PyTimber Web Application and GIF++ Web Site Updates

Katarina Milačić
Supervised by: Blerina Gkotse, Federico Ravotti, Martin R. Jäkel

August 16, 2019

1 Introduction

The Gamma Irradiation Facility (GIF++) uses the PyTimber web application [1] for querying data from DIM and the CERN Logging Service (CALS) through `pydim` and `pytimber` APIs. This project is related to updating the Pytimber web application, resolving existing problems, and improving the user experience on the GIF++ user website.

2 Description of the changes

2.1 PyTimber

There were few issues using the PyTimber web application. First, the Python library `matplotlib` was used to produce plot images and, if after getting one plot image new variables were selected by the user, the old variables were used again so the output wasn't the one requested. In addition, only after getting the plot image the user would have the option to choose between two other plots that were using JavaScript libraries, which wasn't intuitive. Finally, the exported CSV files were not in the desired format – dates were presented as timestamps, and rows were not well formatted, since columns were `timestamp_1, var_1, timestamp_2, var_2`, etc.

In the following subsections, I describe the updates that have been implemented in the web application.

2.1.1 Highcharts

In the new version, visual changes have been made – the timber image and footer are removed, header is restyled. Next, existing options for plotting graphs were replaced with single one - JavaScript library Highcharts [2]. This library offers zooming option along both x and y-axes, while before that was only possible along x-axis. Furthermore, an additional functionality for representing y-axes was implemented. In the previous version, selected variables that have different units were all shown on the same y-axis, which was not intuitive and easily comprehensible. For example, one variable might have range from 0 to 1, whereas another one from 0 to 1000; y-axis would be scaled according to the larger range of values, and then it would be hard to follow changes in values for line graph with range from 0 to 1. In the new version, each unit has its own y-axis, and therefore scaling is done better. Finally, the "boosting method" (Highcharts option) is enabled to render large number of points as quickly as possible and bypass some Highcharts features such as animation.

With Highcharts a user can export graphs as PNG or JPEG images, as PDF documents, CSV files or SVG vector images. There is also an option to view data points in a table on a web page - *view data table* button. This table has also the format of a CSV file. For larger number of points, it could take a while until the table is rendered on the page.

2.1.2 CSV file

The CSV file format has been changed and now it is more user friendly. The timestamp is represented in datetime format and is stored in one column. Now columns are `timestamp, var_1, var_2` etc. If a variable was not measured in a certain datetime, the field in the column of that variable is empty. As detailed above, the Highcharts library also offers option for downloading CSV files, but for larger number of points it will work quite slow or it will crash. Therefore, it is recommended to use the *download CSV* button for larger data since the CSV file is created through a view function in Django and it is much faster.

2.1.3 Timezone

In the project, I also noticed problems with the time of retrieved data. Data had correct timestamps, but the time interval for which data was

requested was actually shifted for two hours. The reason for this is that the local time that is displayed on the page was treated in the background as it was the UTC time, instead of being first converted from local to UTC time. Now this is solved: when you submit data, additional information is submitted – timezone, which is read from browser with `Intl.DateTimeFormat().resolvedOptions().timeZone` in JavaScript. This timezone is used to convert from local time to UTC in Python. Data is always requested in UTC time, then when the data is retrieved, this same timezone is used to convert to local time. In order to convert correctly to local time, Highcharts requires two Javascript files - `moment.min.js` and `moment-timezone-with-data-1970-2030.min.js` [4].

