Report on the Automation of Data Loading from the RT13m in a Python App

M. García

Marzo 2026

Observatorio de Yebes
Apdo. 148, E-19080
Guadalajara, Spain

Table of Contents

List of Figures

1 Introduction

This report describes the work carried out to automate the process of loading data from the `RT13m` into a Python application (`FS_Message_Form`) through the use of graphical interface buttons. The main objective of this task was to improve the usability of the application, reduce manual intervention, decrease the time required to complete the process, and make the workflow more efficient and intuitive for the user.

Before this improvement, the process of loading and managing data involved several manual steps, which increased the risk of user errors and made the application less convenient to use. To address this issue, a button-based interaction model was implemented, allowing users to execute data-loading actions directly from the interface in a simpler and more controlled way.

2 Initial Workflow

Initially, the application relied on a manual method for entering or loading data. The user had to search through a very large log file to find all the required information, copy it into the application, and, in some cases, calculate up to 16 averages from large sets of data.

This made the application cumbersome to use, as it required significant effort and attention from the user to complete all the fields correctly. In addition, the probability of making errors during the process was high.

3 Implemented solution

To solve this problem, the first step was to extract all the necessary information from the `RT13m` logs. For this purpose, a Python script was developed to parse the entire log file using regular expressions and store the extracted data in `JSON` format. Some of the extracted data include:

- Session name
- Dot, Dot2Gps, and Dot2Pps values
- Session start and end times
- Temperature, pressure, and humidity
- Coordinates
- Sky conditions (see Section 3.1)

Among many other fields, the complete data structure can be found in Appendix 1.

Then, several functions were implemented within the application itself to process the extracted data and place each value into its corresponding field. These functions were responsible for reading the generated `JSON` file, selecting the relevant information for each tab, and automatically populating the interface fields with the appropriate values.

3.1 Sky Conditions

To obtain information about sky conditions, a Python module called `skyConditions` was developed. Using the `meteostat` library, it retrieves the measured value of **cldc** (cloud cover) for given coordinates and date. Then, by applying the function shown in Appendix 2, an approximation of the sky conditions during the session is obtained.

Once all the information has been extracted, a `JSON` file is generated. This format is much easier to handle from the software point of view and allows the different tab fields to be filled in automatically.

4 Final Workflow

The application workflow is shown in Figure 4.1. First, the parser is executed by pressing the `Load` button in the `Ready` tab. This generates the corresponding `JSON` file and automatically fills in the `Ready` page. Once this step is completed, the user can press the `Load` buttons in the `Start` and `Stop` tabs, and the fields are then populated directly from the generated `JSON` file.

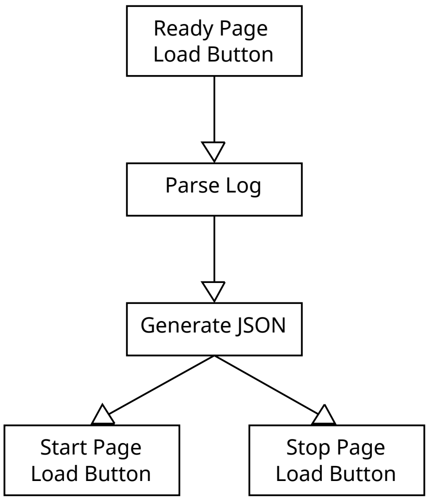
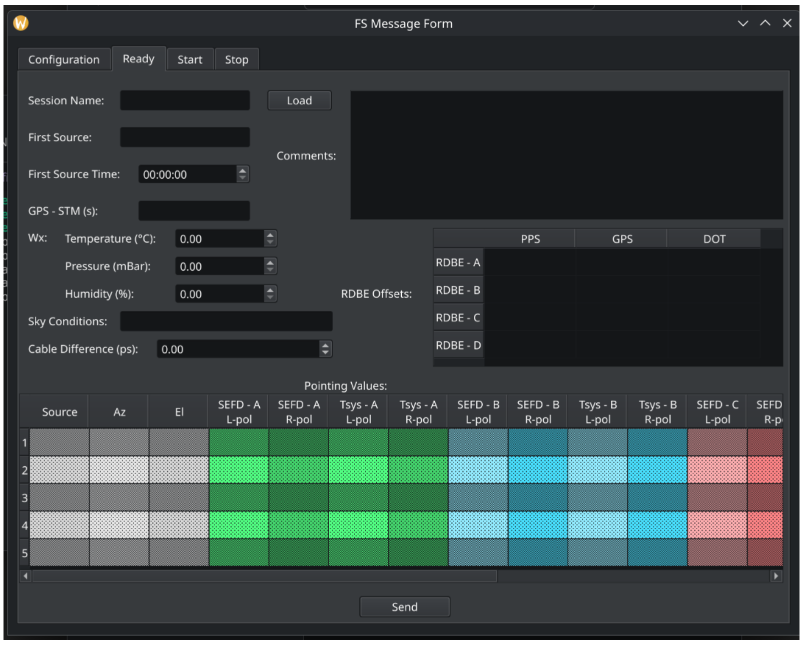


