Space mission message management software using European Space Agency's protocols

Adrià Grau Navacerrada

Resum– L'Institut de Ciències de l'Espai (ICE) utilitza protocols dissenyats per l'Agència Espacial Europea (ESA) per gestionar els missatges que controlen els satèl·lits de les seves missions. Aquests missatges poden tenir milers de variacions en cada missió i la seva gestió es fa manualment amb eines que dificulten tant el seu maneig com la seva creació. Així doncs, aquest projecte ha dissenyat i implementat un sistema de control dinàmic dels missatges de l'esmentat protocol per tal de facilitar-ne la seva creació, control i posterior distribució. Aquest s'ha materialitzat en una pàgina web amb control d'accès i, a més, s'ha introduït millores per exportar els missatges a un llenguatge de programació per tal de que puguin ser manipulats i/o implementats en múltiples projectes. Finalment, el sistema implementat aconsegueix que l'usuari no s'hagi de preocupar pel format del missatge (com passava fins ara) ja que el primer s'encarregarà d'adaptar-los correctament.

Paraules clau– IEEE, ICE, ESA, ESTEC, protocol ECSS–E–50, protocol ECSS–E–70-41A, missatgeria, telemetria, telecomanda, software, gestió de comandes, exportació codi.

Abstract– The Institut de Ciències de l'Espai (ICE) uses protocols designed by the European Space Agency (ESA) to manage the messages that control its satellite missions. Those messages can have thousands of variations in each individual mission and they are currently manually handled with tools that make their creation and management very difficult and tedious. For those reasons this project has designed and implemented a dynamic control system of its messages to make them easier to create, control and manage their posterior distribution. This system is materialized through a web page that features a user acess control and introduces new improvements to export the messages to code language in order to manipulate or implement them in different projects. Finally, the system has been implemented with the main idea of facilitating the tasks to the users as they will not have to worry about the message structure since the system will adapt it correctly.

Keywords– IEEE, ICE, ESA, ESTEC, ECSS–E–50 protocol, ECSS–E–70-41A protocol, messaging, telemetry, telecommand, software, command management, code exportation.

1 INTRODUCTION

THE *ICE-CSIC* (*Institut de Ciències de l'Espai*) [1] is an organization that does aerospace research since its creation in 1988. As all the science fields, there is a lot of regulations and standards that experiments and programs must comply with and for this reason their satellite related projects follow the *ECSS-E-50* [2] protocol of the European Space Agency (*ESA*) [3]. This protocol, which will be described on the first appendix, assumes that there is a direct connection between a ground control centre (in this case the *ICE-CSIC*) and a remote artificial satellite so they both comunicate through a series of heavily formated data packets known as *messages*. Those messages are unique and they can be classified into two different groups:

- **Telecommand:** Data packet that contains orders for the satellite to execute. They are sent from the control centre
- **Telemetry:** Data packet that contains the results of a

• E-mail de contacte: adria.grau@e-campus.uab.cat
• Menció realitzada: Enginyeria del Software
• Treball tutoritzat per: Lluís Gesa Bote (IEEC-CSIC) i Alicia Fornés Bisquerra (CVC)
• Curs 2019/20

satellite order execution or its status. They are sent to the control centre from the satellite.

Both kind of messages can have multiple (hundreds if not thousands) variations for a single project so there is a real need to organize them efficently because they will be manually handled most of the time. For this concrete reason, the *ICE-CSIC* designed a spreadsheet template that joins all the different messages of the same project under the same document but it became too complex (dozens of different sheets interlinked) and less *user-friendly* (all the information was cramped and difficult to read) faster than expected as the years passed by and the projects started being more ambitious.

Just in this point is where this project initiated in order to create a new sorting system that is more *user-friendly* and that it can automatically manage and export the existing messages, minimizing the user interaction and incrementing the efficiency of the overall system. This new system will be named *ASMM* (acronym for *Advanced Space Messagery Manager*) and another important point that it requires is the implementation of the new setup in a remote, multi-device and role based environment so everyone in the *ICE-CSIC* can use it whatever device they use and there's a high level of privacy between different projects or users. Finally, another important point is to create a *plug-&-play live-code* command exportation so the commands can be used right away the extraction in a live project without having the user to adapt them manually.

For all those motives, the main objectives of this project are detailed in the initial planning document [4] and are the following:

- **OBJ-1: Create a message management application using *ESA* protocols:** Main objective of the project that will deliver a working message management system that complies with all the previously listed needs.
- **OBJ-2: Develop a spreadsheet manipulator to import/export messages:** This objective will be required to complete the first one and it will also deliver a new input system of the old spreadsheets used in the currently active projects to adapt them to the new application.
- **OBJ-3: Create a message exportation system in different programming languages:** This last application objective will result in a system that exports existing messages to *ready-to-use* code through previously user-inputed templates to increment work-efficiency.

Apart from the application-related objectives there is also another meta-objective related to the whole software creation process that will applicate all the gained knowledge obtained while doing the *Enginyeria del Software* mention and that will cover and apply in a real case much of its content:

- **OBJ-4: Apply Software Engineering processes in the development of a software application:** Meta-objective of the project that scopes the whole project itself. It will deliver all the associated project documentation, code, meetings... It is, in abstraction, the project itself.

After exposing the main motivations and objectives that define this project, the rest of the present document will be consequently divided in the multiple parts or main processes that conform it. Being so, the following section will be the *State of the art* which will deeper in the actual technologies and the ones used in the application. This will be followed by the *Project methodology* that has been used during the whole development and, afterwards, the next sections will give more context about the mentioned methodology steps (being the *Situation analysis*, *Design proposal*, *Solution implementation* and *Results and validation*). After this, and just before the *Dissertation acknowledgments* and *Bibliography* parts, there will be a *Future work lines* and *Conclusions* parts that will evaluate the project's possible expansions and the knowledge obtained of it as a whole and expose what has been learned or improved from the initial situation. Finally, just after all these sections, there will be one annex that explains the *ECSS-E-50* protocol in more detail and another one with captures of the implemented system.

2 STATE OF THE ART

This project began having an actual technologic base: the spreadsheets that the *ICE-CSIC* created to organize their projects, and the current web development trends. In order to explain the initial situation this section will be divided in three diferent parts: *Initial infrastructure*, *Actual web development and security tendencies* and *State of the art critique*.

2.1 Initial infrastructure

During the conception of this project, and as it was stated before, the infrastructure was based on a series of spreadhseets that contained all the information. Those were uploaded to a shared repository that only contained the latest version of the document without neither access control nor integrity checking. As it has been also mentioned, they stored all the different messages of an *ICE-CSIC* project and the information was cramped and nearly unreadeable for new users as it was distributed between multiple interlaced sheets that contained differents parts of the same message. Even though that just this reason alone can justifiy the need of developing this project, there is also another problem in the initial infrastructure: the project privacy was never considered and any user could share or erase any message of any project without leaving any trace.

In terms of the factual infrastructure all spreadhseets were manipulated with programs like *Microsoft Excel*, meaning that every single one of the messages had to be manually inputed with all its fields defined. Also the integrity checking was always a troublesome matter since any user could upload a newer version that could overwrite any new change in the repository (there was not any version control method implemented).
Finally, another difficult issue to solve was the project or message exportation: the only existing way to export any message was to copy its contents (taken from different sheets) and put them together under the same spreadsheets and, in case of the projects, the only option was to use

the same given spreadsheet without having the posibility to change neither format nor type.