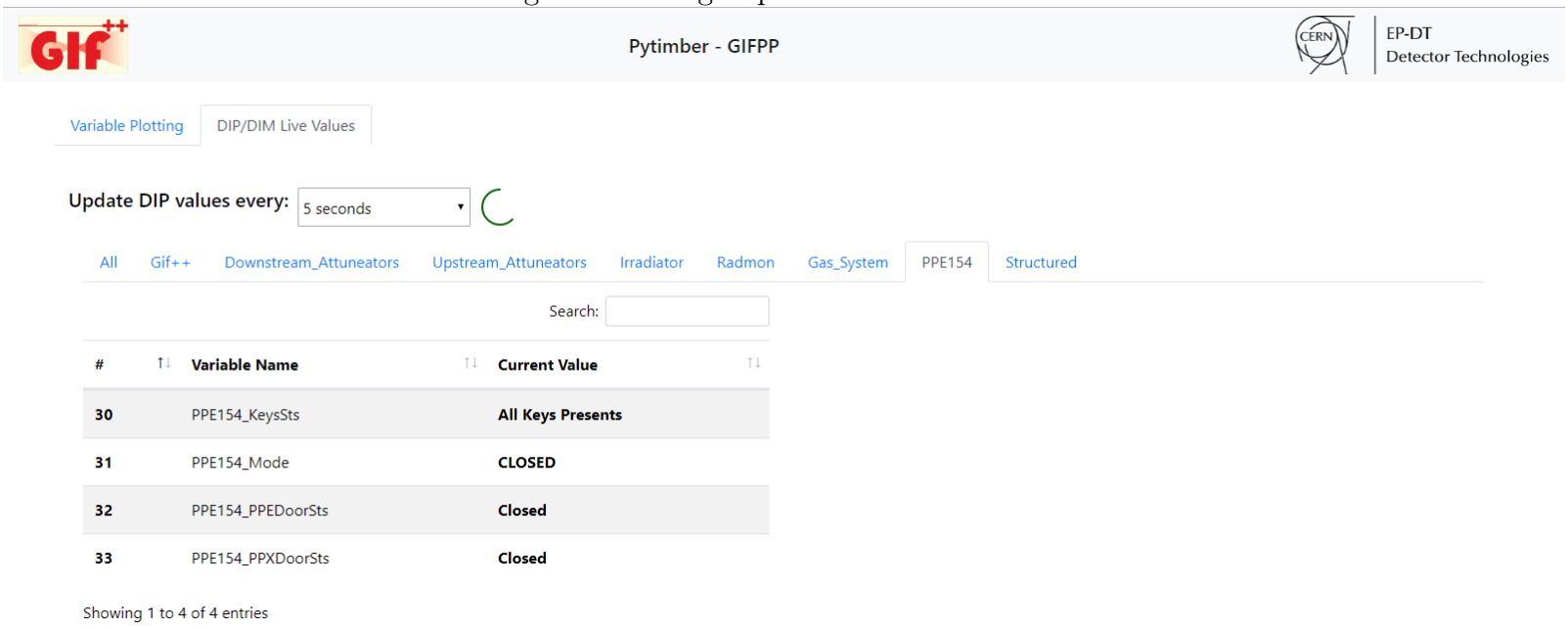
2.1.4 DIP/DIM values

Currently two different types of DIP/DIM variables exist:

1) those with a single value
2) those with multiple values (see Figure 3).

As for the single value variables, a new group of variables was added – PPE154 (Figure 1).

Figure 1: New group of variables - PPE154



As for the multiple value variables, two were added - PXAN1646 and PXAN1647 (Figure 2).