Figure 4.1: Application workflow

5 Ready Page

To automate the loading of information, a `Load` button was added to the interface (see Figure 5.1). This button allows the user to select a log file from the local file system, parse it automatically, and generate a `JSON` file containing all the relevant information.



Session Name:	
First Source:	
First Source Time:	00:00:00
GPS - STM (s):	
Wx: Temperature (°C):	0.00
Pressure (mBar):	0.00
Humidity (%):	0.00
Sky Conditions:	
Cable Difference (ps):	0.00
Comments:	

RDBE Offsets:

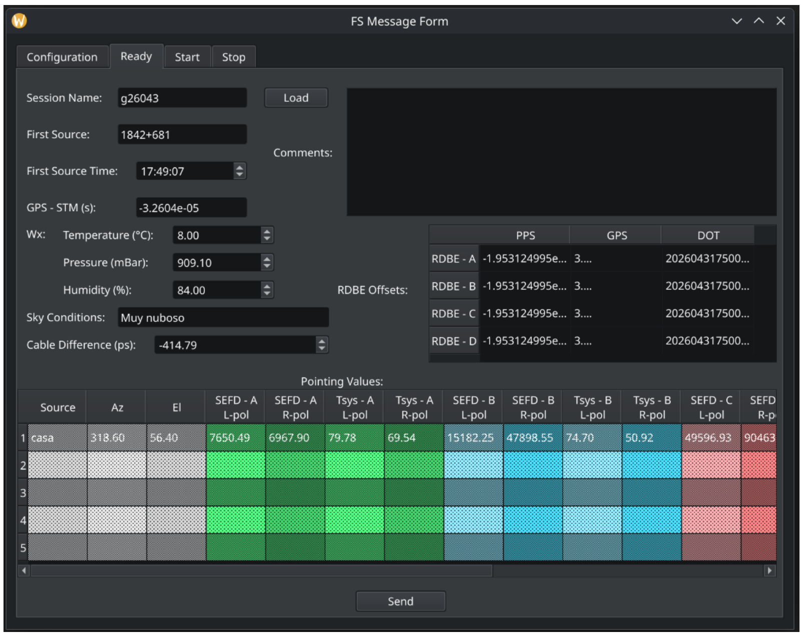
	PPS	GPS	DOT
RDBE - A			
RDBE - B			
RDBE - C			
RDBE - D			

Pointing Values:

	Source	Az	El	SEFD - A L-pol	SEFD - A R-pol	Tsys - A L-pol	Tsys - A R-pol	SEFD - B L-pol	SEFD - B R-pol	Tsys - B L-pol	Tsys - B R-pol	SEFD - C L-pol	SEFD R-p
1													
2													
3													
4													
5													

Figure 5.1: Ready page with the `Load` button before selecting a log file

Once the file has been generated, the page is filled automatically, as shown in Figure 5.2. This saves operators a considerable amount of time compared to manual data entry and significantly simplifies the overall process.