2.2 Actual web development and security tendencies

As it can be consulted in the initial planning report, one of the most important things to develop in this system is a web environment that is implemented in a major current and trending technology so the application can have a longer life expectancy without needing to update its components. Even though this technology was initially proposed to be *Django* [5], after a lot of considerations and different approaches, it derived to *Flask* [6] and *Jinja2* as they are more flexible and easier to modify in the future for the proposed solution.
Flask is a *Python 3.6* web development framework that allows the creation of webpages with a *Python* backend. It uses the *Jinja2* code language to show all the database contents to the user, reducing both time and resources needed. By using this system it is expected to reduce the bandwith and user waiting time between different orders/pages, leading to a more responsive and *user-friendly* environment. The actual and proposed design implementation using this technology will be discussed under the *Design proposal* section.

Another important point to look at is the data persistance. The current and more modern tendencies of *Big Data* centres are technologies like the known as *No-SQL* [7]. Even though this trend was contemplated in the initial design, the final implementation derived to a *SQL* database since all the project data is linear and there is a need for a high integrity of the database that only *SQL* can provide in a highly reliable level. Furthermore, another reason to use this kind of databases is their easy and powerful internal network integration because they can be easily installed in a server (as it is expected after the conclusion of this project) and maintain a high availability time and non-repudiation ratios. As it will be disclosed in further sections, the used system is *phpMyAdmin* since it implements an easy to maintain, free and a reliable *SQL* server database manager that can be installed in all kinds of operative systems (removing by this way a possible server *OS* type incompatibility).

The last and final issue to consider is the security. As the system is accessible through a webpage there is a lot of known vulnerabilities (such as fraudulent code injection, password override, cookie manipulation...) that need to be controlled. Fortunately, *Flask* has build-in functionalities that check if the user tried to trick the system through code injection and removes the malicious content of text strings, so there is no need to create a new system to check this vulnerability.
In regard of password override and cookie manipulation, the most widespread security trends rely in a mitigation of the possible impact since nowadays it is not possible to ensure complete system encapsulation against those threats. For this sole reason, the application needs to use as less cookies as it is possible to reduce the threats to a marginal percentage that would leave traces in the system (specially important since the *ICE-CSIC* network that will use the application will be closed).
Last but not less important, there is also another important tendency: using *RBAC* (acronym for *Role Based Access Control*). This higher level of control allows the application to divide the users in four different categories (*Administrator* who can edit and manage all the entire system, *Project Leader* who can create and edit entire projects, *Project Editor* who can modify the contents of a project and *Visitor* who can only view projects without interacting further with them) and also prevent them to do actions that they are not allowed to (creating a full safe-vault and *KISS* environment as it is established in multiple security standards [8]). Going deeper into this last topic, there is also another access control security policy at the level of projects: no user that hasn't been previously allowed will be able to see a project and interact with it (except for the system administrator who will have control over all the system as most of the current security standards indicates).

2.3 State of the art critique

As it has been exposed in the past two subsections, there does not exist any program that currently manages the *ICE-CSIC*'s messages explicitely except of the complex and underwhelming spreadsheet workaround. For this reason, and seeing that the current trends of web development lead to *Python* oriented frameworks like *Flask* and data persistence for small scale like *SQL* databases, the application will be developed with both technologies to attain a certain degree of innovation and to create a system that doesn't need to be updated in a few years because it has become deprecated. Another reason to use *Python* is its high portability since most of the *ICE-CSIC* servers have it already installed. Also, in regards of the security, the previously mentioned guidelines will be followed to ensure that the projects receives a new layer of privacy. Finally there is another point to consider: one of the most important issues that current webpages suffer the most is loading times. In order to achieve low waiting times between pages the project will use the *Pandas Library* that contains a lot of already optimized functions that reduce the needed time for executing database processes and has a native *.csv* file type management.

Apart from the small reafirmation and critique of those points, there is also another major thing that the application lacks: a better way to export projects. As the message's fields and characteristics are far too particular there is not already know methodology to obtain them from the system, meaning that this will be a new ground to step on and it will generate new functionalities in comparison with the initial system.
Putting everything into a nutshell: the current situation lacks both utilities and functionality and there is new technologies/methodologies that can be applied to create a new system that improves the actual one and allows the student to learn new techniques, thus justifying this project objective and viability.

3 Project methodology

In order to maintain the project's integrity and accomplish the meta-objective *OBJ-4* to develop satisfactorily an entire software application from scratch, a *Spiral* methodology approach has been used. This approach consists in iterating through the same incremental steps over and over to ensure that both direction and status of the project are correct at all times. Each of those iterations will be of two weeks, giving a total of eight exact iterations from the begining of the project (March 8th) and the ending (June 28th). For doing this it is necessary to think in a different attitude since this is a one-man project (thus descarting other methodologies like *Scrum*) and to change the tutor's role from mentor to a client that asked for this application. Following this, there will be a lot of documentation to be done, demonstrations of the project status and more. Without jumping ahead, the following four different phases for each iteration have been established:

- **Planification:** The first phase of each iteration will consist in preparing the iteration itself: tasks to be done, which documentation will be necessary, what will be needed to present at the end of the cycle... It is the most important part since a good planification will mean that there will not be major delays.
- **Risk analysis:** The second phase checks the potential problems that may appear during the project implementation. It is important to documentate them efficiently in order to avoid, mitigate or totally resolve them before showing an error-prone code to the *client*.
- **Development:** This critical part of the cycle is the one where the code will be generated, meaning that it will need a lot of effort to put through each iteration and will deliver most of the body of this project.
- **Client avaluation:** Final part of the iteration that consists in presenting the project status to the *client* in order to receive feedback and also creating the end-of-cycle documentation.

Also, it is important to note that all the documentation that each iteration generates is incremental, meaning that only the real documentation needed will be generated and that each new version will be updated in the document itself without creating a new one. In order to complete the meta-objective *OBJ-4* two documents will be generated in each sprint (one to document the *Planification* phase and one for the *Client avaluation*) [9] that will be added to mandatory *Progress reports* (an initial one before starting, one for the third iteration and a final for the sixth one) [10], the *System Design report* (that includes all the necessary class diagrams related to the program) [11], the *System Requirements report* (which will include all the logic behind the application and its interactions with the users) [12] and a final *User Manual* [13] that will explain how to use the system from the point of view of all the different user levels.

Finally, there is another point to consider regarding the methodology: where all of this documentation, reports and code will be stored. The short and most efficient answer is using a remote repository like the *ICE-CSIC*'s *GitLab* provided by the tutor. This repository will store all the project contents and will also allow to create new developing branches, whose intra-repository life can be seen in the image 1 and the following list shows their details:

- **master:** Main branch of the application that accumulates all the incremental iteration documentation and the final code versions.
- **System:** Secondary branch that implements all the web development and that was fusionated into *master* at the end of the third iteration.
- **SpreadsheetManipulator:** Another secondary branch whose main function is to store all the main functionalities of the system (order management and original spreadsheet manipulator). It is important to see that this branch develops the core functionality of the application. Merged with *master* in the middle of the sixth iteration.
- **Exportator**: Final branch of the project that includes all the project or message exportation functionalities. Merged with *master* in the last iteration.

After all the details of the methodology, documentation and repository structure have been explained it is time to move on to the main project development but considering that all the analysis, designs and implementations have always strictly had in mind the mentioned methodology in order to create a software as detailed and complemented as a real project would deliver.

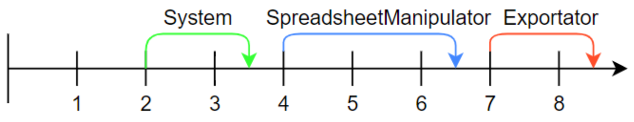


Fig. 1: Branches life cycle through the project's iterations

4 Project development

This section will be divided in the different phases that the project has gone through such as the *Situation analysis and requirements*, the *Design Proposal*, the *Solution implementation* and, finally, the *Results and validation*. Each of them will cover all the information/code that has been generated and how the project grew until its finalization.