Figure 2: New structured variables

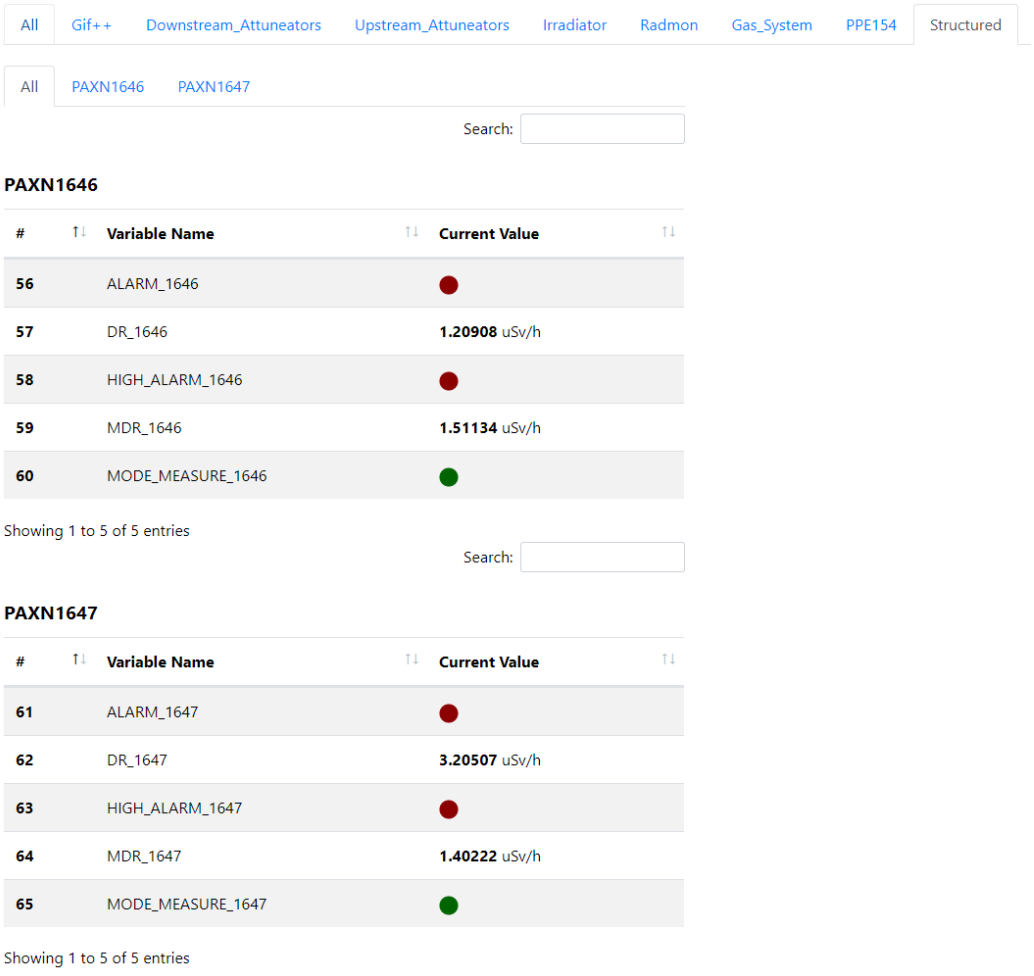
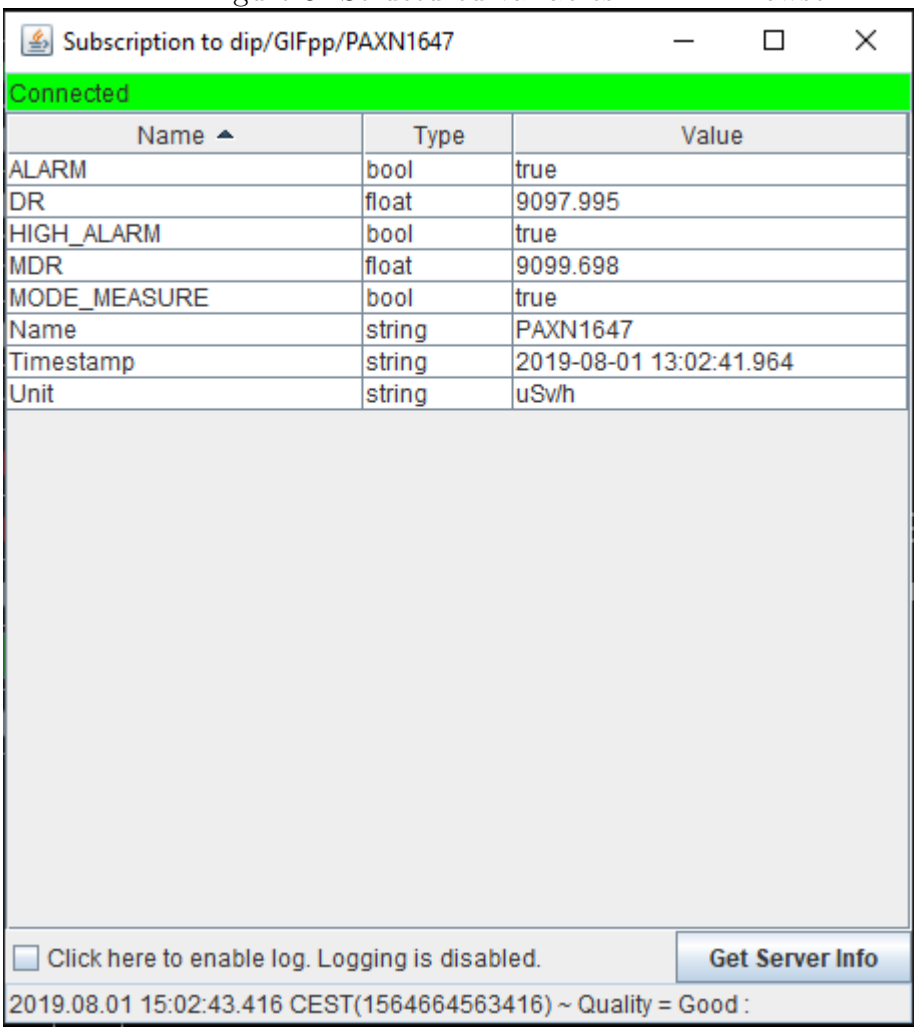