Session Name: g26043
First Source: 1842+681
First Source Time: 17:49:07
GPS - STM (s): -3.2604e-05
Wx: Temperature (°C): 8.00
Pressure (mBar): 909.10
Humidity (%): 84.00
Sky Conditions: Muy nuboso
Cable Difference (ps): -414.79
Comments:

RDBE Offsets:

	PPS	GPS	DOT
RDBE - A	-1.953124995e...	3....	202604317500...
RDBE - B	-1.953124995e...	3....	202604317500...
RDBE - C	-1.953124995e...	3....	202604317500...
RDBE - D	-1.953124995e...	3....	202604317500...

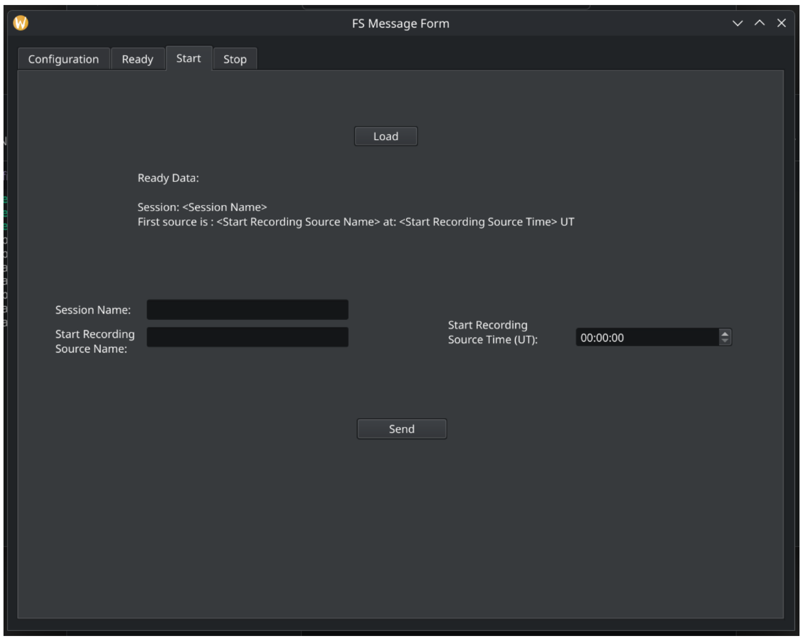
Pointing Values:

	Source	Az	El	SEFD - A L-pol	SEFD - A R-pol	Tsys - A L-pol	Tsys - A R-pol	SEFD - B L-pol	SEFD - B R-pol	Tsys - B L-pol	Tsys - B R-pol	SEFD - C L-pol	SEFD R-p
1	casa	318.60	56.40	7650.49	6967.90	79.78	69.54	15182.25	47898.55	74.70	50.92	49596.93	90463
2													
3													
4													
5													

Figure 5.2: Ready page after parsing the log file

6 Start Page

Once the `JSON` file has been generated, the `Load` button in this tab (see Figure 6.1) simply accesses it and extracts the relevant information required to fill in the page automatically.



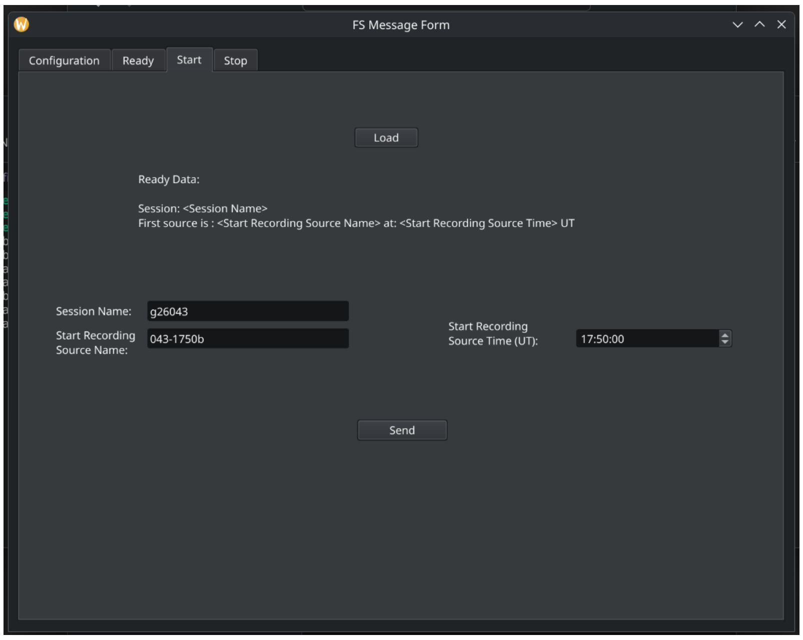
Ready Data:

