LIP-STUDENTS-25-20

XML file generation application for RADEM

Gonçalo Esteves[1,a]

[1] *Instituto Superior Técnico*

Project supervisor: Marco Pinto

October 20, 2025

Abstract. The RADEM (Radiation-Hard Electron Monitor) instrument onboard ESA's JUICE (JUpiter ICy moons Explorer) mission benefits from regularly updated command sequences to configure its operational modes and internal logic, including trigger thresholds and coincidence logic settings. These sequences must be submitted in the form of XML files adhering to specific formatting requirements for spacecraft communication. Traditionally, this process has been manual, time-consuming, and error-prone.
This report presents a Python-based graphical application developed with PyQt5 that automates the generation of RADEM command XML files. The application provides an intuitive interface for configuring command sequences, including bit-level logic definitions with labeled inputs for clarity. Users can build, modify, and manage multiple sequences, save and load configurations, and generate fully formatted XML files ready for uplink. The tool significantly reduces the potential for human error and improves efficiency in mission planning and operation support.

Keywords: ESA, JUICE, RADEM, Operations, ASIC

1 Introduction

The Radiation-Hard Electron Monitor (RADEM) is an instrument onboard the ESA JUpiter ICy moons Explorer (JUICE) mission to Jupiter, launched on April 14, 2023. As part of ESA's Cosmic Vision program, RADEM will operate for over three years, studying the radiation environment of the Jovian system and its icy moons Ganymede, Callisto, and Europa. The instrument operates continuously, supplying data on particle fluxes and energy spectra. It plays a dual role by contributing to spacecraft safety and enabling the acquisition of scientifically valuable measurements.

Modifying RADEM's operational mode or internal configuration requires sending command instructions to the spacecraft, which are delivered as an XML file containing a sequence of executable commands for JUICE. Up to now, the generation of this XML file has been carried out manually, which is both time-consuming and error-prone. For example, updating the coincidence logic requires configuring 255 individual parameters, each represented by a decimal number derived from a 22-bit binary string. These are set manually with the use of lengthy technical tables, making it inefficient and highly susceptible to mistakes. And this is only for one update and one sequence.

The application developed during this internship automates the entire process of XML generation. It provides a graphical interface with clear labels and guided input, allowing the user to configure logic settings efficiently. Configurations can be saved and reloaded, and the final XML file is generated automatically, ready for transmission to the spacecraft.

2 RADEM Details

This section presents only the aspects of RADEM that are relevant to the development of this application. For a more comprehensive description of the instrument, the reader is referred to [1].

2.1 The RADEM instrument

RADEM is composed of three detector heads:

- the Electron Detector Head (EDH) with its Electron Stack Detector (ESD),
- the Proton and Heavy Ion Detector Head (P&HIDH) with its Proton Stack Detector (PSD) and Heavy Ion Stack Detector (HISD),
- and the Directional Detector Head (DDH).

See Figure 1.

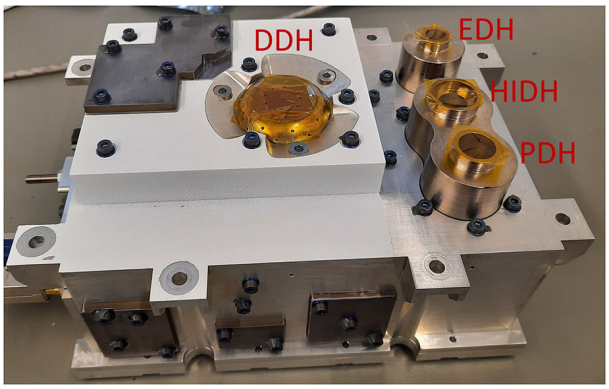

Figure 1. Photograph of the RADEM ProtoFlight Model with the Electron Detector Head (EDH), the Proton and Heavy Ion Detector Head (P&HIDH: PDH and HIDH), and the Directional Detector Head (DDH). [1, p. 5]

Each detector head is connected to dedicated readout electronics equipped with custom-designed front-end Application-Specific Integrated Circuits (ASICs) developed specifically for JUICE by IDEAS (Norway).