Figure 3: Structured variables in DIP Browser



Until now, only single value variables were handled in the PyTimber web application, and it was necessary to implement the handling of multiple value variables. For processing these multiple value variables (Figures 2 and 3), a new configuration file `gifpp_struct.csv` (Figure 4) was added to `epdtdi_timber/static` folder. In the same folder you can find the configuration file for single value variables - `gifpp_dip.csv`. New function `dip_struct_listen` (`epdtdi_web_timber/utils.py`) was implemented in order to keep the handling of the old and new type separately.

Each structured variable has a configuration format as shown in Fig. 5.

Figure 4: Configuration file for structured variables (`gifpp_struct.csv`)

FORMAT: [DIP Variable name - must matched e	[Unit of Variable]	[Variable datatype]	[Number of significant figures	[Lower threshold	[Upper threshold
# STRUCT PATH=dip/GIFpp/PAXN1646/DipData					
ALARM_1646		boolean			
DR_1646	uSv/h	float			
HIGH_ALARM_1646		boolean			
MDR_1646	uSv/h	float			
MODE_MEASURE_1646		boolean			
# END STRUCT					
# STRUCT PATH=dip/GIFpp/PAXN1647/DipData					
ALARM_1647		boolean			
DR_1647	uSv/h	float			
HIGH_ALARM_1647		boolean			
MDR_1647	uSv/h	float			
MODE_MEASURE_1647		boolean			
# END STRUCT					

Figure 5: Structure format in configuration file

```
# STRUCT PATH=...
val_1
val_2
...
val_n
# END STRUCT
```

In `"# STRUCT PATH="` row remember to always add *DipData* at the end of the path from DIP Browser. For example, if path to the variable is *dip/GIFpp/PAXN1647* (see header in Figure 3), then in configuration file it should be written *dip/GIFpp/PAXN1647/DipData*. This also stands for single value variables.

The columns in configuration file for structured variables are the same as in the configuration file for single value variables, in order to keep the same format and make use of existing code for reading columns and representing values with HTML and CSS.

