Prepared by **L. M. Gagliardini**
Reference
Issue **1**
Revision **0**
Date of Issue **11/10/2018**
Status **Issued**
Document Type **TN**

European Space Agency
Agence spatiale européenne

Distribution

1 ABSTRACT

The present document summarises the work carried on through the 6 months going from the 9th of April to the 12th of October 2018, developed at ESA's European Space Operations Centre (ESOC) as the Final Project of Thesis concluding the Aerospace Engineering Master of Studies at the Politecnico di Torino.
It is given, as first, a wide introduction on the environment involving the thesis project, the Copernicus programme, the involved missions, with a specific description upon Sentinel-2 (in terms of satellite's topics, location and mission's objectives) and its operations. Is then exposed the reasons for introducing the REALS software and the automation of commands to face the needs.
The first part of the main document's body contains a detailed description of the path followed to implement the Remote Alert System (REALS) software, touching its sub-parts, their adaptation and customization, to reach the proper exploitation of the mission's needs.
The second part of the project concerns the automated commands and focusses on the modifications that leaded to the result.
The conclusions are then reported to give a clearer idea of the actual status of the mission's operations.

2 DEFINITIONS & ABBREVIATIONS

[To be filled]

3 INTRODUCTION

3.1 Copernicus Programme

3.1.1 *What is Copernicus?*

Copernicus, previously known as GMES (Global Monitoring for Environment and Security), is the European Programme for the establishment of a European capacity for Earth Observation. Copernicus brings together communities from across the environmental scientific spectrum, and delivers operational services, which range from Arctic sea ice monitoring to emergency response, through oil spill detection and monitoring of urban sprawl. Copernicus services are based on information from a dedicated constellation of satellites, known as “Sentinels”, as well as tens of third-party satellites known as “contributing space missions”, complemented by “in situ” (meaning local or on-site) measurement data.

3.1.2 *Actors and Roles in the Copernicus Programme*

How does it work?
The Copernicus programme is supported by a family of dedicated, EU-owned satellites, the Sentinels, specifically designed to meet the needs of the Copernicus services and their users. Since the launch of the first of these, Sentinel-1A in 2014, the Union set in motion a process to place a constellation of more than a dozen satellites in orbit over the course of the next ten years. Copernicus also builds on existing space infrastructure: satellites operated by the European Space Agency (ESA), the European Organisation for the Exploitation of Meteorological Satellites (EUMETSAT), the EU Member States and other third countries and commercial providers. These are known as contributing missions, and have provided satellite data for the programme since its inception.
What is behind Copernicus?
As a publicly-funded European Union programme, Copernicus is first and foremost the property of all European citizens, who remain its ultimate owners and beneficiaries. The European Commission is in charge of setting and developing the political vision of the Copernicus’s programme and of putting in place the elements which allow the smooth and proper functioning of the system. The development of the space component, including the launch of the dedicated Sentinel satellites, has been delegated to ESA, which also acts as the overall systems architect of the Space Component and ensures its technical coordination. The operations of the Sentinels have been entrusted to ESA and to EUMETSAT, on the basis of their specific know-how. The Sentinels are owned by the European Union. The European agencies involved are the European Environment Agency (EEA), the European Agency for the Management of Operational Cooperation at the External Borders of the Member States of the European Union (FRONTEX), the European Maritime Safety Agency (EMSA) or the European Satellite Centre (SatCen).

3.1.3 *Benefits and Applications*

The Copernicus Services transform this wealth of satellite and in situ data into value-added information by processing and analysing the data, integrating it with other sources and validating the results. Datasets stretching back for years and decades are made comparable

and searchable, thus ensuring the monitoring of changes; patterns are examined and used to create better forecasts, for example, of the ocean and the atmosphere. Maps are created from imagery, features and anomalies are identified and statistical information is extracted.
These value-adding activities are streamlined through six thematic streams of Copernicus services:

- Atmosphere Monitoring
- Marine Environment Monitoring
- Land Monitoring
- Climate Change
- Emergency Management
- Security

Atmosphere Monitoring
The Copernicus Atmosphere Monitoring Service provides the capacity to continuously monitor the composition of the Earth's atmosphere at global and regional scales. This service capacity encompasses the description of the current situation (analysis), the prediction of the situation a few days ahead (forecast), and the provision of consistent retrospective data records for recent years (re-analysis). The service generates geophysical products which require further technical processing and various forms of high level information to support decision makers. The main areas the Copernicus Atmosphere Monitoring Service focuses on are:

- Air quality and atmospheric composition
- Ozone layer and ultra-violet radiation
- Emissions and surface fluxes
- Solar radiation
- Climate forcing

Marine Environment Monitoring
The Copernicus Marine Environment Monitoring Service provides regular and systematic information about the physical state and dynamics of the ocean and marine ecosystems for the global ocean and the European regional seas. This data covers analysis of the current situation, forecasts of the situation a few days in advance and the provision of retrospective data records (re-analysis). The Copernicus Marine Environment Monitoring Service calculates and provides products describing currents, temperature, wind, salinity, sea level, sea ice and biogeochemistry. These factors support marine and maritime applications and related EU policies, e.g. in the fields of:

- Marine safety
- Marine and coastal environment
- Marine resources
- Weather, seasonal forecasting and climate

Land Monitoring
The Copernicus Land Monitoring Service provides geographical information on land cover, land use, land cover-use changes over the years, vegetation state or water cycle.

Applications that are uilt upon and integrate the information supplied by the service can provide support in areas such as spatial planning, forest management, water management, agriculture and food security and emergency management, amongst others. Service priorities and their relevance to users are defined and validated by the European Commission and the Member States. The service is implemented by the European Environment Agency (EEA) and the Joint Research Centre (JRC) since 2011.

Climate Change

The Copernicus Climate Change Service is designed to respond to changes in the environment and society associated with climate change. The service will provide information for monitoring and predicting climate change and help to support adaptation and mitigation strategies. It will provide access to several climate indicators (e.g. temperature increase, sea level rise, ice sheet melting, ocean warming) and climate indices (e.g. based on records of temperature, precipitation, drought events) for both the identified climate drivers and the expected climate impacts. The Copernicus Climate Change Service will enter a pre-operational stage by the end of 2017. The operational phase will start before the end of 2018. This pre-operational phase is also supported by a series of projects funded by the EU research framework programme related to climate modelling and observation analyses.

Emergency Management

The Copernicus Emergency Management Service (EMS), managed directly by the European Commission via the Joint Research Centre and DG ECHO, delivers warnings and risk assessments of floods and forest fires and provides geospatial information derived from satellite images on the impact of natural and man-made disasters all over the world (before, during or after a crisis). Through these, it supports crisis managers, civil protection and hydro-meteorological authorities, humanitarian aid actors dealing with natural disasters, man-made emergency situations, and humanitarian crises, as well as those involved in recovery, disaster risk reduction and preparedness activities. As an EU service, the EMS's first priority is responding to EU needs and interests, whether within the EU or abroad. The Emergency Management Service is provided free of charge to authorised users. The service has two main components: Early Warning and Mapping.