[a] e-mail: goncalo.maia.esteves@tecnico.ulisboa.pt

2.2 The ASIC

These and other ASIC circuits are described in detail in [2]. Note that pulse-height analysis is not discussed since it is not used on RADEM. A schematic view can be seen in Figure 2.

The ASIC features 36 input channels equipped with charge-sensitive preamplifiers:

- 4 low-gain (LG) channels, and
- 32 high-gain (HG) channels.

Signal readout is performed via level triggers (thresholds), with the following discriminator configuration:

- For LG channels: one Low-Threshold (LGLT) discriminator
- For HG channels: one Low-Threshold (HGLT) and one High-Threshold (HGHT) discriminator

Each threshold type has a different dynamic range that can be programmed through a 10-bit Digital-to-Analogue Converter (DAC).

The resulting signals are then processed by 36 pattern units, each connected to a dedicated 22-bit counter. Individual patterns can be configured using flag bits corresponding to all 68 thresholds: 4 from the LG channels and 64 from the HG channels ($64 = 2 \times 32$). Each threshold is associated with two configurable bits: one to enable or disable its contribution via a mask, and another to define whether the threshold is used in coincidence (above threshold) or anti-coincidence (below threshold) mode. This results in a total of $136 = 2 \times 68$ configurable bits per counter. The structure of the bit fields in the coincidence logic configuration registers is detailed in the datasheet [3, p. 43], where additional configuration registers for other functionalities are also described.

The coincidence time can also be programmed globally from 50 ns to hundreds of ns, and the global gain of the ASIC can be tuned from 0.75% to 1.25% in case the environmental conditions or aging of the system justifies an adjustment. All settings and counter values can be written and read via a serial interface.

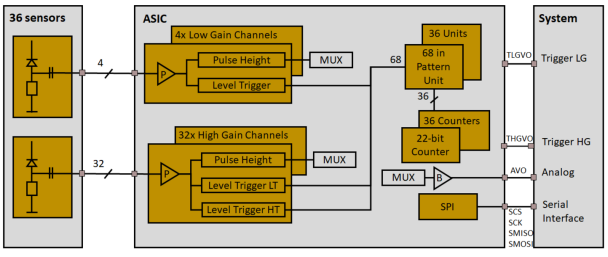

Figure 2. ASIC IDE3466 detector readout schematic [1, p. 7]. Sensors can be connected to either Low-Gain or High-Gain channels. PSD, ESD and DDH are connected to High-Gain channels while the HISD is connected to Low-Gain channels

3 Usage Instructions and Features

To run the application, follow the steps provided in the `README.md` file. Once launched, the main window appears as shown in Figure 3. It is structured into three main components:

- **Sequence selection and editing panel (left):** This section allows the user to select a command sequence (e.g., "To safe mode") from a dropdown menu and set its parameters. For example, the execution time can be specified using the format `yyyy-dddThh:mm:ss.000Z`.
- **Occurrence List panel (right):** Displays the list of sequences added by the user. Sequences appear here in the order they will be written to the final XML file. A "Delete" button allows for sequential removal (i.e., only the last added sequence can be removed).
- **XML generation button (bottom):** The "Generate XML" button creates the XML file based on the defined sequences. The file is saved automatically in the `xml_output/` directory.

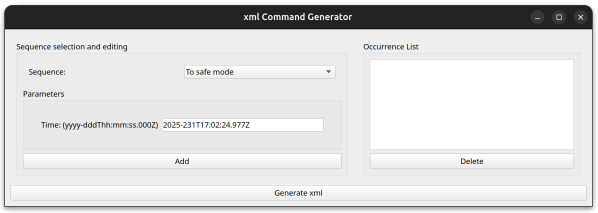

Figure 3. Default view of the application upon starting it.