Function `dip_struct_listen` reads `gifpp_struct.csv` file and produces the following dictionary:

```
OrderedDict(
  [("PAXN1646",
    {
      "dip_path":"dip/GIFpp/PAXN1646/DipData",
      "dip_sub":"...",
      "vars":
        {
          "ALARM_1646": {
            "dip_path":"ALARM_1646",
            "dip_unit": "",
            "dip_format": "",
            "dip_float_display": "",
            "dip_sig_fig": "",
            "dip_lower_limit": "",
            "dip_upper_limit": "",
            "dip_group": "PAXN1646",
          },
          "DR_1646": {...},
            ...
          "MODE_MEASURE_1646": {...}
        }
    }
),
   ("PAXN1647",{...})
])
```

For single value variables this dictionary would look like:

```
OrderedDict(
  [("Atmospheric_Pressure",
    {
      "dip_path":"dip/GIFpp/Atmospheric_Pressure/DipData",
      "dip_sub":"...",
      "dip_unit": "mbar",
      "dip_format": "",
      "dip_float_display": "",
      "dip_sig_fig": "",
      "dip_lower_limit": "",
      "dip_upper_limit": "",
```

```
        "dip_group": "Gif++"
    }
), ... ,
    ("Temp_Outside_Bunker",{...})
])
```

This dictionary for multiple value variables is then handled in `epdtdi_timber/utils.py` by function `getStructDipVals`. This is adjusted version of `getDipVals` function which handles dictionary for single value variables.

You can see the output of `getStructDipVals` function below, and this is the data that is finally forwarded to HTML for rendering:

```
{
  "ALARM_1646": {
    "value": False,
    "unit": "",
    "format": "",
    "flo_display": "",
    "sig_fig": "",
    "lower_limit": "",
    "upper_limit": "",
    "group": "PAXN1646",
  }, ... ,
  "MODE_MEASURE_1646": {...},
  "ALARM_1647": {...},
    ... ,
  "MODE_MEASURE_1647": {...}
}
```

The dictionary for single value variables would look exactly the same (see below). This is done on purpose so code in HTML could be used for both single and multiple value variables.

```
{
  "Atmospheric_Pressure": {
    "value":  ,
    "unit": "mbar",
    "format": "",
    "flo_display": "",
```

```
        "sig_fig": "",
        "lower_limit": "",
        "upper_limit": "",
        "group": "Gif++",
    }, ... ,
    "Temp_Outside_Bunker": {...}
}
```

In Figure 4, you can see that two structures have the values with the same names, only numbers are different - 1646 or 1647. You can name these values however you want; they don't need to have the same name as in DIP Browser (Figure 3). These numbers (1646 and 1647) were added on purpose, since these structures are transformed to dictionary, and if there are different values with the same name, it would mean that different elements are being added to the dictionary but using the same key, which is not possible. Therefore it is important for values to have different names even if they are in different structures.

When reading values from DIP Browser, it is mandatory to have variable datatype field. Each type has corresponding string that describes it (also see `pydim` documentation [3]). For example, one float value is described with *"F:1"*. With structured variables, we are dealing with multiple values. Since in Figure 4 there are datatypes boolean - float - boolean - float - boolean, the description string will look like *"C:1;F:1;C:1;F:1;C:1;"*. Function `dip_struct_listen` will take these datatypes and then form description string. Note that values from structured variable have to be written in the same order as in DIP Browser (Figure 3) since this is simply the way `pydim` API [3] was written. It is not possible to skip variables from the beginning or in the middle, but you can ignore variables at the end and simply not write them in configuration file. As shown in Figure 3, the Name, Timestamp and Unit are not in configuration file (Figure 4) and the application is properly functioning.

In case that Name, Timestamp, and Unit are required, it is necessary to specify the number of characters for each value. For single value variables there is no need to specify length of string, but for structured variables it is necessary to write the number of characters in 'Number of significant figures' column.

Also, new types have been configured - long, short and double. Attention should be paid to always write the same type in the configuration file as it

is in DIP Browser.

If the structure format (Figure 5) is used, also single value variables can be added to configuration file for structured variables (`gifpp_struct.csv`, Figure 4) and it will work, as well.

2.2 GIF++ Operation Web Page

At the beginning a questionnaire was prepared to get user feedback about the usability of the old GIF++ web site [6] (Figure 6), to find out which pages were more/less used, which functionalities should be added/removed/more emphasized, etc. The results of this questionnaire (see Figure 8) were used to make some improvements and modifications to web site.