Security

The Copernicus Security Service aims to support related European Union policies by providing information in response to the security challenges Europe is facing, namely improving crisis prevention, preparedness and response capacities in the following key areas:

- Support to EU External Actions
- Maritime surveillance
- Border surveillance

3.2 The Sentinels

As a new family of missions, specifically developed by ESA, the Sentinels represent the constellations on which the Copernicus Programme relies for coverage requirements, providing robust datasets for Copernicus Services. These missions carry a range of

technologies, such as radar and multi-spectral imaging instruments for land, ocean and atmospheric monitoring:

- Sentinel-1 is a polar-orbiting, all-weather, day-and-night radar imaging mission for land and ocean services. Sentinel-1A was launched on 3 April 2014 and Sentinel-1B on 25 April 2016. Both were taken into orbit on a Soyuz rocket from Europe's Spaceport in French Guiana.
- Sentinel-2 is a polar-orbiting, multispectral high-resolution imaging mission for land monitoring to provide, for example, imagery of vegetation, soil and water cover, inland waterways and coastal areas. Sentinel-2 can also deliver information for emergency services. Sentinel-2A was launched on 23 June 2015 and Sentinel-2B followed on 7 March 2017.
- Sentinel-3 is a multi-instrument mission to measure sea-surface topography, sea- and land-surface temperature, ocean colour and land colour with high-end accuracy and reliability. The mission will support ocean forecasting systems, as well as environmental and climate monitoring. Sentinel-3A was launched on 16 February 2016 and Sentinel-3B will join its twin in orbit on 25 April 2018.
- Sentinel-5 Precursor, also known as Sentinel-5P, is the forerunner of Sentinel-5 to provide timely data on a multitude of trace gases and aerosols affecting air quality and climate. It has been developed to reduce data gaps between the Envisat satellite – in particular the Sciamachy instrument – and the launch of Sentinel-5. Sentinel-5P was taken into orbit on a Rockot launcher from the Plesetsk Cosmodrome in northern Russia on 13 October 2017.
- Sentinel-4 is a payload devoted to atmospheric monitoring that will be embarked upon a Meteosat Third Generation-Sounder (MTG-S) satellite in geostationary orbit.
- Sentinel-5 is a payload that will monitor the atmosphere from polar orbit aboard a MetOp Second Generation satellite.
- Sentinel-6 carries a radar altimeter to measure global sea-surface height, primarily for operational oceanography and for climate studies.

3.3 Sentinel-2

3.3.1 *Satellite's Description*

Each of the Sentinel-2 satellites weighs approximately 1.2 tonnes. Both have been launched with the European launcher VEGA. The satellite lifespan is 7.25 years, which includes a 3 month in-orbit commissioning phase. Batteries and propellants have been provided to accommodate 12 years of operations, including end of life de-orbiting manoeuvres. Two identical Sentinel-2 satellites operate simultaneously, phased at 180° to each other, in a sun-synchronous orbit at a mean altitude of 786 km. The position of each Sentinel-2 satellite in its orbit is measured by a dual-frequency Global Navigation Satellite System (GNSS) receiver. Orbital accuracy is maintained by a dedicated propulsion system.
The MSI works passively, by collecting sunlight reflected from the Earth. New data is acquired at the instrument as the satellite moves along its orbital path. The incoming light beam is split at a filter and focused onto two separate focal plane assemblies within the

instrument; one for Visible and Near-Infra-Red (VNIR) bands and one for Short Wave Infra-Red (SWIR) bands . The spectral separation of each band into individual wavelengths is accomplished by stripe filters mounted on top of the detectors.

3.3.2 *Mission's Objectives*

The Sentinel-2 mission comprises a constellation of two identical polar-orbiting satellites placed in the same orbit, phased at 180° to each other. It aims at monitoring variability in land surface conditions, and its wide swath width and high revisit time (10 days at the equator with one satellite, and 5 days with 2 satellites under cloud-free conditions which results in 2-3 days at mid-latitudes) will support monitoring of changes to vegetation within the growing season. The coverage limits are from between latitudes 56° south and 84° north.
The objectives of Sentinel-2 are set to provide:

- Systematic global acquisitions of high-resolution, multispectral images allied to a high revisit frequency
- Continuity of multi-spectral imagery provided by the SPOT series of satellites and the USGS LANDSAT Thematic Mapper instrument
- Observation data for the next generation of operational products, such as land-cover maps, land-change detection maps and geophysical variables.

These high-level objectives will ensure that Sentinel-2 makes a significant contribution to Copernicus themes such as climate change, land monitoring, emergency management, and security. With its 13 spectral bands, 290 km swath width and high revisit frequency, Sentinel-2's MSI instrument supports a wide range of land studies and programmes, and reduces the time required to build a European cloud-free image archive. The spectral bands of Sentinel-2 will provide data for land cover/change classification, atmospheric correction and cloud/snow separation.

3.3.3 *The Sun-Synchronous Orbit*

The Sentinel-2 mission orbit is sun-synchronous. Sun-synchronous orbits are used to ensure the angle of sunlight upon the Earth's surface is consistently maintained. A Sun-synchronous orbit (SSO, also called a heliosynchronous orbit) is a nearly polar orbit around a planet, in which the satellite passes over any given point of the planet's surface at the same local mean solar time. More technically, it is an orbit arranged so that it precesses through one complete revolution each year, so it always maintains the same relationship with the Sun. Apart from small seasonal variations, anchoring of the satellites orbit to the angle of the sun minimises the potential impact of shadows and levels of illumination on the ground. This ensures consistency over time and is critical in assessing time-series data. Sentinel-2A and Sentinel-2B occupy the same orbit, but separated by 180 degrees. The mean orbital altitude is 786 km. The orbit inclination is 98.62° and the Mean Local Solar Time (MLST) at the descending node is 10:30 (am). This value of MLST was chosen as a compromise between a suitable level of solar illumination and the minimisation of potential cloud cover. The MLST value is close to the local overpass time of LANDSAT and almost identical to that of SPOT-5, permitting the integration of Sentinel-2

data with existing and historical missions, and contributing to long-term time series data collection.

3.4 The Operations

The European Space Operations Centre (ESOC) serves as the main mission control centre for the European Space Agency (ESA) and is located in Darmstadt, Germany. ESOC's primary function is the operation of unmanned spacecraft on behalf of ESA and the launch and early orbit phases (LEOP) of ESA and third-party missions. The Centre is also responsible for a range of operations-related activities within ESA and in cooperation with ESA's industry and international partners, including ground systems engineering, software development, flight dynamics and navigation, development of mission control tools and techniques and space debris studies. Like many ESA missions, the scientific and operational life of the Sentinel-2 spacecraft begins with the launch and early orbit phase.

3.4.1 *Launch and Early Orbit Phase*

Following separation of the satellite from the Vega launcher upper stage following lift off, an automatic sequence controlled by the on-board software deploys the single solar array and initialises the acquisition of the satellite's 'local normal' pointing, so that it could establish a telecommunication link with the team at ESOC via the ground station network.

- On Day 1, the Flight Control Team switches on all platform systems and makes sure the satellite is basically healthy.
- One important system is GPS navigation; Sentinel-2 must deliver imagery with an accuracy up to 20 meters on the ground, that means engineers must know very precisely where the satellite is at any time. That is achieved using GPS signals. Moreover, the Flight Dynamics team at ESOC will use ranging and Doppler data extracted from the satellite's radio signal to precisely determine the Sentinel-2A orbit prior to the full operation of the on-board GPS receiver.
- On Day 2, the team activates the on-board mass memory unit and starts to retrieve some of the recorded data, and released the shutter lock on the MSI payload instrument that protects the very sensitive detectors from the light of the Sun.
- On Day 3, the satellite performs the trajectory correction manoeuvres, trimming the orbit into which the spacecraft was delivered by the launcher to achieve the final height of 786 km, at an inclination of 98.5°, passing almost directly over the poles.

3.4.2 *Commissioning Phase*

The operational phases of the Sentinels (each mission comprising two spacecraft) foresees a gradual increasing of the ground coverage for instrument observation. For each mission, the ramp-up phase starts upon the launch and commissioning, and continues until Full Operational Capacity is reached. In the Sentinel-2 commissioning phase, lasting about three months from the end of LEOP, the Flight Control Team will be primarily focused on checking out and characterising the performance of all systems including attitude and orbit control, the telecommunication links and thermal and power.