3.1 Sequence Selection and Editing

The left panel allows the user to configure and add command sequences to the Occurrence List. It includes a dropdown menu where the user can select from four predefined sequences:

- **To safe mode**
- **CL logic**
- **To normal mode**
- **TL logic**

The Trigger Logic (TL) sequence is not implemented and serves as a placeholder only.

Mode Change Sequences

The "To safe mode" and "To normal mode" sequences allow the user to switch RADEM into and out of Safe mode. This is necessary for logic updates to be performed safely. Both sequences only require a timestamp parameter, which must be entered in the format `yyyy-dddThh:mm:ss.000Z`. See Figure 4. Once configured, clicking the "Add" button inserts the sequence to the end of the Occurrence List.

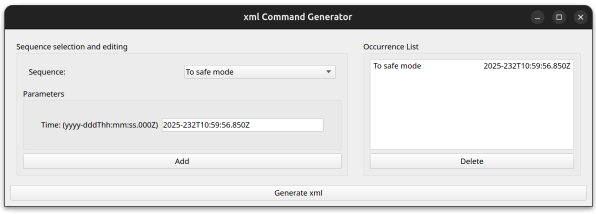

Figure 4. Application Window after the addition of a Safe mode sequence.

Coincidence Logic (CL) Sequence

When the user selects "CL logic", a detailed parameter panel appears. See Figure 5. This sequence is used to update the coincidence logic configuration of a specific ASIC (EDH, PIDH, or DDH). The following parameters are required:

- execution time,
- ASIC selection (EDH, PIDH, DDH),
- counter index (1–36),
- register index (1–7),
- bit values (0 or 1) for each logic channel.

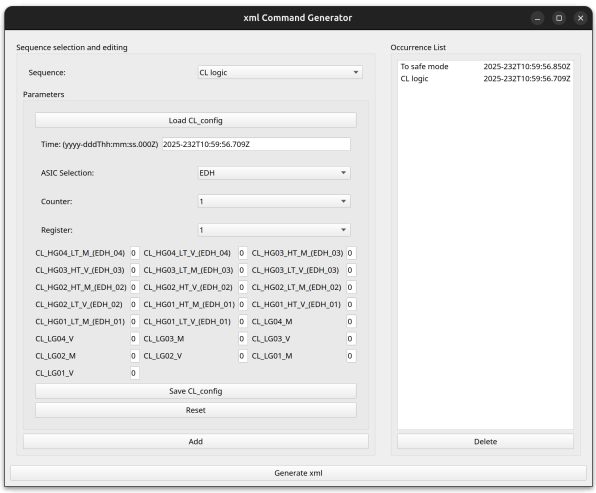

Figure 5. Application Window after the addition of a CL logic sequence. Shows the Sequence "CL logic" editing parameters.

Each bit is labeled with its corresponding detector and logic type (e.g., `CL_HG01_HT_M_(EDH_01)`), allowing the user to modify logic settings with clear reference to the hardware.

Additional functionality is provided to support user workflows:

- **Load CL_config**: Load a previously saved configuration.
- **Save CL_config**: Save the current configuration to a file.
- **Reset**: Clear all parameters to default values.

These configuration files are saved by default to the `config/` directory, but the user may choose a different location via the standard file dialog (Figure 6).

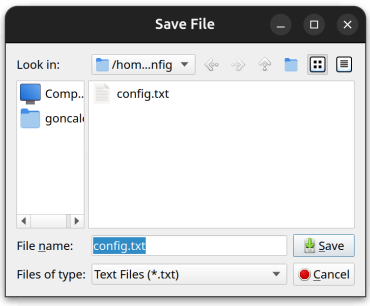

Figure 6. Save file dialog window.