The GIF++ Operation Page now has different style (Figure 7). Structure of the web site remained the same, and Semantic UI framework [5] was used to give a similar look of the PS-IRRAD web site [7].

Figure 6: Previous GIF++ Operation Page

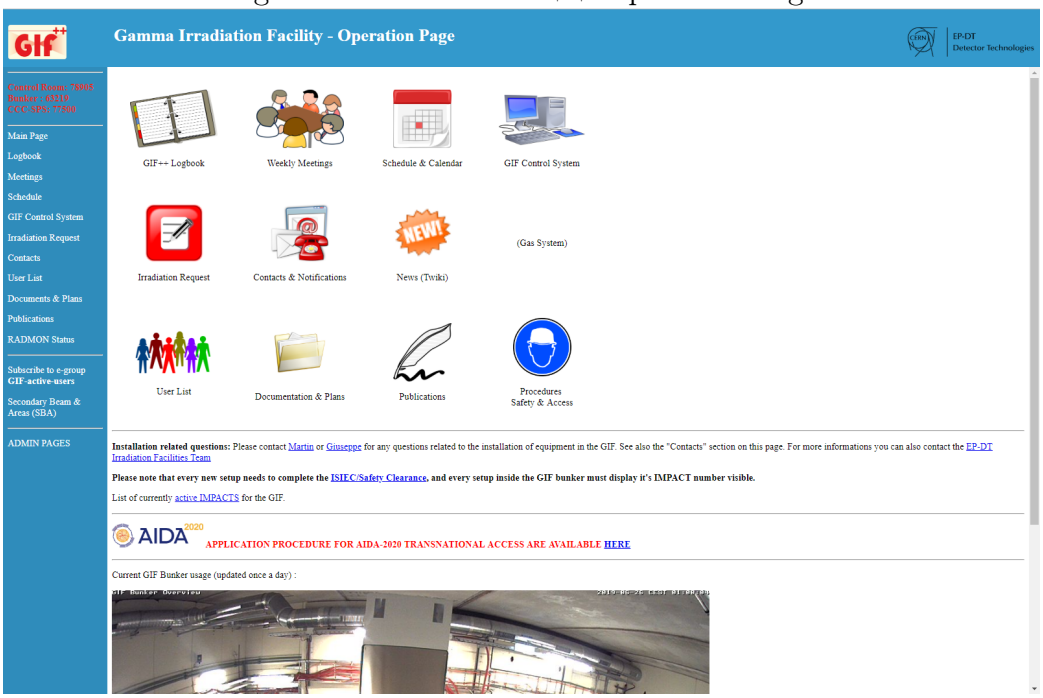


Figure 7: New GIF++ Operation Page

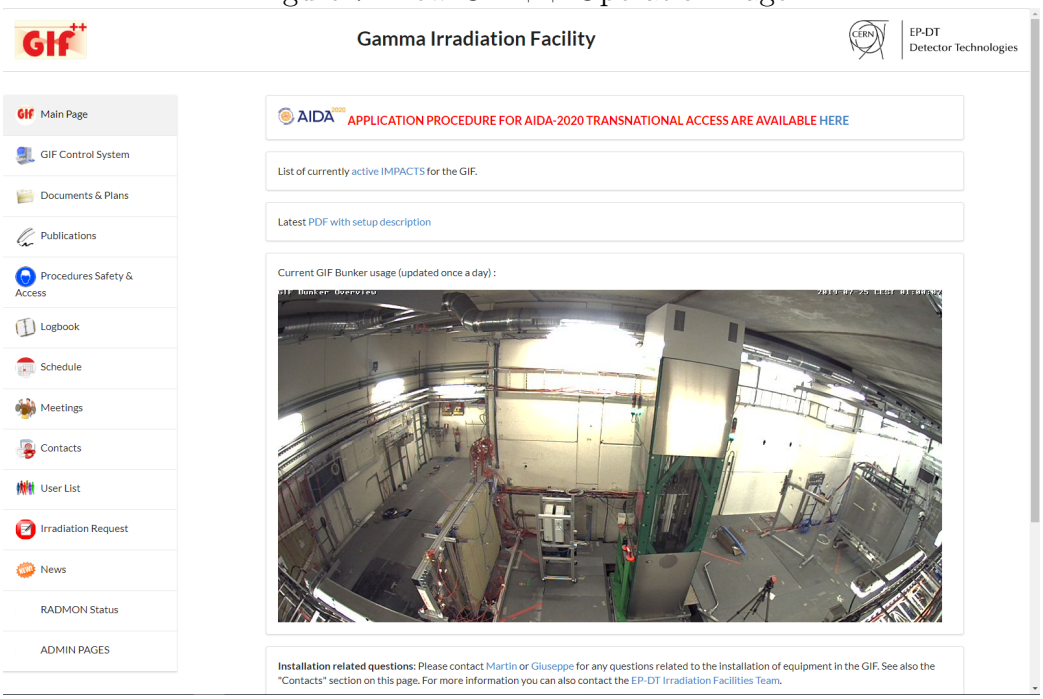
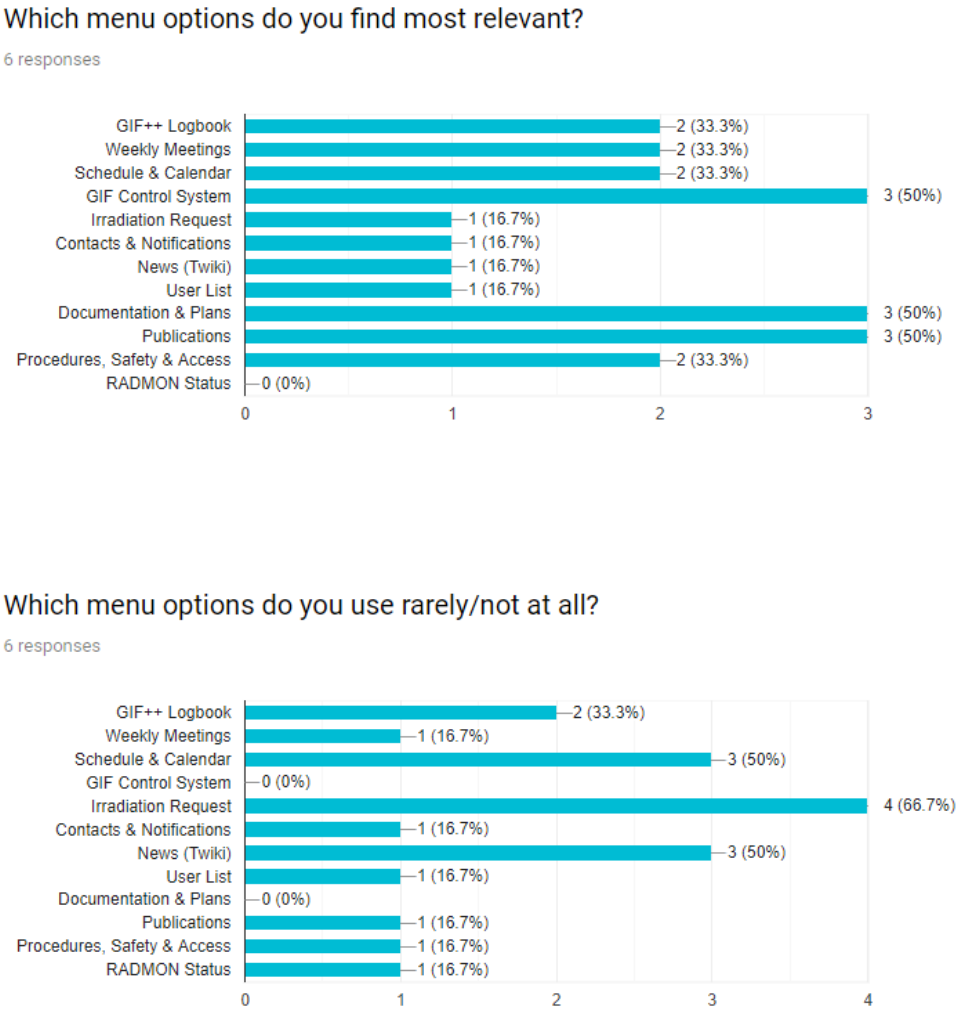
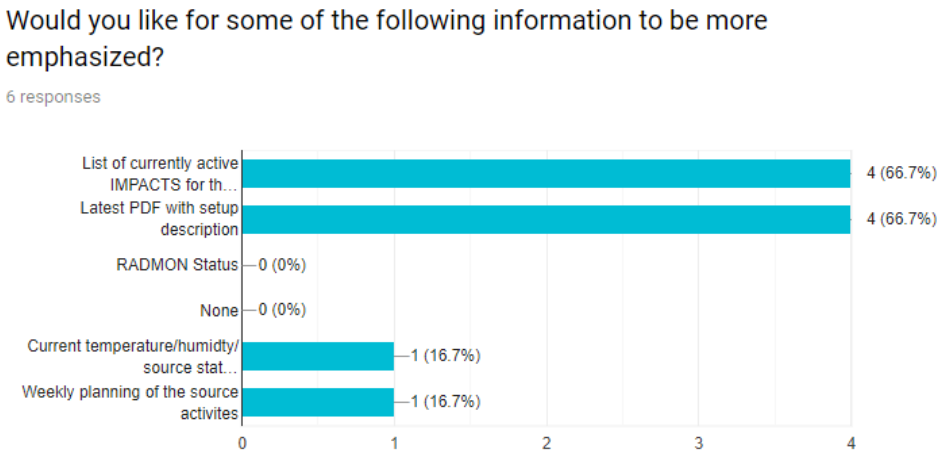