3.4.3 *Routine Operations*

Once in the routine phase, the mission's operations cycle typically includes:

- Orbit maintenance
- Mission planning, including systematic image acquisitions and downlink and activities such as:
    - Multi Spectral Instrument image acquisition.
    - Playback from the MMFU [Mass Memory and Formatting Unit] of MSI and House Keeping Telemetry.
    - X-Band acquisition
    - S-Band acquisition
- Downlink of data via EDRS Performance monitoring

3.5 Thesis Objective

Since the Copernicus Programme involves several missions, for each Sentinel mission a Flight Control Team manages and performs the operations required to accomplish the nominal activities. Each Team is directed by the Spacecraft Operations Manager (SOM), followed by a Deputy SOM, the Spacecraft Operations Engineers (SOE) and a team of spacecraft controllers (SPACONs) shared between all the Sentinel missions. The teams are supported by other specialists at ESOC, including experts from flight dynamics, ground facilities, tracking stations, space debris and mission data systems. As the Sentinel constellation increases, the amount of operations required for the routine activities, thus the amount of work performed by the team, rises and therefore the need for automation aimed at reducing the needed man-work.
The aim of the developed project is to face the automation needs focusing on two different aspects:

- REALS (Remote Alert System) is a software externally developed and implemented in the specific mission's server, in ESOC, which takes input directly from the Mission Control System (SCOS 2000). The software collects and filters specific telemetry packets (i.e. those leading to urgent reactions by the Flight Control Team) and notifies them via SMS, rather than Phone Call, to the On-Call Engineer, in case of serious spacecraft issues. The Sentinel-2 Flight Control Team expressed the will to implement and customize the software for handling telemetry packets concerning specific conditions of the spacecraft itself rather than its subsystems aimed at allowing the On-Call Engineer being informed on the status of the spacecraft whenever specific telemetry packets are received on ground. The big advantage in implementing the REALS software is represented by a 24 hours per day/full coverage from the On-Call engineer which receives detailed information regarding each remarkable issue. Each notification is composed of the needed information for a clear comprehension of the Spacecraft status. Otherwise, in case of packets received during the uncovered hours (with no Engineer or Spacecraft Controller on console), a delay will occur between the packet reception and the engineer awareness as the engineer will need to arrive at the Space Operation Centre (ESOC).

- To reduce the amount of activities of the Spacecraft Controllers, the FCT raised the need to insert more than half of the commands manually released, at the begin and at the end of each spacecraft pass over the ground stations, into the automatic stacks. One of the most demanding operation involving the Engineers and the Spacecraft Controllers is the Mission Planning, used to plan instruments and spacecraft activities. The Mission Planning System is intended to generate the command stacks scheduled for the next 2 weeks of routine spacecraft activities taking source from different input files on the base of the requests and needs coming from the Payload Data Ground Segment, Flight Dynamics rather than third parties. The Spacecraft Controllers are called to interact with the Mission Control System and operate the uplink of the generated stacks. The stacks are divided into three different classes:
  - Automatic Release Based Stack, scheduling the time of the release from the Mission Control System.
  - Automatic Execution Based Stack, released manually, scheduling the on board execution time.
  - Manual Release Stack, whose release is manually performed for each command and the relative execution is scheduled ASAP.

The over mentioned tasks have been developed in ESOC and achieved during the 6 months from the 9th of April to the 12th of October 2018.

4 DEFINITIONS

4.1 What is the Mission Control System (MCS)?

Once the first Acquisition of Signal (AOS) occurs, the data-flow from the spacecraft subsystems starts and the downlink of the telemetry, generated by the several instruments and by the on-board software, and the uplink of the commands can be performed. On the groundside, the engineers and controllers need a software to:

- Receive from the spacecraft and interpret the telemetry
- Send commands to be interpreted by the spacecraft
- Operate Mission Planning
- Maintain the On-Board software through the mission lifetime
- Store of all mission data and operate its distribution

Without sophisticated software, it would be impossible to operate a modern space mission. ESA's mission operations software experts design and develop software systems used in ground segments to carry out satellite control, mission planning, simulations, training and ground station control and management. To allow a reuse of the software through the different mission, standardization is used to keep up to 80% of the software common. The remaining customizable part, up to the 15% of the software, can keep commonalities within the Families Missions:

- Interplanetary Missions
- Earth Observation Missions
- Astronomy Missions

This leaves the 5% of functionalities to be developed from scratch for each specific mission.

4.2 What is the Packet Utilization Standard (PUS)?

The PUS addresses the utilization of telecommand packets and telemetry source packets for purposes of remote monitoring and control of spacecraft subsystems and payloads, during both on-ground testing and in-flight operations. The standard is devoted to define the services between on-board application processes and cooperating applications on the ground, and their associated telemetry and telecommands packets structure. The PUS service, also called Type, in conceived as a set of on-board functions that can be controlled and monitored by several users, such as a ground system, through a defined set of service requests and reports. An on board application process ins defined as the source telemetry packets and the sink for telecommand packets. Example of applications processes is a subsystem, a unit within a platform or a payload. Each application process has a unique identifier called Application Identifier (APID). The identification, selection and standardisation of PUS services were based on the past ESA experience in operating satellites. It has been a long process that culminated in the definition of 17 services. They cover the most frequent needs in the field of space mission operability. Each service is made of several sub-services called sub-type, which can be service requests, service reports or capabilities.

4.3 Remote Alert System Input.

4.3.1 *The Event Logger Application*

As mentioned above, the Mission Control System is composed of different applications handling different telemetry packets and telecommands. The Event Logger Application is a base on which the REALS software relies acting as the primary input source. The Event Logger Application selects the packets from specific services from the telemetry received on ground and lists them into an interface to make them understandable by the Engineers and Spacecraft Controllers. [To Add an image of the interface]

4.3.2 *The ACTION Application*

The ACTION Application function is to check what is handled by the Event Logger Application. In case of specific packets received on ground by the Event Logger, the ACTION Application directs the content of the specific packets through the call of an action (e.g. the execution of a script receiving the packet content as input). The Application configurations are defined into the Action.dat file, described below.

4.4 Text Standardization

In order to allow an easier handling and classification of the files, a standard format to their names is given.
This classification comes from the need to relate the content of a file to the mission it belongs to (i.e. the respective domain in the server) and the way the content is to be handled (i.e. Events rather than OOLs).
It is also convenient to refer to files whose name is related to the calling script's name.
As an example, in such a way, the reals_action_1_ool.sh will take the specifications from its own name (i.e. domain ***1***, handling ***ool***) to point the *whitelist_1_ool.txt* file.
As an output the script will produce a text file whose name is as well defined by domain and information to be handled, *eventREALS_1_ool_###.txt*.
In such a way, there is no need to modify the content of the file but only to detail its name.
Hereafter a detailed explanation follows.
The file names must respect the following standard:

The first part of the name identifies the matter (the content) of the file (i.e. whether it is a whitelist rather than a script).
In case the file is a script, this part must define the functionality (e.g. if the script is called by a cronjob the standard name will figure with *reals_cronjob*, if the script is called to handle text messages the standard will be *reals_action*).

The second part of the name refers to the mission, in terms of server's domain.
If the file refers to all the domains, this specification is not needed (e.g. reals_cronjob.sh).

The third part of the name refers to the information handled (i.e. whether it refers to Events or OOLs).
If the file refers to all the domains, this specification is not needed.

The fourth and last part of the name can be customised depending on the needs and no constraints must be respected.