4.1 Situation analysis and requirements

As it has been stated in the *State of the art* section, all the project necessities were acquired after multiple meetings with the tutor during the second iteration and a *Use cases and requirements report* was issued. This document contains all the possible interactions between the system and its users and also explains with a lot of details which are the functional (*what* the system has to do) and non-functional (*how* the system has to do it) requirements (with their corresponent flux diagrams for each one) and designing scopes. The functional requirements that were stated and divided through the three main objectives of the project cover the following:

- The system will let the user to view/modify/export a project
- The system will have a login page

- The system will have *Role Based Access Control*
- The system's interface will be a webpage
- The system will feature new languages in the future
- The system will have a user configuration page
- The system will be retrocompatible with old projects
- The system will accept different input format files
- The system will have multi-concurrency
- The system will be modular and able to expand
- The system will let its components to have a description
- The system will feature multiple code parsers and expand them in the future

In the other hand, the non-functional requirements were stablished at the same time and they cover the following application topics:

- The system will encrypt its user/project data
- The system will be programmed in *Python*
- The system will work in a server
- The system's database will be a SQL type
- The system will be available at all times
- The system will work with *CSV* and *Excel* files
- The system will check the role of the user before any step
- The system will have a word dictionary for parsing projects

Even though all this information can be consulted in that report, there are two requirements that are common through all the objectives and are really worth mentioning and explaining individually because they shape the whole project: the user permission matrix and the modularity requirements:

- **User matrix:** This matrix contains all the possible actions that an user can do in the system, meaning that the full project design and implementation will revolve around it to ensure security and privacy as much as posible. Table 1 clearly shows them (some actions like changing user name or profile pic have been excluded since they are not critical elements. Also, the *Visitor* role has been excluded since it only can view assigned projects) and how they are distributed through the different systems roles that were explained before.
- **Modularity:** This is the most important requirement of all that were obtained as it expresses the need of a modular project to easily expand it and add more functionalities in the future. This means that all the different parts of the system (webpage, message manager and exporter) will need to consider that users will modify them, so it is necessary to design the application with this in mind and wider the project scope.

TABLE 1: USER MATRIX AND ROLES

	Admin	Leader	Editor
View assigned projects	✓	✓	✓
View non-assigned projects	✓		
Edit assigned projects	✓	✓	✓
Edit non-assigned projects	✓		
Create new projects	✓	✓	
Modify project members	✓	✓	
Create new users	✓		
Modify users and permissions	✓		

4.2 Design Proposal

After obtaining all the requirements the project moved on the designing phase. During this period the application was designed in four different levels that, when united, give a complete and global vision of the system. Those levels are the following:

- **Frontend level:** This level contains all the views and possible interactions that the application can offer to the user. Since this is a user-related environment, this was the first to be designed since the rest will depend of its conclusions. In order to design it the first step was to create a paper prototype that simulated the final product and was given to the tutor to try it and receive feedback. After some corrections and iterations over this demonstration, a final web design mutual accord was achieved. Being the frontend design proposal finished, then the project moved forward to design its counterpart: the backend.
- **Backend level:** The backend level comprises all the entities that, apart from the data persistance, conform the system and that the user can not interact with. The main goal of this phase is to achieve a design that includes the final decisions of both final web design mutual accord and the technologies to use (seen before in the *State of the art* section) so, for this reason, a new report of the *Software Design* was created in order to documentate the whole process and results. Although the report itself contains more information about the decisions made, the most notable higlights are the following:
  - Program structure: The final structure of the backend is made of multiple modular classes that implement each one of the main functionalities of the application. Being so, the system's web will have its own directory were it will be programmed and the same goes for the *message manager* and the *extractor*. For those two last elements it is worth mentioning that, even though they will share data between them, their functions and methods are higly encapsulated granting that the *minimize sharing between different modules* development principle is achieved [14]. This principle is also obtained by another one of the main motives for doing this project: the data integrity. This means that, as any user must have

access to the latest data on the server side, every time he wants to perform an action the application must consult the server and this leads to an isolation of each module united solely by the database. Image 10 of the appendix will show the current infrastructure conceptual diagram status in more detail.

  – Class diagrams: Complementing the program structure, a series of class diagrams were made to ensure an earlier documentation of what of each modules had to contain from the beggining. Those are included in the *Software Design* report and are higly detailed: they name which classes will exist for every module, their respective functions and their entring parameters. Finally, and as it can be expected, they have been modified and improved along the project progressed but they have maintained both objective and independency between each other class.
  – Permanent files: There will be permanent files that will not be modificable by an user despite its role. Those file are, for example, the *project templates* that will be used during the creation of a new project and will be placed together in the same directory of the backend. Even though they are not a real functional part of the backend, is it important to keep them inbetween it and not in the database to ensure a fast order process for any operation done with them.

- **Persistence level:** The *SQL* project's database will contain all the information required to operate the system. This includes all the project's data (stored as *.csv* type files), the users and its parameters, the project meta-information, the existing parameters for all projects... As it can be seen in the image 11 of the annex that represents a diagram of the database, there will be one table for each one of the possible information datasets to prevent a major information loss in case of an incident. It is important to also explain that the tables will be connected through some common parameters that in most cases will be the *project* or *order* unique identifier.
  One final needed point to clarify is the data migration: in order to migrate all the structure to the actual server that will alocate the project, all flags containing information of the local development environment have been deleted and the database is even ready for remote use outside the *ICE-CSIC*.

Having explained in detail the major decisions taken during the design proposal, it is time to move on the next part that will explain how the design went from an idea to a reality: the *Solution implementation*.

4.3 Solution implementation

This section will explain how the most time consuming part of the project was organized and done. Most of the major ideas (specially the ones related with the documentation process and iteration outputs) has been already mentioned in other parts of this reports so this section will only cover the implementation process as itself and its results that can be validated (i.e. the application code). Being this mentioned, this section will be divided in two parts: *Implementation methodology* and *Implementation*

4.3.1 Implementation methodology

To implement this project's design ideas and maintain the integrity of the project methodology, every iteration has two documents (the *Start* and *End* of iteration reports) that include all the tasks that need to be done to accomplish an objective of the project. Those tasks have been kept in track in the middle of each cycle through using virtual *Task Managers* like *Trello* to see if the system's implementation was getting delayed or not. Another tool used to control those possible project delays is a series of *Gantt Charts* that allowed to see the status at the long term and rearrange the tasks during the various incidents that happened while developing the project, specially in the later iterations (4 to 8) where it really helped organizing tasks (this *Gantt Chart* can be found in the second annex represented by the image 19).

As it can be seen in the tree view of the image 12 found in the second annex, this workspace also contains a set of directories that store all the necessary files to execute and bring up the webpage alive, creating a project master folder directory that stores all the different modules (which are consequently in their own directories: *Spreadsheet-Manipulator* for **OBJ-2** and *Exportator* for **OBJ-3**) and the rest of necessary components that links them. Another important aspect about the implementation methodology is the recurrent use feedback of previous iterations to increase the output's quality. All this feedback was received with meetings with the tutor that also helped to rearrange tasks or refocus on previously considered as done to improve some points that lacked complexity or detail.

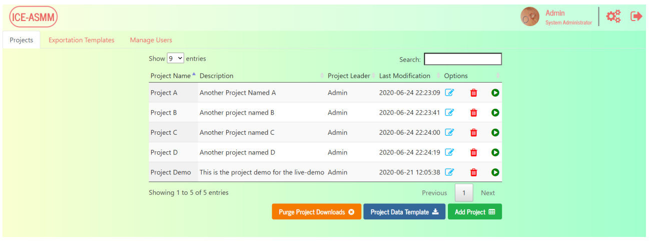
Finally, and about the code writting itself, the methodology to implement new methods has been the process known as *TDD* (*Test Driven Development*) that has consisted in creating the test cases for each different method before coding it. This alternative way of programming derives in a more consistent and already tested solution that benefits both usability and code readability (allowing to reduce comments too). An important detail about the test that have been done is that they are not related to the webpage's interface or the database itself since both of them are better tested with *manual testing* in the *manual test campaigns* that have been done before finishing any objective or task to ensure its completness and bugless behaviour.