Session: <Session Name>
First source is : <Start Recording Source Name> at: <Start Recording Source Time> UT

Session Name:
Start Recording Source Name:
Start Recording Source Time (UT): 00:00:00

Figure 6.1: Start page before loading the data

In this page, the session name, the time of the first recording, and the name of the first source observed by the radio telescope at the moment the recording starts are filled in automatically, as shown in Figure 6.2.



Ready Data:

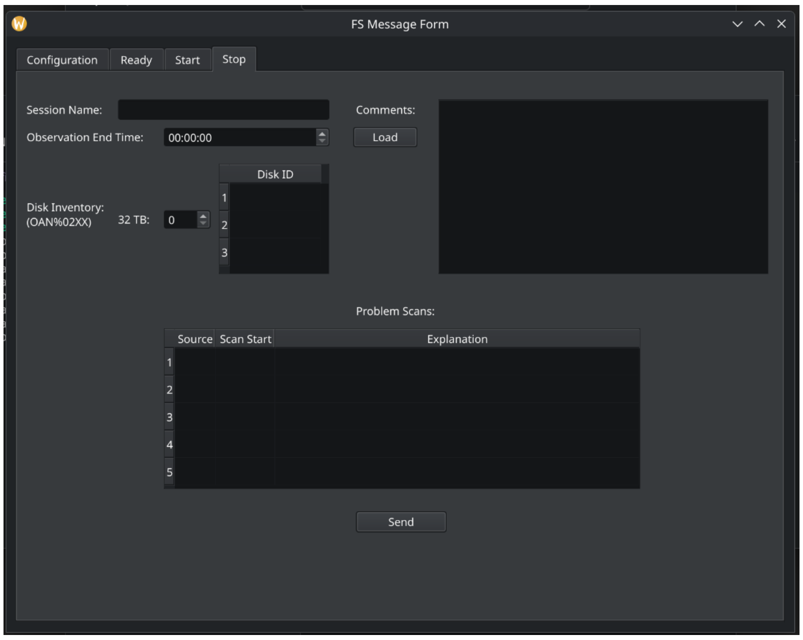
Session: <Session Name>
First source is : <Start Recording Source Name> at: <Start Recording Source Time> UT

Session Name:	g26043
Start Recording Source Name:	043-1750b
Start Recording Source Time (UT):	17:50:00