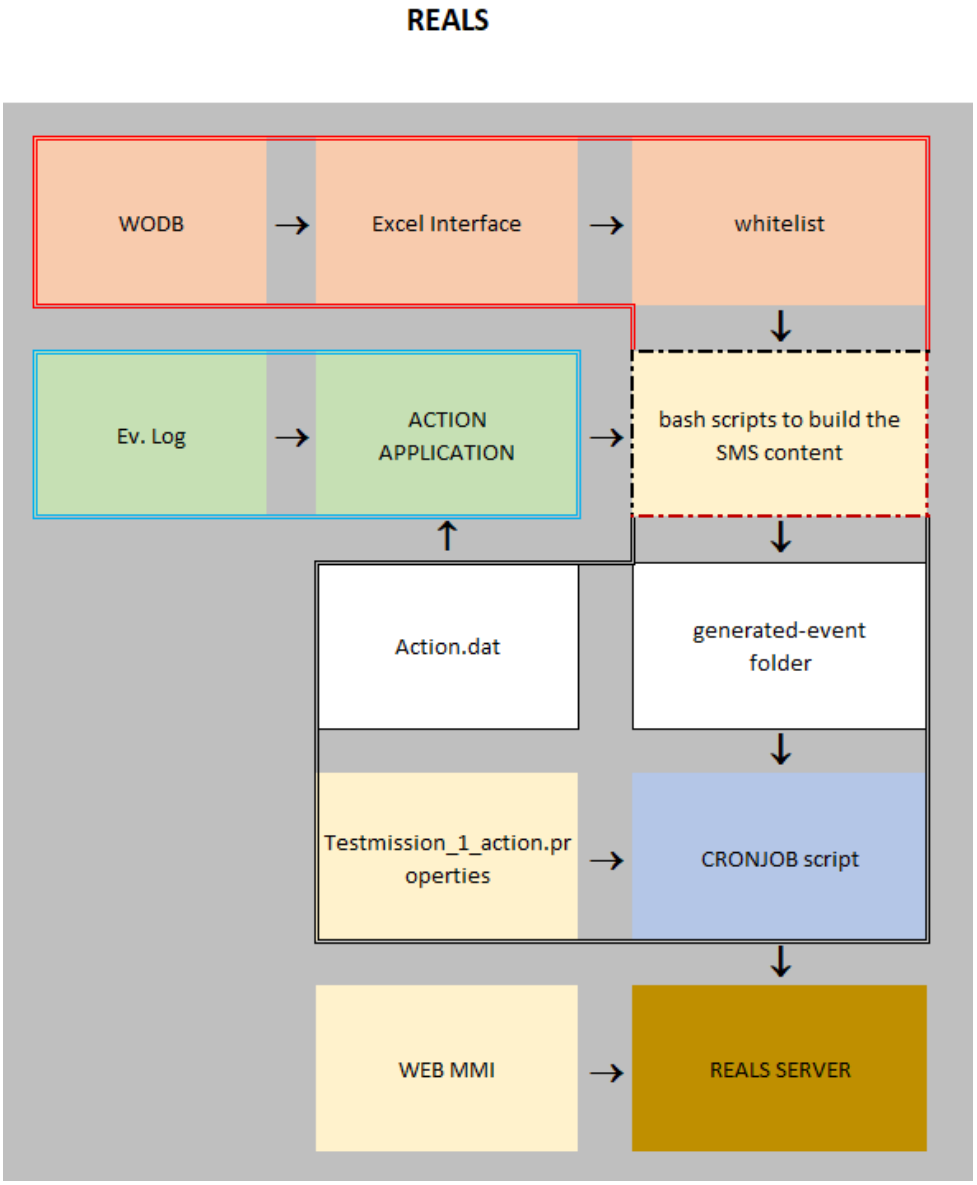
All the fields must be single Underscore separated.
The REALS files are so called:

reals_action	“_”	1	“_”	obe	.sh		
reals_action	“_”	1	“_”	ool	.sh		
reals_cronjob	“_”				.sh		
whitelist	“_”	1	“_”	obe	.txt		
whitelist	“_”	1	“_”	ool	.txt		
eventREALS	“_”	1	“_”	obe	“_”	#date	.txt
eventREALS	“_”	1	“_”	ool	“_”	#date	.txt

5 THE SOFTWARE STRUCTURE

This chapter is intended to give an overview of the logic behind the structure of the REALS software, in order to allow an eventual modification of one or more of the files comprised in the software.
Be aware that all what is contained in this document describes the only parts modified by the user and that have been adapted and configured for the Sentinel-2A operations.
Nothing is treated in this document of how the Events and OOLs are handled in SCOS-2000 (before the reaction of REALS) nor by the REALS server (concerning the work of the software developer).

5.1 Block Diagram



5.2 Action.dat

The very first step is made using the Action.dat file.
The Action.dat file has to be located in the mission's server, under the path specific for the domain (mission) to monitor (i.e. NOT into the ***REALS*** main folder).
Thus, the Action.dat file stored under domain 1 must refer to the files relative to the S2A mission.
As mentioned, the ACTION Application of SCOS-2000 performs this first step.
It takes the input (from the Event Log) and check them against the Action.dat file.
The Action.dat file for S2A structure and functionalities is here detailed:

```
TPKT_OBE::1104      SCRIPT      /home/se2ops/REALS/scripts/reals_action_1_obe.sh
BEHV::10163         SCRIPT      /home/se2ops/REALS/scripts/reals_action_1_ool.sh
BEHVNLimCPB2::10163     SCRIPT      /home/se2ops/REALS/scripts/reals_action_1_ool.sh
BEHVNLimCPB3::10163     SCRIPT      /home/se2ops/REALS/scripts/reals_action_1_ool.sh
```

Three strings separated by single TAB characters.

5.2.1 *Field 1: <APPLICATION::EVID>*

The first string has the format APPLICATION::EV.ID.

5.2.1.1 Application

The application field contains the name of the source application from SCOS, these are:

- TPKT_OBE, is the SCOS application receiving all the On-Board Events and shows them into the Event Logger Interface.

- BEHV (Behaviour Limit Checker), is the application receiving telemetry from the Live Data Stream and checking against the database whether the parameter's value within the packet respects the limits. If the parameter is not respecting the constraints, the BEHV Application gives two different output:
    1. Shows the Parameters with its current value (Out Of Limits) into the OOL Display.
    2. The parameter Out of Limits is shown as an Event into the Event Logger Application, on which REALS relies.

- BEHVNLimCPB2, receives the telemetry from Play Back Data Stream 2 (S-band) and handles it in the same way the BEHV application does.

- BEHVNLimCPB3, receives the telemetry from Play Back Data Stream 3 (X-band) and handles it in the same way the previous two applications do.

5.2.1.2 EV.ID

The EV.ID has nothing to deal with the identifier of the telemetry packet.
It is a field used as reference for several SCOS applications (e.g. the ACTION application) and in this case requires to be filled for the correct functionality of REALS.
The EV.ID must match the field EV_ID in the PID table. For S2 the PID table have been selected ***all and only the services (5, 4)*** and to them was assigned the same Event ID (1104).
For further details, see ***Paragraph 13.4*** concerning the EV.ID 1104 for (5, 4) Events.
For what concerns the Out Of Limits, the HARD Out Of Limits (i.e. those having an impact on the further functionality of the relative instrument) are defined by ID 10163.
Those are the only kind of OOLs defined in the Action.dat since Soft Out Of Limits are not of interest for the Flight Control Team.
This Ev. ID is repeated three times in the Action.dat file, each one depending whether the parameter is within a TM packet downlinked by Data Stream 1, 2 or 3, whose source application is different.
In case a matching Application and EV.ID are found in the Event Log, the rest of the line is processed and the fields two and three are applied.

5.2.2 *Field 2: <ACTION>*

The second field in the line represents the action to perform, in our case it is the call of a SCRIPT (Specified as an Upper case, single word).

