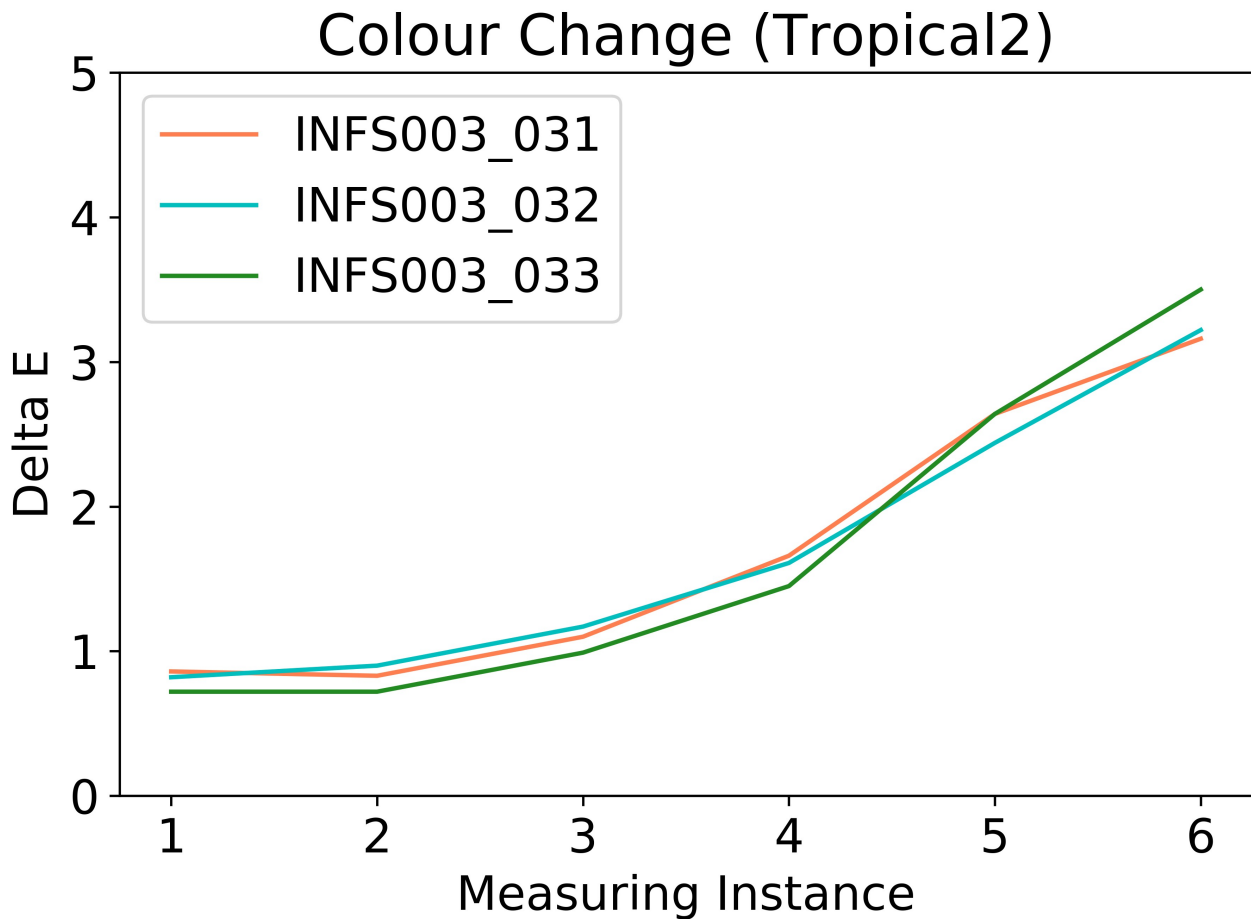
def printColorVals():
    """
    :return:
    calls the database
    create a graph from cycles and deltaE
    """
    color_vals_001, cycle_001 = get_Color('INFS003_031')
    color_vals_002, cycle_002 = get_Color('INFS003_032')
    color_vals_003, cycle_003 = get_Color('INFS003_033')