These configuration files are stored as `.txt` files containing a dictionary (in JSON format) with three keys: "time", "asic", and "logic". These keys correspond directly to the data shown in each parameter editor, as illustrated in Figure 5.

3.2 Occurrence List

The Occurrence List (right panel) displays all sequences added by the user. Each entry shows the sequence type and its associated execution time, providing a clear overview of the scheduled sequences.

A "Delete" button allows for the removal of the most recently added sequence, i.e. the last row.

3.3 XML Generation

Once all desired sequences have been added to the Occurrence List, the user can generate the final XML file by clicking the "Generate xml" button at the bottom of the window. A file dialog will prompt the user to select the save location. By default, xml files are saved to the `xml_output/` directory.

The resulting XML file is formatted according to the specifications required for upload to ESA's systems, containing all configured sequences in the correct structure for execution on JUICE.

4 Implementation

The application was developed in Python 3.13.2 using the PyQt5 5.15.11 library for the graphical user interface and the xml.etree.ElementTree module from the standard library for XML parsing. All code was written and tested on Linux Ubuntu 24.04.2 LTS using Visual Studio Code.

4.1 Project Files and Directories

The application is divided into several components, as shown below:

```
XMLgeneratorApp/
 config/
    config.txt
 gui/
    widgets/
       Parameters_editors/
          CL_editor.py
          NormalMode_editor.py
          SafeMode_editor.py
       dropDown.py
       OccurListGroup.py
       SequenceGroup.py
       time_input.py
    main.py
    mainWindow.py
 source/
    labels_bits/
       label_names_DDH.txt
       label_names_EDH.txt
       label_names_PIDH.txt
    SequenceClasses/
       CLSequence.py
       NormalModeSequence.py
       SafeModeSequence.py
       Sequence_base_class.py
    functions.py
    OccurList.py
 xml_output/
    output.xml
 README.md
```

The gui/ folder contains all the custom widgets developed for the application. It includes the mainWindow.py file, which integrates these widgets into a single window (as shown in Figure 3), and the main.py file, the program's entry point.

The source/ folder defines the internal logic of the application. It includes:

- The SequenceClasses/ module, which defines the sequence types and a base class.
- The OccurList class in OccurList.py, used to manage the list of added sequences.
- The functions.py file, which provides utility functions such as reading from a file to a matrix.
- The labels_bits/ folder, which contains label definition files for each ASIC.

The config/ folder stores configuration files saved during application use, and the generated XML files are saved by default in the xml_output/ folder. Also a README.md file is included to provide basic documentation and usage instructions for the application.

4.2 Working Principle

The internal logic of the application is implemented in the source/ module. This module defines the core classes responsible for representing individual sequences and managing the overall list of sequences to be included in the final XML file.

The file Sequence_base_class.py contains an abstract base class that defines attributes and methods shared by all sequences. Each specific sequence type, "To Normal Mode", "To Safe Mode", and "CL Logic", is implemented in its own file: NormalModeSequence.py, SafeModeSequence.py, and CLSequence.py, respectively. These classes inherit from the base class and define sequence-specific parameters, as well as methods for XML building and writing (though the latter is primarily used for debugging and testing purposes).

The file OccurList.py defines a class responsible for storing the list of sequences created by the user. Internally, the sequences are stored in a dictionary, with the position in the Occurrence List as the key and the corresponding sequence object as the value. As the user adds a new sequence, an object of the respective class is instantiated, populated with the user input, and inserted into the OccurList dictionary.

The OccurList object is instantiated in mainWindow.py, and its methods are called when the "Add", "Delete", or "Generate xml" buttons are pressed to perform the corresponding actions. Sequence objects are instantiated upon pressing the "Add" button, based on the user-defined parameters.

4.3 GUI Implementation

The graphical user interface (GUI) of the application was developed using the PyQt5 framework and is organized within the gui/ directory. The core window layout is defined in mainWindow.py, which integrates the various interface components into a single window.