5.2.3 *Field 3: <SCRIPT PATH>*

The third field represents the object to which the action refers.
To call a script is necessary to specify its entire server-path.
When, in the Event Log, the Action application of SCOS checks any Service (5, 4) or OOL whose source Application and Ev.ID are reported in the Action.dat file, the relative script is called.
The input of the script is the content of the “Event Message” of the checked Service (5, 4)/OOL in the Event Log (i.e. the description of the Event/OOL figuring in the Event Log).
Depending whether the Event Log raised a (5, 4) Service or an OOL a different script is called (as can be seen above).

5.3 VBA Macro and Excel interface

To let the bash scripts work they need a list of Events/OOLs and relative filtering constraints for parsing the messages from SCOS.
The VBA Macro and the Excel interface furnish this input to the bash scripts, acting as a mid-step between them and the FCT choosing the Events and OOLs to be notified.
A deeper description of the interface is under ***Chapter 10***

The Macro aims also to give a standard format to the Whitelists coming from the FCT, in line with the bash scripts handling them.

5.4 reals_action_1_obe.sh & reals_action_1_ool.sh

As mentioned, if the raised Event/OOL's Application and ID figure in the Action.dat file, the corresponding bash script is called.
The reals_action_1_obe.sh and the reals_action_1_ool.sh, relying on the Whitelists, act as the most specific filter step of the software and their detailed description is in the shared drive.

reals_action_1_obe.sh & whitelist_1_obe.txt
\\ESAAD\esoc\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\Documents\Described_scripts\reals_action_1_obe_whitelist_description.pdf

reals_action_1_obe _bash_script.pdf
\\ESAAD\esoc\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\Documents\Described_scripts\reals_action_1_obe_bash_script.pdf

reals_action_1_ool.sh & whitelist_1_ool.txt
\\ESAAD\esoc\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\Documents\Described_scripts\reals_action_1_ool_whitelist_description.pdf

reals_action_1_ool _bash_script.pdf
\\ESAAD\esoc\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\Documents\Described_scripts\reals_action_1_ool_bash_script.pdf

The file names respect the standardization.
Each of the scripts is intended to be called in case of the raising of, respectively, a (5, 4) Event or an OOL.
As already mentioned, the input to both reals_action_1_obe.sh and reals_action_1_ool.sh is the content of the "Message" column in the Event Log application in SCOS, referring to the Event/OOL raised.
Once the ACTION Application of SCOS, according to the Action.dat file, has passed the Event Message as input to the bash script, the check against the Whitelists as defined by the FCT is made.
The whitelist_1_obe.txt and the whitelist_1_ool.txt contain the constrains for a proper filtering of the incoming Event/OOLs and the information to detail the final content of the notification (SMS).

The script selects and merges the information given by both the Event Message and the Whitelist to create a more specific SMS content helping the On-Call SOE in his work.
Doing this operation the script also updates the Whitelist's constraints for those Events/OOLs that have been raised.
Once the SMS content is created, in line with the constrains given by the Whitelist, it is saved as a text file under the generated-events folder in the server:

home/se2ops/REALS/generated-events

Shown in ***Paragraph 2.1***.
The name of the files generated as output respect as well the standardization:

eventREALS_1_obe_DOY###_h##_m##_s##_mi###.txt

eventREALS_1_ool_DOY###_h##_m##_s##_mi###.txt

5.5 reals_cronjob.sh

The reals_cronjob.sh is a bash script called at the occurrence of a cronjob, as the name can suggest.
It is located into the *scripts* folder into the server, under:

home/se2ops/REALS/scripts

/home/se2ops/REALS/scripts/

Name	Size	Changed	Rights	Owner
..		30/07/2018 10:48:26	rwxrwxr-x	se2ops
Latest_Whitelists		03/07/2018 16:26:46	rwxrwxr-x	se2ops
Moved_out		13/06/2018 10:00:52	rwxrwxr-x	se2ops
old		14/03/2018 13:40:06	rwxrwxr-x	se2ops
To_be_moved_in		14/06/2018 17:05:32	rwxrwxr-x	se2ops
cronjob_status.txt	1 KB	03/03/2017 16:32:21	rwxrwxr-x	se2ops
EVENTDB.txt.ORIG	2 KB	19/05/2017 14:17:13	rwxrwxr-x	se2ops
OOLDB.txt	11 KB	18/05/2017 14:26:08	rwxrwxr-x	se2ops
pcf_main.txt	1 KB	16/05/2018 11:16:11	rw-rw-r--	se2ops
reals_action.sh	5 KB	09/07/2018 16:18:33	rwxrwxrwx	se2ops
reals_action.sh.ORIG.1905	3 KB	19/05/2017 13:41:58	rwxrwxr-x	se2ops
reals_action_OOL.sh	5 KB	09/07/2018 15:13:28	rwxrwxrwx	se2ops
reals_action_OOL.sh.ORIG.1905	3 KB	19/05/2017 13:42:10	rwxrwxr-x	se2ops
reals_cronjob.sh	12 KB	17/07/2018 11:25:52	rwxrwxrwx	se2ops
reals_send_client.py	3 KB	03/03/2017 16:32:21	rwxrwxr-x	se2ops
reals_status.txt	1 KB	03/03/2017 16:32:21	rwxrwxr-x	se2ops
WHITELIST_OBE.txt	4 KB	30/07/2018 17:16:03	rw-rw-r--	se2ops
WHITELIST_OOL.txt	14 KB	30/07/2018 17:16:03	rw-rw-r--	se2ops

At the occurrence of a cronjob (currently each 5 minutes), the script takes the eventREALS files saved into the generated-events folder, containing the Events/OOLs raised by SCOS filtered by the previous scripts, and uses them for the next step.

/home/se2ops/REALS/generated-events/

Name	Size	Changed
..		18/09/2018 14:53:02
eventREALS_1_ool_DOY268_h09_m15_s48_mi811_32511.txt	1 KB	25/09/2018 11:15:48
eventREALS_1_ool_DOY268_h09_m16_s20_mi994_1134.txt	1 KB	25/09/2018 11:16:20
eventREALS_1_ool_DOY268_h09_m16_s45_mi248_28610.txt	1 KB	25/09/2018 11:16:45
eventREALS_1_ool_DOY268_h09_m17_s03_mi875_32384.txt	1 KB	25/09/2018 11:17:03

In this step, the Testmission file assigns enrooting information to each Event/OOL.
Once performed this operation, the same script deletes the eventREALS and restores the constraints to their original value.
Into the "scripts" folder, there is a backup sub-folder containing the last Whitelists version generated by the FCT.

home/se2ops/REALS/scripts/Latest_Whitelists/

After the calling of the Python script, the reals_cronjon.sh restores the whitelist_1_obe.txt and whitelist_1_ool.txt with the back-up versions, overwriting the handled ones from the last modification.

5.6 Testmission_1_Action.properties

The Testmission file is located into:

/home/se2ops/REALS/config/#domain/***Testmission_1_action.properties***

This means that for each new domain (mission) in the server, a new Testmission file must be created respecting its actual structure.
As previously mentioned, this step is not intended to cut out the incoming messages but to create a reference for their routing; indeed all of the notifications, at this point, have to be sent but not all to the same address.
The work of the Testmission file is as follows:

- Match the input coming from files saved into the ***generated-event*** folder.
- Depending on the matched regex, route the message to different addresses.
  From this point is clear that the Testmission file has to be configured depending of the structure of the bash script's output, saved into the generated-event folder.
  Since we are interested on distinguish between a (5, 4) Event, an OOL or "others" (in case the reals scripts were not able to handle the input), three different regex must be defined.