    fig, ax = plt.subplots()
    fig.canvas.set_window_title('delta E')
    ax.set_title('delta E (ageing acc. to scheme moderate 2)')
    ax.plot(cycle_001, color_vals_001, 'b', label='INFS003_031')
    ax.plot(cycle_002, color_vals_002, 'k', label='INFS003_032')
    ax.plot(cycle_003, color_vals_003, 'r', label='INFS003_033')

    plt.ylabel('delta E')
    plt.xlabel('Measurement cycle')
    ax.xaxis.set_major_locator(MaxNLocator(integer=True))
    ax.legend()
    plt.show()

```

get_Color() returns cycle numbers and colorvalues



IR Spectra (backsheet) over aging cycles for given module

```
class Curve:
    """
        Helper class to return a concise object
    """
    def __init__(self):
self.cycle = 0
        self.y_transmission = []
self.x_wavenumbers = []

def fetch_ir_bs_curves(module_name):
    """
        :param module_name: eg: INFS003_033
        :return:
        query the ir curves for a given module
    """
    curves = []
    measurement_instances = (Measureing_Instance
                            .select()
                            .join(Module)
                            .where(Module.name == module_name)
                            .order_by(Measureing_Instance.cycle.desc()))

    for instance in measurement_instances:
        ftir_inst = FTIR_Spectroscopy.select().where(FTIR_Spectroscopy.measureing_instance_fk == instance)
    if ftir_inst:
        ftir_data = (FTIR_Spectroscopy_Data
```

```

        .select()
        .where(FTIR_Spectroscopy_Data.ftir_spectroscopy_fk == ftir_inst)
        .order_by(FTIR_Spectroscopy_Data.wavenumber.asc())
        .execute()

    c = Curve()
    c.cycle = instance.cycle

for record in ftir_data:
    c.x_wavenumbers.append(record.wavenumber)
    c.y_transmission.append(record.transmission)

    curves.append(c)