Figure 6.2: Start page after loading the data

7 Stop Page

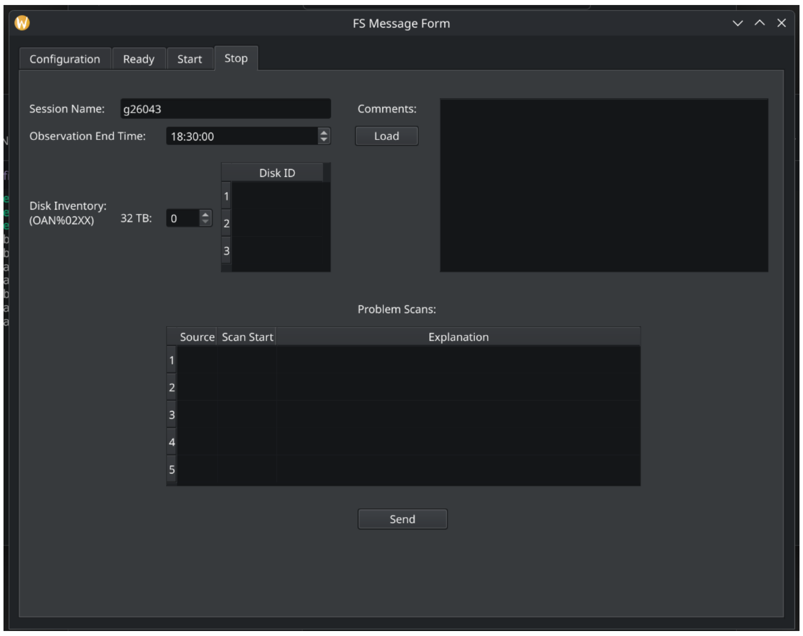
The `Stop` page follows the same automated loading logic as the previous tabs. Once the `JSON` file has been generated, the `Load` button in this tab (see Figure 7.1) retrieves the required information and uses it to automatically populate the corresponding fields.



Session Name:	
Observation End Time:	00:00:00
Disk Inventory: (OAN%02XX) 32 TB:	0

Figure 7.1: Stop page before loading the data

In particular, this page is completed with the session name and the information corresponding to the last disk record. The final result after loading the data can be seen in Figure 7.2.


FS Message Form

Configuration | Ready | Start | Stop

Session Name: g26043

Observation End Time: 18:30:00

Disk Inventory: (OAN%02XX) 32 TB: 0

Comments: Load

	Disk ID
1	
2	
3	

Problem Scans:

	Source	Scan Start	Explanation
1			
2			
3			
4			
5			

Send

Figure 7.2: Stop page after loading the data

8 Results and Benefits

The implemented solution significantly reduced the amount of manual work required to fill in the application fields. Tasks that previously required searching through the log files, copying values manually, and performing repetitive calculations can now be completed automatically through the interface.

This improvement reduced the probability of human error, simplified the workflow for operators, and made the application faster and more convenient to use in practice. In addition, the use of an intermediate `JSON` structure made the solution more modular and easier to maintain or extend in the future.

A Appendix: Codes

Listing 1: Initialization of the session data structure

```python
session_data = {
    "session_name": session_name,
    "sky_conditions": None,
    "first_source": None,
    "first_source_time": None,
    "first_source_exper_initi": None,
    "first_source_time_exper_initi": None,
    "end_source": None,
    "end_source_time": None,
    "cryogenic_prev": [],
    "cryogenic_post": [],
    "location": [],
    "gps_stm": [],
    "wx": [],
    "cdms": [],
    "rdtca_dot": [],
    "rdtca_dot2gps": [],
    "rdtca_dot2pps": [],
    "rdtcb_dot": [],
    "rdtcb_dot2gps": [],
    "rdtcb_dot2pps": [],
    "rdtcc_dot": [],
    "rdtcc_dot2gps": [],
    "rdtcc_dot2pps": [],
    "rdtcd_dot": [],
    "rdtcd_dot2gps": [],
    "rdtcd_dot2pps": [],
    "disk_record": [],
    "onoff_pre_initi": [],
    "onoff_post_initi": [],
    "scans": [],
    "_pending_scan": [],
    "onsource_blocks": [],
    "_pending_onsource": None
}
```

Listing 2: Function used to estimate sky conditions from cloud cover

```python
def sky_from_cldc(cldc):
    """Devuelve un string con el estado del cielo a partir de la nubosidad media (cldc)
    Args:
        cldc: Nubosidad media (cldc) en octas (0-8)
    Returns:
        Un string con el estado del cielo
    """
    if pd.isna(cldc):
        return None
    if cldc <= 1:
        return "Despejado"
    elif cldc <= 3:
        return "Poco nuboso"
    elif cldc <= 5:
        return "Intervalos nubosos"
    elif cldc <= 7:
        return "Muy nuboso"
    else:
        return "Cubierto"
```