The structure of the file is divided into 4 blocks of code, whose fields are described below:

- The first block groups the fields common for detailing all the incoming messages.
  To configure this file the help of the software developer is needed.
  In this block, we specify the folder created into the REALS server for the new mission.
  The software developer when configuring REALS for the Sentinel-2 missions (one folder for all the S2 domains) created this folder.

```
config.tmQueryServerHost=localhost
config.tmQueryServerPort=12345
config.realsSshServerHost=10.33.202.4
config.realsSshTimeoutMs=2000
config.realsSshServerUsername=reals
config.realsSshServerPassword=C2btoj3ma
config.realsSshDestinationDir=/home/reals/input/SENTINEL_2
config.mission=Sentinel-2
config.domain=1
```

- Into the second block we find the regex to match the input corresponding to the structure of the OOLs whose structure has a date prefix in the format YYYY.DOY.HH.MM.SS.mmm.
  Therefore, the regex figuring in the present block will be:
  (\\d{4}\\.\\d{3}\\.\\d{2}\\.\\d{2}\\.\\d{2}\\.\\d{3}) (.*$\r?\n?)
  Matching a date format specified, followed by any string excluding Carriage Return and Line Feed.
  For the OOLs the other specific fields are defined as follows:

```
action.event.hard.source=se2amce
action.event.hard.source.id=.
action.event.hard.source.type=E
action.event.hard.context.row=: {1} {2}
```

  Where the elements {1} {2} are the first and second sub-strings matched by the regex.
  Into the context.row field, the final content of the notification is specified (this field has not a twin field into the WEB MMI since is not used as a reference for the addressing but to give an content to the SMS).
  Depending on the needs, it is convenient to just take the regex matched text to fill it or, in case of a Phone-Call, define there a different readable message, as shown in the next point.

- Into the third block we find the regex matching the standard message occurring in the need of a Phone-Call:
  Sentinel_requires_call
  This regex matches the message “Sentinel_requires_call” followed by any string excluding Carriage Return and Line Feed.
  For this kind of notifications the other fields are detailed as follows:

```
action.event.hard.source=se2amce
action.event.hard.source.id=.
action.event.hard.source.type=S
action.event.hard.context.row=: Alert from scos. Sentinel 2A requires
actions
```

  In this case the message notified by Phone_Call is not a copy of what is matched by the regex (i.e. {1} {2} as in the previous case) but the standard message “`Alert from scos. Sentinel 2A requires actions`”.

- The third block is meant to direct all the messages left.
  Therfore the regex matches anything but Carriage Return and Line Feed.

```
action.event.hard.source=se2amce
action.event.hard.source.id=.
action.event.hard.source.type=H
action.event.hard.context.row=: {1}
```

5.7 WEB MMI

A good description of the WEB MMI can be found into the documentation for Cryosat-2, since it handles the notifications for all the missions.

The documentation figures under:

\\ESAAD\esoc\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\Documents\Described_scripts\C2-RP-ESC-FS-3130.pdf

From paragraph 2.3 on.

6 OPERATIONAL CONCEPT

6.1 Unmanned passes

Events are currently processed from both data streams, 2 for S-band data and 3 for X-band.
If a hard (red) OOL is detected, or if a severe (5, 4) Event is received, the system will scan the events and OOL periodically (default every 5 minutes) and send an SMS to a preconfigured mobile phone number.
The system can be configured so that a message is not acknowledged back during a specific time period (default setting is 30 minutes).
The acknowledgments (i.e. responses in the REALS terminology) are done via SMS as well.
In case there are many events, it may be that replying to all messages becomes too cumbersome.
In any case, if there are many events it is likely there is a problem serious enough that the SOM needs to be notified as well.
In the case of receiving of many events, it might be confusing to understand to which one we are giving response.
In case of receiving an SMS (or a series of SMSs) the REALS MMI allows two possibilities:

- When a short response is configured (as default) on the MMI, it will be enough to send a no-empty message back to acknowledge the alert.
  Be aware that the response to a series of several notifications start from the last received, unlikely it could be intuitive.
  So if we sort the SMS received by the time we received them, the first response we give is referred to the latest received, the second response to the 2nd latest and so on.
- In case the WEB MMI is configured for specific response, the On-Call SOE will be asked to structure the response SMS following the instructions (here summarized) on the documentation under ***Paragraph*** *Errore. L'origine riferimento non è stata trovata.* ***[RD-4]***.
  The user has several options.
  He can accept the notification using the **PIN** (defined into the WEB MMI for each SOE profile) creating an SMS having the structure: "**ALERT #### *PIN ACCEPT***".
  Here the #### represents the name of the alert received into the incoming SMS.
  Therefore the SOE is allowed to specify the which of the several received alerts is acknowledging.
  Using the same structure the SOE is allowed to ***BLOCKALL***, ***ACCEPTALL***, ***REJECT*** and ***FORWARD*** the alerts.

If no response is given the alert is automatically forwarded to the Engineering figuring in the next position into the notifications list.
If no other engineer figures in the forwarding list, the alert remains pending.
The system will be kept running permanently to avoid configuration changes between passes.
The on call SOE will thus receive SMSs corresponding to the OOLs and events even during manned passes and also the responses follow the same rules.
For these, the SPACON would anyway contact the on call SOE.

6.2 Events to be monitored

The Events and OOLs to be monitored are meant to be selected, by the FCT, directly from the WODB and to be copied and listed in:

\\ESAAD\esoc\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\REALS_FCT_Event_OOL_list.xlsx

The file works as an interface between the FCT and the REALS software.
It has been decided that in the beginning the Events requiring a reaction by the on-call SOE are (5, 4) FMONs and OOLs.
As explained in ***Paragraph 5.3*** , a Macro has been created into the Excel file to take the contents of the lists in the file and to copy and format them into the whitelist_1_obe.txt and whitelist_1_ool.txt in such a way that the reals scripts can read and write them.
Indeed the Whitelists are used as reference by the reals_action_1_obe.sh and reals_action_1_ool.sh for the notification to be forwarded.