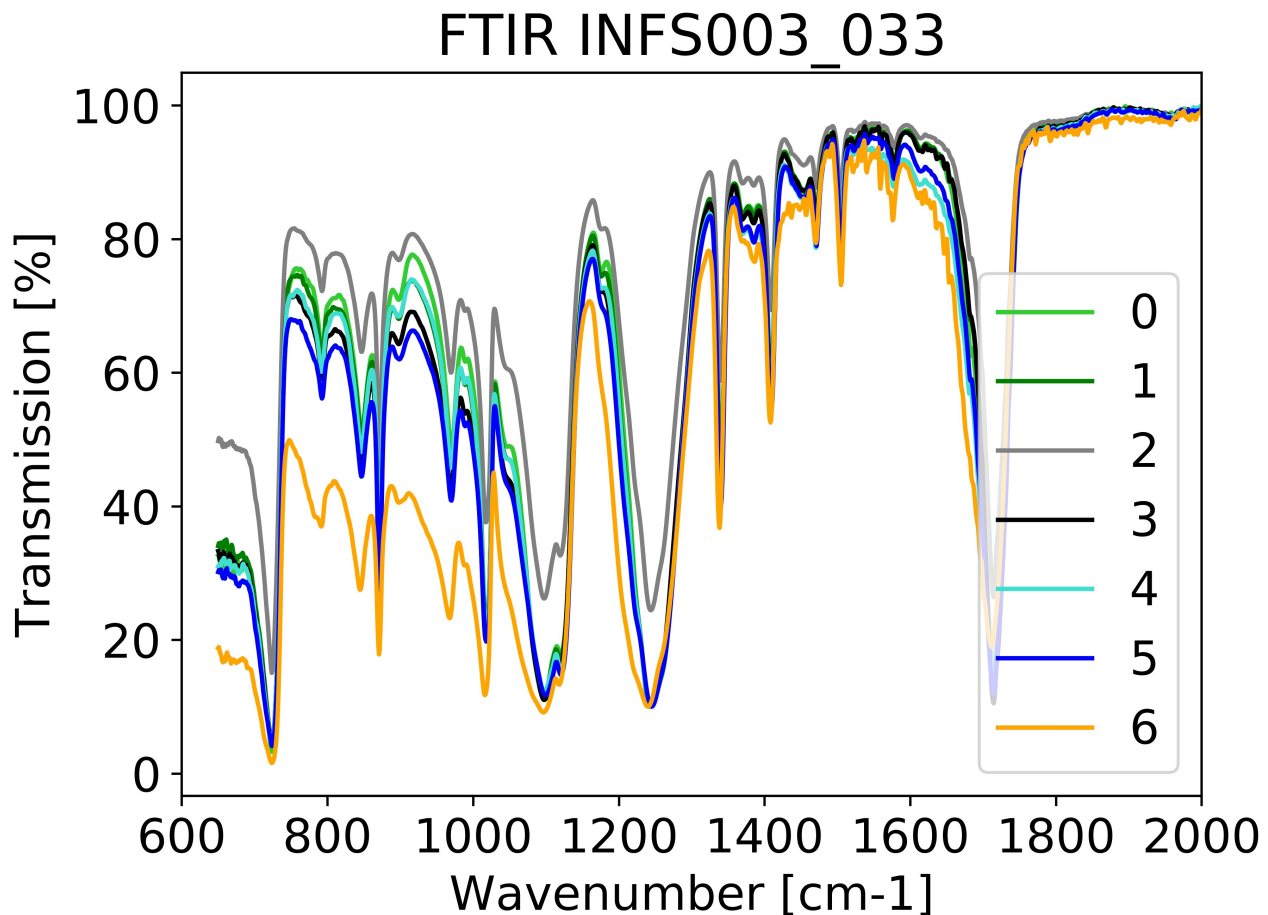
return curves

def show_ir_bs_curves(module_name):
    """
    :param module_name: eg: INFS003_033
    :return:
    create graphs for every module
    """
    curves = fetch_ir_bs_curves(module_name)
    color_list = ["b", "g", "r", "c", "m", "y", "k", "b", "b", "g", "r", "c", "m", "y", "k", "b"]

    fig, ax = plt.subplots()
    fig.canvas.set_window_title('FTIR')
    ax.set_title('FTIR ' + module_name)
    for curve, color in zip(curves, color_list):
        ax.plot(curve.x_wavenumbers, curve.y_transmission, color, label=str(curve.cycle))
    plt.ylabel('Transmission [%]')
    plt.xlabel('Wavenumber [cm-1]')
    ax.legend()
    plt.show()

def print_ir_bs_curves():
    show_ir_bs_curves('INFS003_033')
```

fetch_ir_bs_curves() returns a complete curve with x_wavenumber and y_transmission arrays, which is plotted in the sequence below. Each curve is colored according to color_list



Export EL pictures over aging cycles for given module

```
def copy_over(filename, new_filename):
    filename_with_path = path.join(db_directory, filename)
    new_filename_with_path = path.join(out_directory, new_filename)
    copy2(filename_with_path, new_filename_with_path)

def get_pics_for_module(module_name):
    """
    :param module_name: eg: INFS003_019
    :return:
    query db for pictures of the given module and copy that file over to a specified location
    also the name is changed to reflect the needed information.
    """
    measurement_instances = (Measureing_Instance
                             .select(Measureing_Instance.cycle, EL_Electro_Luminescence.filename)
                             .join(Module)
                             .join(EL_Electro_Luminescence,
                                   on=(EL_Electro_Luminescence.measureing_instance_fk == Measureing_Instance.i
                                       d))
                             .where(Module.name == module_name)
                             .order_by(Measureing_Instance.cycle.asc())
                             .naive())

    for measurement_instance in measurement_instances:
        print(measurement_instance.cycle, " ", measurement_instance.filename)
        el_part, filename = path.split(measurement_instance.filename)
```

```
hashcode, ext = path.splitext(filename)
new_filename = path.join(el_part, module_name + '_' + (str(measurement_instance.cycle) + ext))
copy_over(measurement_instance.filename, new_filename)
```

```
def exportPictures():
    create_dir(path.join(out_directory))
    create_dir(path.join(out_directory, 'EL'))
    get_pics_for_module('INFS003_010')
    get_pics_for_module('INFS003_011')
    get_pics_for_module('INFS003_012')
```

UVFSP (Encapsulant) over aging cycles for given module - Example