4.3.2 Implementation

In order to see the implementation and the created items that the user will interact with (in other words the *frontend* since the user will not have access to the *backend* code that has also been developed), this section will be divided in the three main functionalities of the system that show more clearly *what* has been implemented : webpage, message management and code exportation.

- **Main webpage:** From the point of view of the user this item is the main *User Interface* and system point of entry to the message manager and code exporter. It consists in a series of integrated secondary pages that show to the user the results of the processes of the backend and it has three major parts:

- User system: This comprises both *Login* and *User Preferences* pages, respectively seen in the images 13 and 14 of the second annex. Those two screens allow the user to log in into the system and to configure its own preferences (such as profile pic, username, password...) and have a generic behaviour as any other Internet site.
- Functionality tabs: The main page of the system features three principal tabs visible for all users. The first one is the *Projects Tab* that lists the projects that the user is part of (image 2) and, the second one is the *Code exporter* tab that lists the current parsers that the system has (image 15 found in the second annex). As it can be seen, both tabs features buttons to add, modify or delete projects/parsers and methods to manually purge the system downloads and logs (to increase its workflow in case of a puntual increase in the services demand) and download their respective uploading templates (so the user can upload new projects and code parsers in a faster and more reliable way).

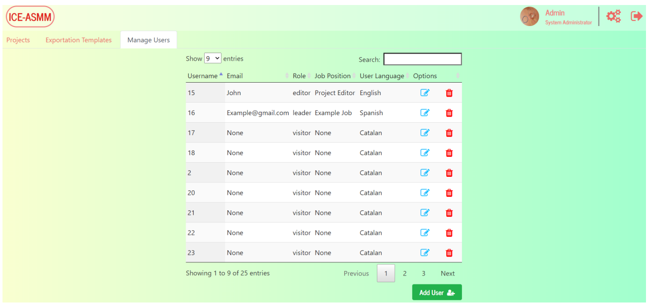


Project Name	Description	Project Leader	Last Modification	Options		
Project A	Another Project Named A	Admin	2020-06-24 22:23:09	✎	🗑	▶
Project B	Another project named B	Admin	2020-06-24 22:23:41	✎	🗑	▶
Project C	Another Project named C	Admin	2020-06-24 22:24:00	✎	🗑	▶
Project D	Another project named D	Admin	2020-06-24 22:24:19	✎	🗑	▶
Project Demo	This is the project demo for the live-demo	Admin	2020-06-21 12:05:38	✎	🗑	▶

Showing 1 to 5 of 5 entries

Fig. 2: View of the *User projects* page

- User Management tab: This last tab (image 3) is only visible to the system administrator team and allows to manage the system's users. There is a list of all the existing users and it features options to create, modify or delete them. Also, the administrator can set new passwords (or prompt the user to change it) from this window.

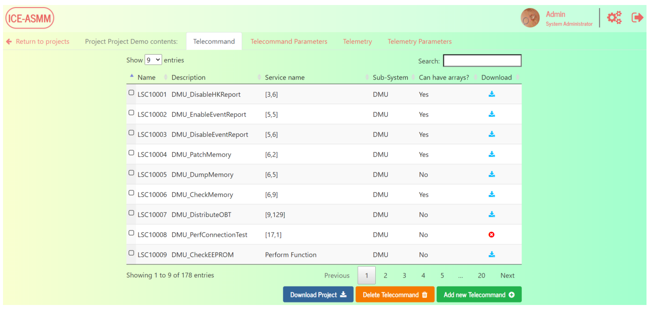


Username	Email	Role	Job Position	User Language	Options	
15	John	editor	Project Editor	English	✎	🗑
16	Example@gmail.com	leader	Example Job	Spanish	✎	🗑
17	None	visitor	None	Catalan	✎	🗑
18	None	visitor	None	Catalan	✎	🗑
2	None	visitor	None	Catalan	✎	🗑
20	None	visitor	None	Catalan	✎	🗑
21	None	visitor	None	Catalan	✎	🗑
22	None	visitor	None	Catalan	✎	🗑
23	None	visitor	None	Catalan	✎	🗑

Showing 1 to 9 of 25 entries

Fig. 3: View of the *User management tab* page

- **Message management:** From the point of view of the user this item (image 4) can be accessed from the *View Projects tab* mentioned earlier. It will consist in a series of tabs that will display all the contents of a project (being so its *Telecommands*, *Telecommand Parameters*, *Telemetries* and *Telemetry Parameters*) and show only the information that a user really requires without any of the *"distractions"* that the original spreadsheets had. As it is displayed in the second annex's image 16, the system allows the user to add or delete multiple messages at the same time through a series of modals that act as a configuration *wizard*. Regarding the message parameters, they have also the same mechanics and logics as their original message counterpart and have a new window to see the different variables that they contain (image 17 of the second annex). Finally, there is a download icon for every telecommand or telemetry that has already defined parameters that will guide the user to the exporter module.

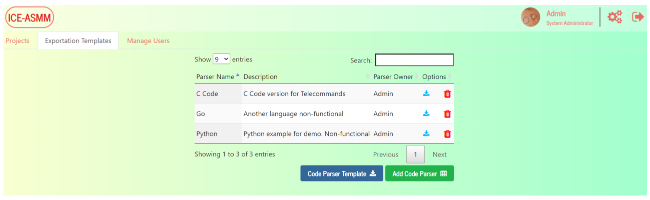


Name	Description	Service name	Sub-System	Can have arrays?	Download
LSC10001	DMU_DisableHKReport	[3,6]	DMU	Yes	⬇
LSC10002	DMU_EnableEventReport	[5,5]	DMU	Yes	⬇
LSC10003	DMU_DisableEventReport	[5,6]	DMU	Yes	⬇
LSC10004	DMU_PatchMemory	[6,2]	DMU	Yes	⬇
LSC10005	DMU_DumpMemory	[6,5]	DMU	No	⬇
LSC10006	DMU_CheckMemory	[6,9]	DMU	Yes	⬇
LSC10007	DMU_DistributeOBT	[9,129]	DMU	No	⬇
LSC10008	DMU_PerfConnectionTest	[17,1]	DMU	No	✖
LSC10009	DMU_CheckEEPROM	Perform Function	DMU	No	⬇

Showing 1 to 9 of 178 entries

Fig. 4: View of the *Project view* page

- **Code Exportation:** This item implements the exporter module functionality and, from the point of view of the user, it is divided in two different parts: the *Code exporter tab* that was mentioned in earlier points and the download message page:
  - Code exporter tab: This tab is implemented in the main page of the program and it contains a list with all the parsers that the system currently has. As it is seen in image 5, all the parsers can be downloaded to check what output they will produce and any user can upload a new parser through a templated method. Those parsers will contain a template of the code to export a message to and they will have commands that start with *ASMM* at the places that the parser's creator desires to output a part of the message (for example, the *ASMM-MSG-NAME* will place the message's command name in that position of the code).
  - Download message page: This page can be reached after selecting a message (with already setted up parameters) download icon in the *Message management*. As second annex's image 18 shows, the user can simply just select the code that he wishes to download the message as without needing to input anything more. This automatic process was not a feature in the original system, meaning that this new upgrade simplifies the required steps to export messages.