6.3 Reaction to REALS Messages

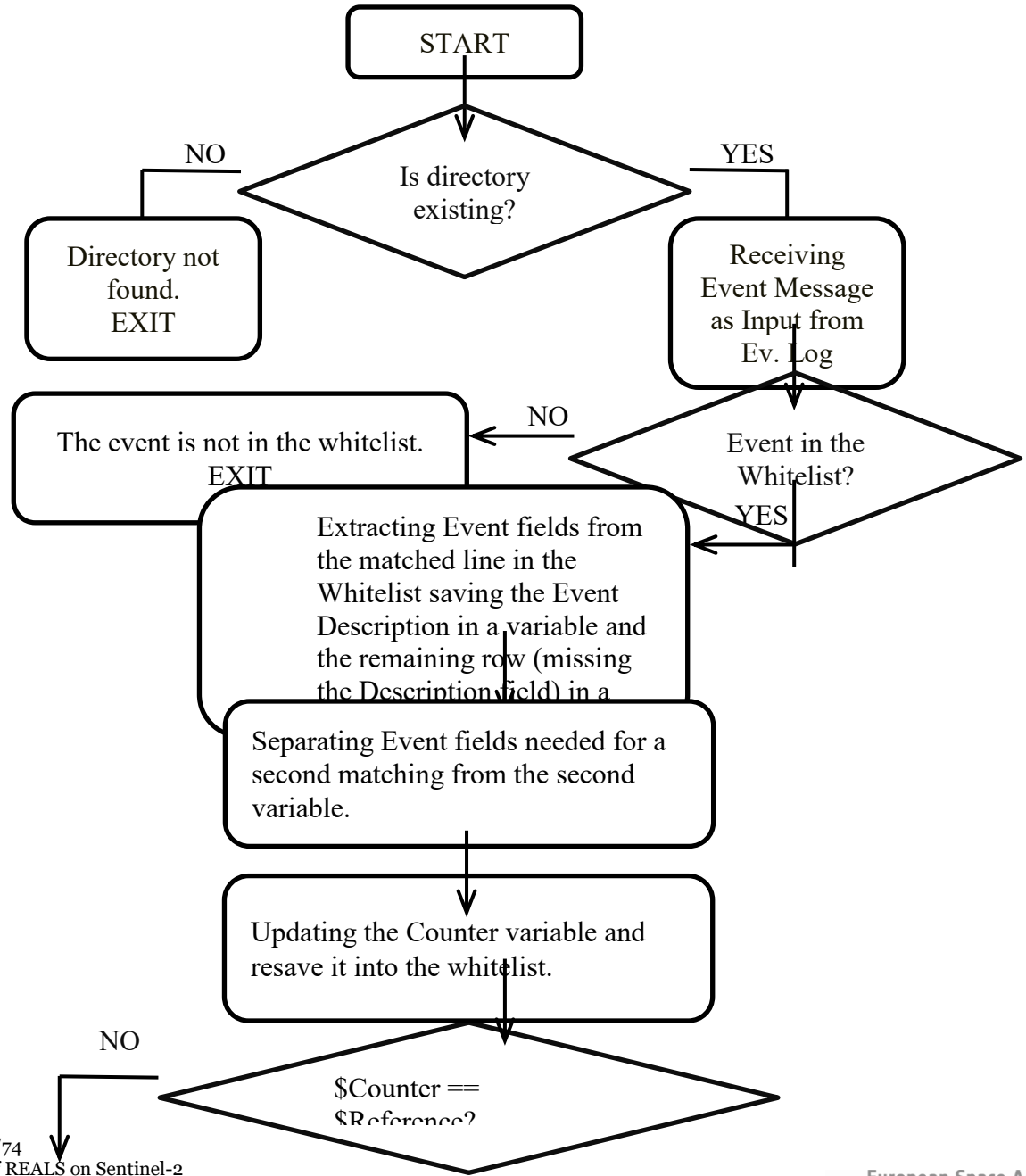
The main guiding principle is for the on call SOE to react to the events with the same effect as if a SPACON was present.
It must however be clear that an unmanned pass, albeit monitored with REALS, is not the same as a manned pass. Systems can fail, GSM network may not be available, etc., so this is considered an extra layer of protection, enabling a quicker reaction in case of serious problems, which otherwise would be reacted upon at the next manned pass.

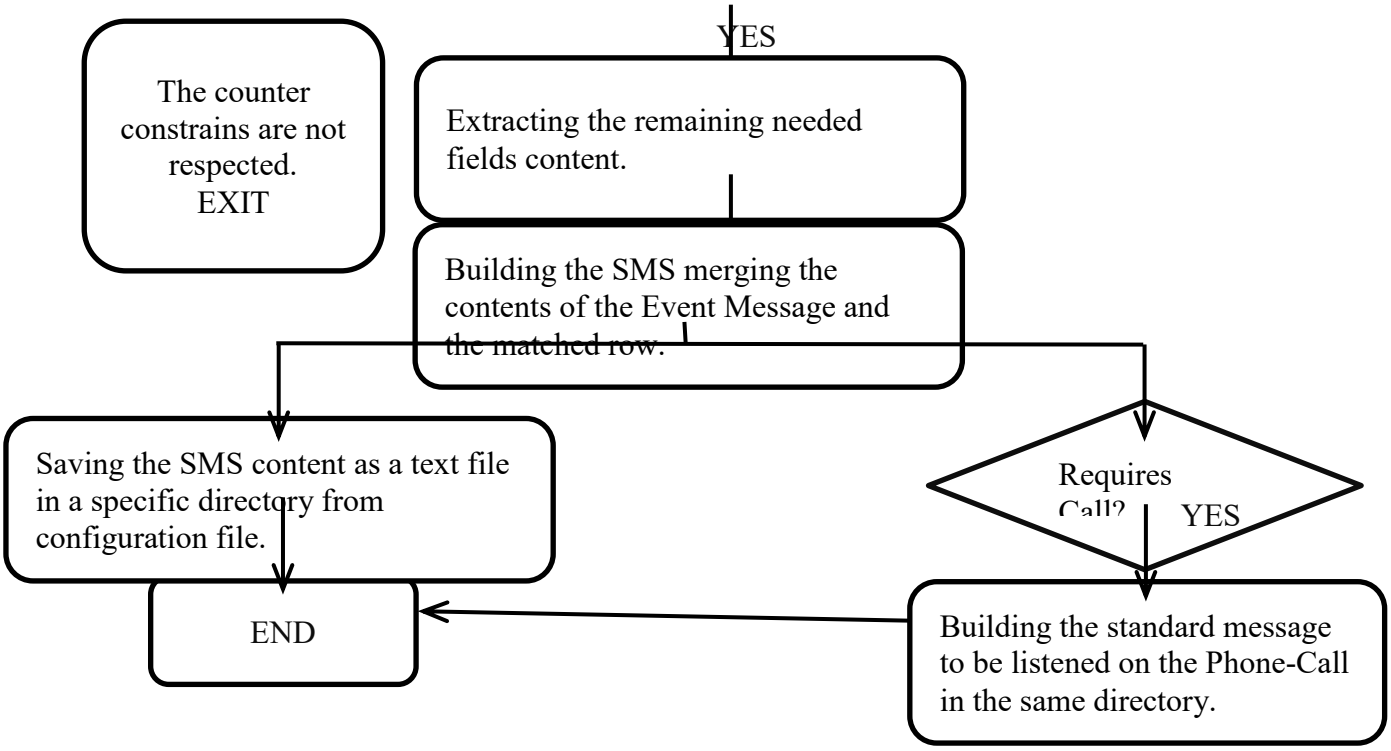
6.4 Managing On-Call SOE Rotation

Since the REALS software is only configured to send SMSs, the need of a phone call was early raised by the FCT.
The LAN phone line in ESOC gives the possibility of receiving a vocal message on the local phone reading the content of the SMS.
It has been decided to choose a local phone number to which all the (5, 4) Events are notified.
From here the vocal message is being forwarded to the On-Call Phone.
In such a way the raising of a (5, 4) Event is notified as a phone call to effectively alert the SOE.
In any case, all the (5, 4) Events and all the OOLs are notified as SMSs to the On-Call Phone to give a more detailed description.

7 REALS_ACTION_1_OBE.SH

The aim of this script is to generate the alerts only for those events respecting specific constrains, expressed by the FCT, in terms of timing references, amount of maximum notifications and so on.
How does it work?
The script takes in input the description (from the “Message” column) of the event raised by SCOS in the Event Log and extracts specific meaningful elements for making a match with the content of a Whitelist in which are specified those events and those constrains to be respected to allow the creation of an alert.
The logical structure
In the following is shown the logical structure that the scripts follows to accomplish its aim.





The logical structure exposed as flow chart is now rewritten as a list of statements.

```
IF [to verify the presence of a file containing the configurations (directories and filenames)]
|
|       <creation of variables containing them>
|
ELSE
|
|       <if cannot find the directories nor the files exit the script>
|
END
<Receiving the “Event Message” as input>
IF [to verify if the event is in the whitelist]
|
|       <Extracting Event fields from the matched line in the Whitelist saving the Event
Description   |       in a variable and the remaining row (missing the Description field) in a
second variable>
|
|       <Separating Event fields needed for a second matching from the second variable>
|
|       <Updating the Counter variable and resave it into the whitelist>
|
|       IF [to verify constrains on the Counter]
|       |
|       |       <Extracting the remaining needed fields content>
```

```
|       |
|       |       <Building the SMS merging the contents of the Event Message and the
|       |       matched row>
|       |
|       |       <Saving the SMS content as a text file in a specific directory>
|       |
|       |       IF [to check if the Event requires a call]
|       |       |
|       |       |       <Building the standard message to be listened on the Phone-Call>
|       |       |
|       |       |       <Saving the Phone-Call content as text file into the same specific
directory of | |       |       the SMSs >
|       |       |
|       |       END
|       |
|       ELSE
|       |
|       |       <The counter constrains are not respected, exit the script>
|       |
|       END
|
ELSE
|
|       <The event is not in the whitelist, exit the script>
|
END
```

The Whitelist structure for OBE
The REALS software relies on several Whitelists defined to specify those OOLs and (5, 4) Events that interest the FCT.
The Whitelists have a double scope:

- The Whitelists are meant to give References and Constraints to the reals_action_1_obe.sh and reals_action__1_ool.sh scripts for the filtering of the incoming messages raised by the Event Log in SCOS.