The majority of the GUI elements are implemented as modular widgets stored in the widgets/ subdirectory. These include components for managing the sequence selection, parameter editing, and occurrence list. Each widget is implemented in its own file to promote modularity and reuse.

Among these components, the parameter editors play a central role in data handling. Located in widgets/Parameters_editors/, each editor corresponds to a specific sequence type (e.g., CL_editor.py, NormalMode_editor.py). These editors are responsible for collecting user input, storing the data in internal dictionaries, and performing the necessary data transformation and validation via their methods, to produce a format compatible with the corresponding sequence classes defined in source/.

The interaction between the GUI and the internal logic follows a clear separation principle. The parameter editors serve as the interface between user input and the underlying application logic. They are instantiated in mainWindow.py, and once the user fills in the required fields and presses the "Add" button, the editor processes the input and passes the resulting data to instantiate the appropriate sequence object. These sequence objects are then stored in the OccurList instance, as described in Section 4.2.

Other GUI components, such as the dropdown menu for selecting sequences, the time input widget, and the occurrence list display, are used mainly in structural or organizational roles.

Overall, the GUI is structured to preserve modularity and separation between interface and logic, enabling easier maintenance and future extension.

5 Conclusion

The development of this Python-based graphical application successfully addressed the challenges of generating RADEM command XML files for the JUICE mission. By automating a previously manual and error-prone process, the tool streamlines the creation of complex configuration sequences, reducing the likelihood of mistakes and significantly improving efficiency.

5.1 Future Improvements

5.1.1 Codebase structure Improvements

The apllication was initially structured with a high degree of modularity, following the principle of defining a single class per Python file. While this approach facilitates extensibility and separation of concerns, it has led to an excessive level of fragmentation, which may hinder maintainability in the long term.

In future updates, it would be advisable to reduce the number of individual files by grouping related classes into single modules. For instance, the `SequenceClasses/` directory could be replaced by a single Python file containing all sequence-related classes. The same applies to the `Parameters_editors/` subdirectory, and, to some extent, to the `widgets/` directory as well.

5.1.2 Interface Improvements

Several enhancements could be implemented to improve the application's usability and robustness but were not addressed due to time constraints. In addition to extending the number of supported sequences, such as implementing the TL logic sequence, more immediate and practical improvements are suggested below. These recommendations are intended to guide future developers in continuing and refining the application.

- **Input validation for time format:** Currently, the application does not verify whether the user-provided timestamp adheres to the required format. Integrating a dedicated date-time widget from PyQt5 would be a reliable way to ensure valid input and improve the user experience.
- **Flexible sequence management:** At present, sequences can only be appended to the end of the Occurrence List and removed in a last-in-first-out manner. Future versions should allow users to insert and delete sequences at arbitrary positions in the list, enabling more precise control over the command schedule.

References

[1] W. Hajdas, P. Gonçalves, M. Pinto et al., Space Sci Rev **221**, 43 (2025)

[2] T.A. Stein, P. Påhlsson, D. Meier, A. Hasanbegovic, H.K.O. Berge, M.A. Altan, J. Ackermann, B. Najafiuchevler, S. Azman, J. Talebi et al., *Front-end readout ASIC for charged particle counting with the RADEM instrument on the ESA JUICE mission*, in *Space Telescopes and Instrumentation 2016: Ultraviolet to Gamma Ray*, edited by J.W.A. den Herder, T. Takahashi, M. Bautz, International Society for Optics and Photonics (SPIE, 2016), Vol. 9905, p. 990546, `https://doi.org/10.1117/12.2231901`

[3] Preliminary Datasheet DS, Integrated Detector Electronics AS (IDEAS), under ESA contract (2016), project: Jovian RADEM (RadHard Electron Monitor Proto-Flight Model). ESA sub-contract 4000110643 between EFACEC and IDEAS, under ESTEC contract 4000104778/14/NL/HB