Show 9 entries   Search:

Parser Name	Description	Parser Owner	Options
C Code	C Code version for Telecommands	Admin	
Go	Another language non-functional	Admin	
Python	Python example for demo. Non-functional	Admin	

Showing 1 to 3 of 3 entries   Previous 1 Next

Fig. 5: View of the *Exportation tab* page

4.4 Results and validation

After explaining *how* the system has been implemented and *what* has been implemented, this section will provide the different results and validations that this project has produced and gone through. In order to do this, this part has been divided into two different subsections that will cover the project's improvements over the initial situation and the test campaigns and final user testing results:

4.4.1 Project improvements

The implemented application represents a huge improvement over the original situation since it has a more *user-friendly* message manager system and includes new features that the initial situation did not have. A part from the graphical user interface that has been created through a webpage, the centralization of all the project data and contents, the new message exportation system and the newly created project viewing methods has also led to a more practical and convenient environment that has reached completely all the different requirements that were established at the beggining of the project. Regarding those mentioned requirements, they all have been successfully finished through the different iterations, meaning that the system is the exact result of what it was designed for and leaving room for the expected improvements and future work lines that will be explained in the next section.

Another important point to discuss are the results itself. These results must answer a simple question: *How is the new system better than the original?*. The best way to answer to it is considering three different points: a comparison between the old and actual system in terms of process, code exportation and the flaws that they have.

- **Process comparison:** The old system had a noticeable problems related to the data cramping and creation and management of messages. Those led to an overcomplicated process that resulted in investing more time checking that created message was correct than creating the message itself. A user had to check multiple factors like if any of the unique values (like the ID, message/parameter name, variables...) had already been used, if the message had to be divided into blocks due its size, the relations with other possible messages of the same type...
On the other hand, the current situation has improved a lot and turned into a *plug-&-play* system that simplifies the process. As it can be seen in image 6, the number of steps to create a new message has been reduced and also creates a process-safe environment where any input the user makes is verified for him in order to prevent the tedious field by field comparison between similar messages. Another

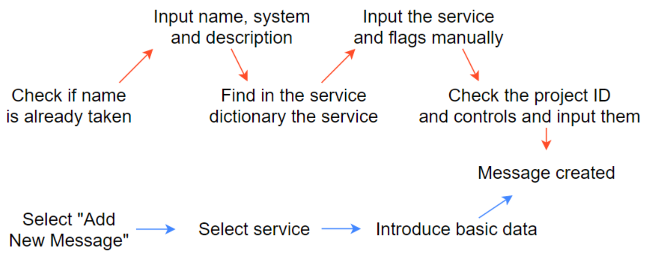

Check if name is already taken
Input name, system and description
Find in the service dictionary the service
Input the service and flags manually
Check the project ID and controls and input them
Message created
Select "Add New Message"
Select service
Introduce basic data

Fig. 6: Comparison between message creations steps (old system in top, new system at bottom)

important improvement regarding the processes is the message management as a whole. In the initial state a user had to check if it was safe to upload a new version of the message sheet to prevent an accidental over-writting of another user's new data. The newer application already checks it and improves both privacy and integrity of a project and its messages.

So, in terms of process, it is clear that the new application has really improved over the initial situation and turned the message developing environment into an easier and more reliable process that greatly simplifies the work of *ICE-CSIC* investigators and developers.

- **Code exportation:** This new feature of the application creates a new system that was one of the biggest problems in terms of efficiency. The original state only allowed to export the messages manually by copying field by field a message's content to the *live-code*, which was prone to errors like misplacements or using wrong data from another telecommand. The new system features an automatic export method to introduce code templates to simplify the user's experience and efficiency while reducing the mentioned common errors. As image 7 illustrates, the new process requires a bit more effort creating the code template but the whole exportation process gets a final boost compared to the original methodology.
So, in terms of *live-code* exportation, the new system makes a huge improvement and resolves one of the most time-demanding and error-prone steps of the process.

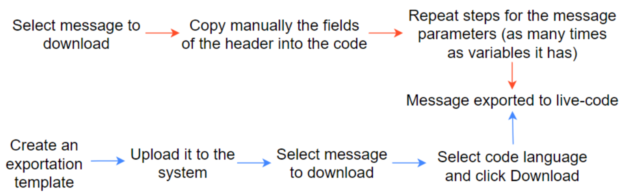

Select message to download
Copy manually the fields of the header into the code
Repeat steps for the message parameters (as many times as variables it has)
Message exported to live-code
Create an exportation template
Upload it to the system
Select message to download
Select code language and click Download

Fig. 7: User management

- **Flaws:** Both processes are far from perfect and have flaws. Despite the new application covers a vast majority of the oldest method's flaws, there is still some points that the new system has to cover. Some of them are related to the application itself like its presentation (functionality was preferred over a aesthetic environment) and performance (in heavy-loaded networks it may have complications) but the most important one is the message creation: despite being a huge improvement over the initial state, the users still need to input a lot of data. A way to solve this could be *import individual messages* or *copy message* features that give more options regarding this issue that could be a future work line. Despite those, it is important to remark how this application has fulfilled its creation goals and the gains win over the flaws it may have.

4.4.2 Test campaigns and final user testing

This section will explain the results of the project from a more technical and statistic point of view in the first part and will go for the user's point of view on the second part to see what do they think about the application.

- **Project metrics:** In a more technical detail, the metrics of the project can be classified in a time aspect from the performance and response time that the system requires to load or execute a new order. As table 2 shows, the time needed to load a project's content needs the highest amount of time while all the other functions have very similiar loading time ratios. The only time that needs a higher time is when loading a project with a lot of data, meaning that this is a point to improve in future iterations of the application (as it will be explanined in the *Future Work Lines* section). A part from this issue, all the times fall into the expected time-zone established in the requirements, meaning that project works as expected.

TABLE 2: APP. RESPONSE TIME (IN SECONDS)

Project loaded size	Low (5 MB)	High (30 MB)
Initiate system	0.4	0.6
Open project	1.5	6
Add new message	1.2	1.4
Add new parameter	0.9	1.1
Delete/Modify item	0.7	0.9
Add new parser	1.2	1.4
Modify users	0.6	0.6
Export project	1.3	1.6
Export message	0.4	0.7

- **User testing:** After the application has been implemented and gone through several tests campaigns, another test with the final user has been done to check if it works as expected. On June 25th, Víctor Martín (a worker of the *ICE-CSIC* and co-tutor of this project for that organization) tried the application from the point of view of an admin of the system and, after going through a basic routine (log in the system, creating a new project with new telecommands and parameters, uploading a new parser and exporting a message) the results were conclusive and the system successfully passed the user testing. Some minor issues that he found about *UI design* and exporting functionality were corrected after the meeting but, overall, he considered that the application fulfilled its duty and worked as expected in the beginning without any major flaw, meaning that the developing phase could be finished in that point.

5 CONCLUSIONS

The conclusions extracted after completing the project are very diverse and touch multiple ambits. For this reason, they will be divided in different parts to firstly explain the *Project's results conclusions*, followed by the *Developing process conclusions*. Finally there will be a *Future work lines* that will expand possible future expansions of this project.

5.1 Project's results conclusions

The best conclusion that can be obtained about the project results is that this project has been completely fulfilled and all the main objectives have been achieved. This is a major point since it demonstrates that all the planifications and documentation have been useful and an important part of the project. About the delivered projected, the application has fulfilled the expectatives of the *"clients"* and, even though it has some minor flaws, it solves the performance and data cramping that the older system created had and extended the capabilities of the possible code exportation. This can be added to the webpage creation that was developed with some of the current trending *frameworks* from zero. All in all, it can be assumed that this project has improved both the work quality for everyone in the *ICE-CSIC* that works with those messages and the student's general knowledge of the current work, web and developing tendencies.

5.2 Developing process

The whole developing procedure has allowed the student to apply for the first time and in a semi-realistic environment all the knowledge obtained during both *Computer Science* degree and the *Enginyeria del Software* mention after 4 years. This has clear implications in the project (since it fulfills the meta-objective **OBJ-4**). A part from this aspect there is also another important point that has been learned during the development phase: the importance of documentation. Documentation is always one of the points that suffers most in any project and most people directly ignore it. Developing in a controlled iterative environment has stated clearly how important is to documentate from the beggining of the process to obtain the best possible outcome, being an example of its utility the project traceability and task-tracking that has been done through all the project.