Figure 8: Questionnaire results





3 Conclusion

My summer student project achieved the following goals improving both the PyTimber and GIF++ web pages:

1) Highcharts handles well the rendering of millions of data points. It offers many options and is user friendly.

2) The CSV file has consistent and comprehensible format.

For large number of data points (couple of millions) it will be a problem to generate CSV file, but for hundreds of thousands it will work fine. It may take too much time for file to be generated so the site crashes, or it may happen that even for request of smaller number of points, the server doesn't respond in time. This problem may be on the CALS side. In future work, one could try to understand these problems better and solve them.

3) The PyTimber can handle structured variables.

4) The timezone is handled correctly.

5) The GIF++ web site has nicer look that is similar to the one of the IRRAD facility [7] operated by the same CERN team and is more user friendly.

Acknowledgments

I would like to express my special thanks of gratitude to my supervisor Blerina Gkotse for her guidance and encouragement. I would also like to thank team members - Federico Ravotti, Giuseppe Pezzullo, Martin R. Jäkel, Isidre Mateu, Jes Rydall Larsen and Sabina Azimova for friendly and welcoming atmosphere and making this experience unique. Special thanks to Roland Sipos for his help with the code. Finally, I would like to thank Summer Student team for doing a great job!

References

[1] PyTimber web application - https://epdtdi-pytimber.web.cern.ch/app/gifpp

[2] Highcharts JavaScript library - version 7.1.2: https://code.highcharts.com/?_ga=2.22018747.1854952904.1565599672-339542877.1564570309

[3] Pydim - http://lhcbdoc.web.cern.ch/lhcbdoc/pydim/guide/pydim-c.html

[4] Moment timezone .js file - https://momentjs.com/timezone/

[5] Semantic UI framework - https://semantic-ui.com/

[6] GIF++ Operation Page https://gif-irrad.web.cern.ch/gif-irrad/

[7] PS-IRRAD web site - http://ps-irrad.web.cern.ch/