Another important achieved conclusion regarding the process is the utility of the *Spiral* developing process in project with only one developer. This iterative-based process has really shinned through the whole project and showed its true potential when it comes down to revisions, planification (specially through Gantt's diagrams) and creating code of good quality. It also helped incrementing the contact with the *"client"* (the tutor) to receive constant feedback from an external source of the developing team.

5.3 Future work lines

Even though that this project is completed as it is and is ready for delivery and installation, there are some parts that can me expanded in the future. Thanks to the modular nature that was postulated during the *Design Phase* any desired new feature can be added without affecting the rest of the system. Being that said so, there are some points that this project has not touched and that they could be added in the future to make it more versatile:

- **Use of *Django*:** *Django* is a powerful web developing *framework* that *Flask* looks up to it. Since the later is the system that has been used to implement this project and has shown some flaws, it is logical to assume that with a minor rework of the code and integration *Django* the webpage would be more stable.
- **Improve local and database storage:** The actual system heavily relies on the database performance and the default file explorer of the computer that its running the program. This may lead to a general slow down of the system as its databases becomes more populated and a slower response time of the webpag. Due to this reason, another point to explore would be using non relational databases (like *No-SQL*) and improving the file location in the executing system to prevent this issue at the long term.
- **Binary message decoder:** The last feature that seems interesting to introduce is a binary message decoder that translates a message from its raw format (a binary sequence that may come from the satellite or control centre) to *user-readable* text or viceversa.

Dissertation acknowledgments

This project could not be completed without the help and advises of Lluís Gesa, the first tutor and a very skilled and friendly professional that unfortunately left us during the final part of this project. He was really appreciated by all of his students and we hope the best for his family that he loved to mention in his classes. There is also a big thanks for the actual tutor, Alicia Fornés, for getting involved and helping this project to pull through difficult times and to Víctor Martín for its user testing and recommendations. A final shotout must be made for my family and Marta Montpeyó for supporting me during the whole process. Thanks a lot for all your help and cooperation!

References

[1] "Institute of Space Sciences" ice.csic.es https://www.ice.csic.es/ (accessed June, 28, 2020).

[2] ESA-ESTEC, Requirements and Standards Division and ECSS Secretariat "Ground Systems and operations - Telemetry and telecommand packet utilization", ISSN: 1028-396X, European Cooperation for Space Standardization (ECSS), 2003.

[3] "The European Space Agency" esa.int http://www.esa.int/ (accessed June, 28, 2020).

[4] A. Grau. "Informe Inicial del TFG", Unpublished Manuscript, Universitat Autònoma de Barcelona, March 2020.

[5] "Meet Django" djangoproject.com. https://www.djangoproject.com/ (accessed June, 28, 2020).

[6] "Flask web development,one drop at a time" flask.palletsprojects.com. https://flask.palletsprojects.com/en/1.1.x/ (accessed June, 28, 2020).

[7] D. Feinleib "Big Data and NoSQL: Five Key Insights" forbes.com. https://www.forbes.com/sites/davefeinleib/2012/10/08/big-data-and-nosql-five-key-insights/#5e9a9f444245 (accessed June, 28, 2020).

[8] ISO and IEC "ISO-IEC/27002 Second Edition", ISBN-13: 1028-396X978-9267107189, January 2013.

[9] A. Grau. "Informe Inici/Final d'Iteració 1-8", Unpublished Manuscripts, Universitat Autònoma de Barcelona, March to June 2020.

[10] A. Grau. "Informe de Progrés 1-2", Unpublished Manuscripts, Universitat Autònoma de Barcelona, April to June 2020.

[11] A. Grau. "Informe de Disseny del Software", Unpublished Manuscript, Universitat Autònoma de Barcelona, April 2020.

[12] A. Grau. "Documentació de funcionalitats, diagrames de flux i requisits del projecte", Unpublished Manuscript, Universitat Autònoma de Barcelona, April 2020.

[13] A. Grau. "Manual d'usuari d'ASMM", Unpublished Manuscript, Universitat Autònoma de Barcelona, June 2020.

[14] "Introducció a l'Enginyeria del Software", Internal manuscript of the subject *Enginyeria del Software*, Universitat Autònoma de Barcelona, September 2017.

[15] CCSDS Secretariat, Office of Space Communication and National Aeronautics and Space Administration "Space Packet Protocol - Blue Book", Public Code: 133.0-B-1, Consultative Committee for Space Data Systems (CCSDS), September 2003.

Packet Header (48 Bits)							Packet Data Field (Variable)			
Packet ID				Packet Sequence Control		Packet Length	Data Field Header (Optional) (see Note 1)	Application Data	Spare	Packet Error Control (see Note 2)
Version Number (=0)	Type (=1)	Data Field Header Flag	Applica-tion Process ID	Sequence Flags	Sequence Count					
3	1	1	11	2	14					
16				16		16	Variable	Variable	Variable	16

Fig. 8: Packet Structure for the *ECSS-E-50* protocol

CCSDS Secondary Header Flag	TC Packet PUS Version Number	Ack	Service Type	Service Subtype	Source ID	Spare
Boolean (1 bit)	Enumerated (3 bits)	Enumerated(4 bits)	Enumerated (8 bits)	Enumerated (8 bits)	Enumerated (n bits)	Fixed BitString (n bits)
					Optional	Optional

Fig. 9: Detail of the Packet data field for the *ECSS-E-50* protocol

APPENDIX

A.1 The ECSS-E-50 protocol

The *ECSS-E-50* protocol was created in 2003 by the *ESA-ESTEC*, the *ESA*'s Requirements and Standards Division and *ECSS Secretariat* in order to regulate the *end-to-end* transference of data packets between a satellite and its ground control centre. Before explaining it with more detail, its necessary to define the two possible types of data packets that are transfered (which, from now on, will be refered to as *messages*):

- **Telecommand:** Message that contains orders for the satellite to execute. They are sent from the control centre
- **Telemetry:** Message that contains the results of a satellite order execution or its status. They are sent to the control centre from the satellite.

Both kind of messages will have a series of parameters associated that will contain all the necessary data to create orders for the satellite or interpretation of the results ground control centre, but this will be referenced later on in this appendix.
Before explaining the structure of the messages, it is important to mention another order-execution related parameters the services and subservices:

- **Service:** 8 Bits code number that identifies a range of functions that the on-board satellite's computer can execute. For example, one service could be "Turn on a light" or "Start sensors".
- **Subservice:** 8 bits code number that identifies an unique function of the whole service set. Following the before-mentioned example, one subservice for "Turn on a light" could be "Turn on lightbulb in color red" and another for "Start sensors" could be "Start thermometer calculation".

As it will be explained later on, the process to transfer a message will be sending a series of intermitent binary structures (just like in the famous *UDP* protocol) that contains the instructions to execute. In other words, the control centre just needs to fulfill the fields of the messages to completely control a satellite (and this is just where the developed *ASMM* application enters: it controls how the messages are created and manages their data fields).

Finalizing this section of the annex, and after having considered all those important terms and processes, all messages will have a multi-field structure with two major segments (one of 48 Bits known as *Packet Header* and another of a variable length known as *Packed Data Field*) that will contain the communication information and the message's data respectively. The first part of the structure, which will be common for both telecommands and telemetries, can be seen in the image 8 and it consists of the following fields:

- **Packet ID:** This 16 Bits field uniquely identifies a message. In order to do this, it needs the following sub-fields:
  1. Version Number: Code version that shows if the message is a variation of another. This protocol demands it to be 0 at all times
  2. Type: Code with the type of the message (0 for telemetries and 1 for telecommands)
  3. Data Field Header Flag: Indicates if the message has a parameter associated. This protocol demands it to be 1 at all times
  4. Application Process ID: Also known as *APID*, it is testination application code that it is on-board of the satellite. It is unique and mission-specific
- **Packet Sequence Control:** This 16 Bits field controls the message behaviour in case that it is too big to be sended in one data packet. It consists of two different parts:

1. Sequence flags: Also known as *Grouping flags* for telemetries, they mark the message's position in all of the sent messages
2. Sequence Count: Unique message counter that identifies it in the communication's network. It will be unique for every different project and it is related to the *APID*

- **Packet Length:** Number of data octets that are present within a message data field to calculate its size

On the other hand, the second part of the structure will be consequently divided in the following fields:

- **Data Field Header:** This variable-sized field will contain all the message's data and will be discussed after the header since it has some minor differences between telemetries and telecommands.
- **Application Data:** This is another variable-sized that contains all the application's information between itself. It manages to ensure a connection to a satellite and has information of its *ports* and *firewalls*.
- **Spare:** Variable number of 0s that are used pad a message until it reaches its maximum size.
- **Packet Error Control:** This last variable-sized field contains the error code that it is to be transmited to the control centre or the satellite, in case that the operation or order failed to reach its destination or could not be executed.

After having explained the contents of the message header, it is time to move on to describe more thoroughly the *Packet Data Field* (image 9) which will contain all the message's parameter data:

- **CCSDS Secondary Header Flag:** This flag denotates that the current message's data is part of the *CCSDS* standard [15]. This protocols demands it to be 0 since it is not contemplated that any message it is part of it.
- **TC Packet PUS Version Number:** Version control of the data message packet. In this case it is setted for a telecommand message but it works the same way for the telemetries.
- **Ack:** This 4 Bits field is only used in telecommands to make the satellite update its status to the control centre while executing an order. Depending of its contents it may send a signal when it has accepted an order, starts executing it, it has been executing it for some time or has completed the execution. In the case of the telemetries it will always have four 0s to show that it is empty.
- **Service Type:** Field that contains the service type that was explained before.
- **Service SubType:** Field that contains the service subtype that was explained before.
- **Source ID:** Field that indicates the source of the telecommand (for example the control centre identification). In the case of the telemetries, it is know as *Destination ID* and contains the identification of the receiver.

- **Source ID:** Field that indicates the source of the telecommand (for example the control centre identification). In the case of the telemetries, it is know as *Destination ID* and contains the identification of the receiver.

A.2 Figures and diagrams

This second section of the appendix will contain all the different images and diagrams that are mentioned through the report and that they have not been directly included in it because their content is barely readable without the proper size.

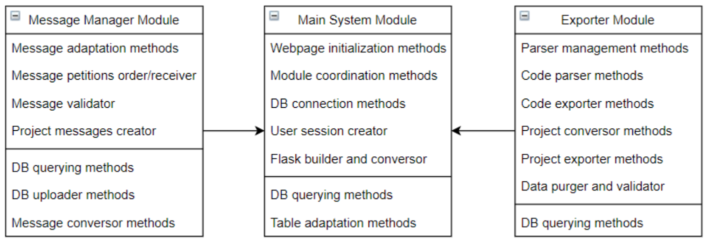


Fig. 10: Backend conceptual structure diagram

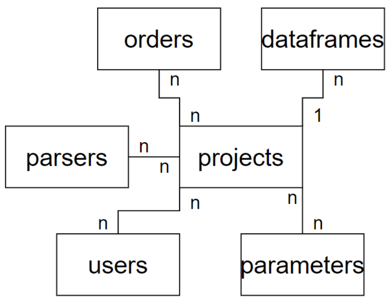


Fig. 11: Simplified view of the database's diagram

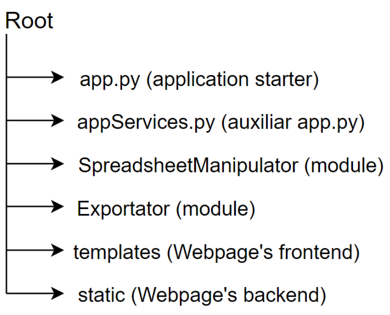

Root
app.py (application starter)
appServices.py (auxiliar app.py)
SpreadsheetManipulator (module)
Exportator (module)
templates (Webpage's frontend)
static (Webpage's backend)

Fig. 12: Structure of the main workspace and brief explanation of the modules

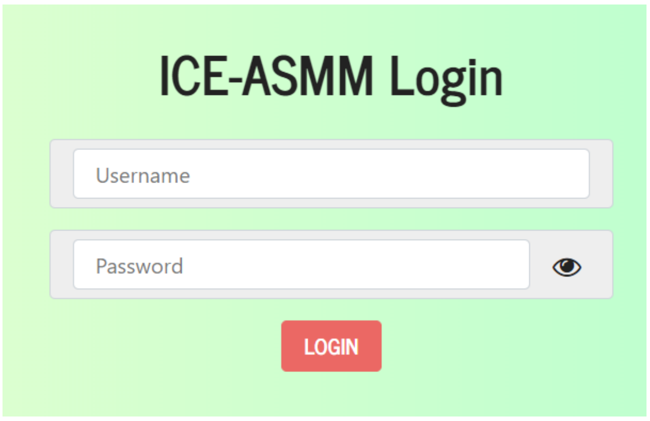
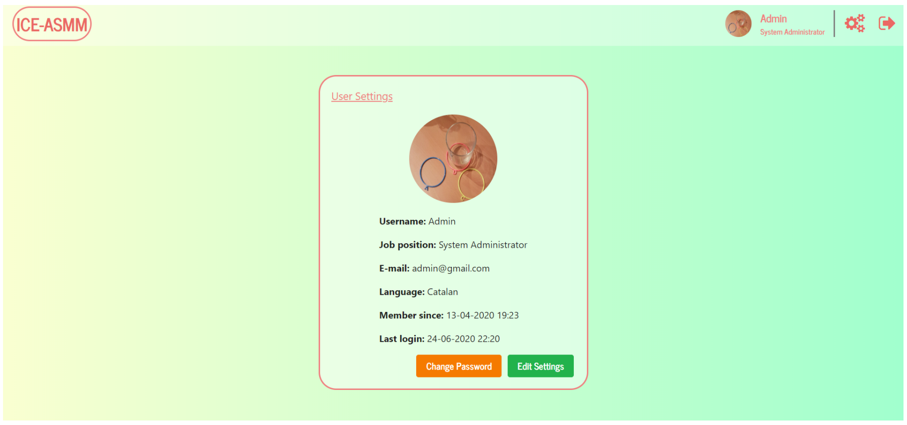


Fig. 13: View of the *Login* page



Username: Admin

Job position: System Administrator

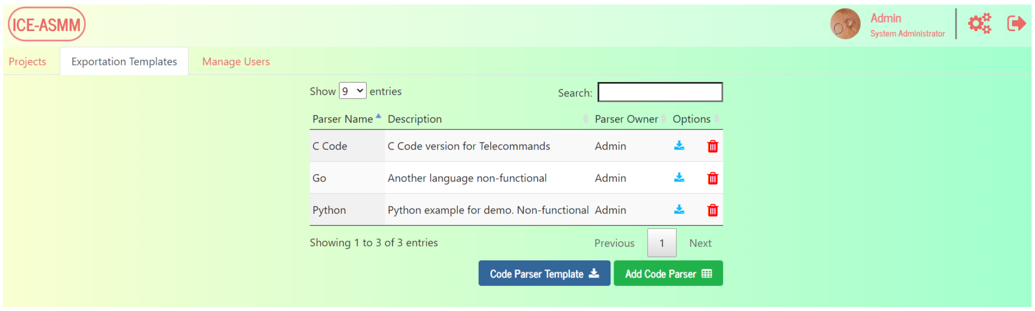
E-mail: admin@gmail.com

Language: Catalan

Member since: 13-04-2020 19:23

Last login: 24-06-2020 22:20

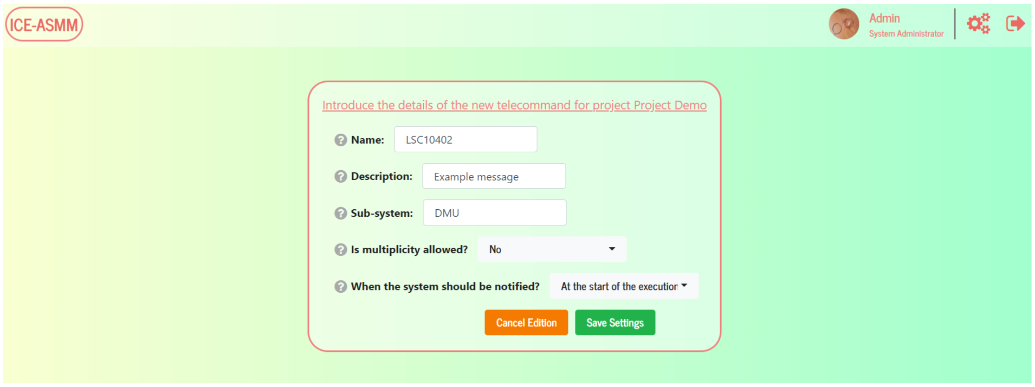
Fig. 14: View of the *User preferences* page



Parser Name	Description	Parser Owner	Options
C Code	C Code version for Telecommands	Admin	
Go	Another language non-functional	Admin	
Python	Python example for demo. Non-functional	Admin	

Showing 1 to 3 of 3 entries

Fig. 15: View of the *System parsers* page



Introduce the details of the new telecommand for project Project Demo

Name: LSC10402

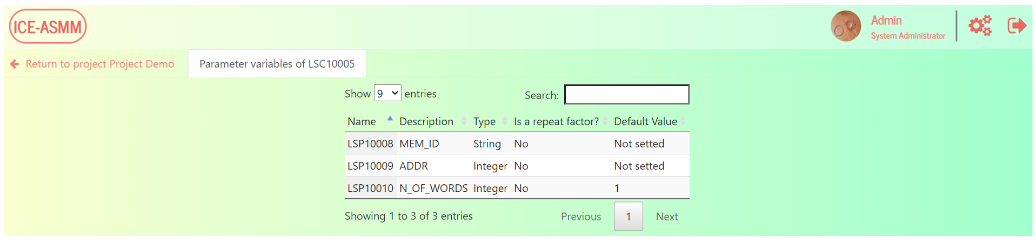
Description: Example message

Sub-system: DMU

Is multiplicity allowed? No

When the system should be notified? At the start of the executior

Fig. 16: View of the *Message Creation* page

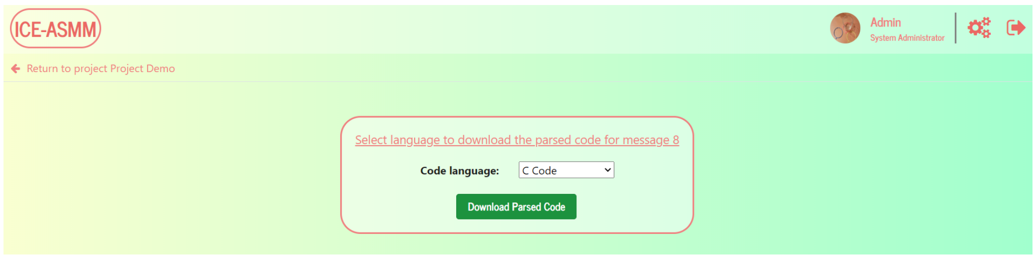

ICE-ASMM
Admin
System Administrator
Return to project Project Demo
Parameter variables of LSC10005

Show 9 entries Search:

Name	Description	Type	Is a repeat factor?	Default Value
LSP10008	MEM_ID	String	No	Not setted
LSP10009	ADDR	Integer	No	Not setted
LSP10010	N_OF_WORDS	Integer	No	1

Showing 1 to 3 of 3 entries Previous 1 Next

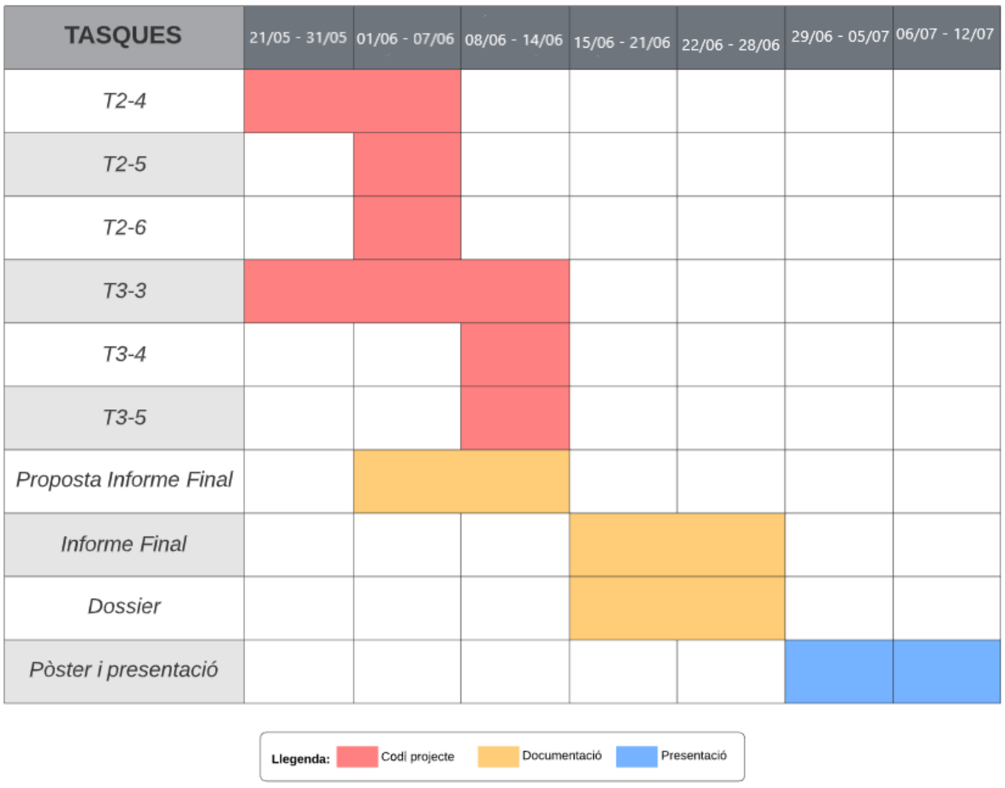
Fig. 17: View of the *Parameter view* page



Select language to download the parsed code for message 8

Code language: C Code

Fig. 18: View of the *Exportation code* page



TASQUES	21/05 - 31/05	01/06 - 07/06	08/06 - 14/06	15/06 - 21/06	22/06 - 28/06	29/06 - 05/07	06/07 - 12/07
T2-4	🟥	🟥					
T2-5		🟥					
T2-6		🟥					
T3-3	🟥	🟥	🟥				
T3-4			🟥				
T3-5			🟥				
Proposta Informe Final		🟨	🟨				
Informe Final				🟨	🟨		
Dossier				🟨	🟨		
Pòster i presentació						🟦	🟦

Llegenda: 🟥 Codi projecte 🟨 Documentació 🟦 Presentació

Fig. 19: View of the *Gantt Chart* of the latest iterations (4 to 8)