- The Whitelists contain, for each OOL/Event, Informations for detailing of the notification.

The Whitelist for the OBE (the one we refer to, in this document) is defined for only Events (5, 4), that should require urgent reaction by the FCT.
Both the constraints and the detailing informations can be either numerical values or characters but since UNIX does not distinguish between the two, the difference consists in the way they are handled.
The constraints are all the values used to verify whether a notification is to be crated or not.
Under the *Constraints* of a specific Event we find:

- The Event Description, is the very first constraint since it is used to check whether the event is or not in the Whitelist (first IF condition into the script).
- The *Counter* and the *Reference_counter*, they go together because of their usage as second verification done by the script.
  The *Reference_Counter* is a fixed value and represents the times the event has to be raised to generate an alert.
  The *Counter* is updated each time the event is raised and restored to 1 by a cronjob (5 minutes).
  When its value gets equal to the *Reference_Counter*'s value, the second IF statement is verified (and the first filtering step is passed).
  These are the two main IF conditions, as shown in the logical Structure of the script.
  If all of them are achieved, the SMS text is generated and its content is saved in a specific folder whose directory path is taken from the configuration file figuring into the confing sub-folder into the REALS directory, as described in the TN.
- The Requires_Call flag is used to verify a third optional IF statement, nested into the previous ones, i.e. the need of receiving a Phone Call for the raised Event in addition to the SMS notification.
  If the flag is set to "Y" (Upper case Y), this last IF condition is verified and a second message for the same Event is created.
  This message will be listened when the Phone Call is received message will be handled in a second step by a specific code block into the Testmission file, described into the TN.

Under the *Detailing Informations* of a specific Event we find:

- Type of TM packet.
- Sub Type of TM packet.
- APID
- Event Identifier
- SPID
- Sub System originating the event.

Some of them might be redundant and not all of them will figure in the content of the notification (depending on the engineer's needs).
In such a way it is possible to modify the notification content without changing the whitelist.
Here is an example of the content of the whitelist:

Type	Sub Type	APID	Event ID	SPID	Description	SubSystem	Reference	Counter	R_C	End Line
5	4	151	54689	17823	E_EID_MSIC_FMON_2_MSI_COM_SURVREF1	MSI	1	1	Y	E_L
5	4	151	54690	17824	E_EID_MSIC_FMON_3_MSI_COM_SURVREF2	MSI	1	1	Y	E_L
5	4	151	54691	17825	E_EID_MSIC_FMON_4_MSI_COM_STBYREF	MSI	1	1	Y	E_L
5	3	151	36679	89901	E_EID_MSI_FS_MSI_155_FE_01	MSI	1	1	Y	E_L
5	4	215	54700	26334	E_EID_PFCT_FMON_13_SBS_COM	SBS	1	1	Y	E_L

It's structure is made of a series of rows (one for each different Event (5, 4)) respecting the same pattern.
Each field is separated by a TAB character "\t" and does not contain SPACE characters " ", needed since this Whitelist is formatted in order to be handled by the reals_action_1_obe.sh script.
At this point the "Event Description" field is made of a singles string (not containing SPACE characters " "), as explained into the macro_whielist_description file, at the following link:

\\escwindfs01\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\Documents\Described_scripts\macro_whitelist_description.pdf

Description of the statements in detail
First indentation, IF statement:

```
IF [to verify if the event is in the whitelist]
|
|       <Extracting Event fields from the Whitelist matched line saving the Event
Description   |       in a variable and the remaining row (missing the Description field) in a
second variable>
|
|       <Separating Event fields needed for a second matching from the second variable>
|
|       <Updating the Counter variable and resave it into the whitelist>
|                       .
|                       .
|
ELSE
|
|       <if the event is not in the whitelist exit the script>
|
END
```

The condition enclosed in the red brackets aims to verify IF the content of the message received when the Event is raised (content that should identify the event) figures in one of the rows of the Whitelist.
The command:

event_desc=` echo $@ | sed 's/ /_/g'`

saves the Event Description (received as input taken from the Event Log) in the $event_desc variable substituting each SPACE character " " with an underscore "_".
In such a way the variable contains a singles string and can be compared with the same field into the Whitelist.
The command:

go_ahead=` grep -c -i -m 1 "$event_desc" /home/se2ops/REALS/scripts/whitelist_1_obe.txt`

take the Event Description (as a variable in $event_desc) and gives the number of matched lines as output in the go_ahead variable if the content is matched, using for the comparison the content of the text file whitelist_1_obe.txt.
The flag "-m 1" stops the matching process at the first matched result.
Since in the whitelist there might be repetitions, in such a way the result in the $go_ahead variable is maximum equal to one "1".
In this way the IF condition will only verify the go_ahead variable content being equal to 1.
If the condition is verified, this means that the event raised figures in the Whitelist and the FCT is interested on it.
The matched row is copied into the variable $entire_line and, as was done previously, the output is meant to be only the current matched line "-A 0" and to stop the matching at the first result "-m 1".
In such a way the content of the $entire_line variable avoids to contain a repetition of the matched line itself.

```
entire_line=`grep -A 0 -m 1 "$event_desc"
/home/se2ops/REALS/scripts/WHITELIST_1_obe.txt`
```

The variable $entire_line has the same structure of the matched row in the Whitelist.
The following commands take the respective fields occupying position 9, 8 and 7 in the variable $entire_line representing the $counter, $reference and $SubSystem and saves them into their respective variables:

```
counter=`echo $entire_line | cut -d ' ' -f9`

reference=`echo $entire_line | cut -d ' ' -f8`

SubSystem=`echo $entire_line | cut -d ' ' -f7`
```

Be aware that, reading the Whitelist (text file) UNIX interprets TAB characters as if they were SPACE characters, so the separator character into the cut function is a SPACE.
This command is to update the value of the counter:

```
counter_update=$[1+$counter]
```

This command updates the Whitelist substituting the variable $counter with the variable $counter_update and saves the modifications:

```
sed -i
"s/$event_desc\t$SubSystem\t$reference\t$counter/$event_desc\t$SubSystem\t$refer
ence\t$counter_update/g" /home/se2ops/REALS/scripts/WHITELIST_1_obe.txt
```

The characters used for the separation of the different fields into the Whitelist are TAB characters “\t”.
In the case the IF condition is not respected, the script gives back the message ‘Miss Match’ and terminates the script execution through exit 1.
Second indentation, IF statement:

```
IF [to verify constrains on the Counter]
|
|       <Extracting the remaining needed fields content>
|
|       <Building the SMS merging the contents of the Event Message and the
|       matched row>
|
|       <Saving the SMS content as a text file in a specific directory>
|                       .
|                       .
|                       .
|
ELSE
|
|       <The counter constrains are not respected, exit the script>
|
END
```

The condition enclosed in the red brackets takes as variables the $counter and the $reference to check their values to be equal.
3 different cases can happen:

- $counter < $reference
  In this case the IF condition is not respected and the SMS content is not generated, the counter has been already updated to $counter_update = $[1 + $counter].
  The event has to be raised again by SCOS until the value of the counter reaches the reference one and create the notification content, unless a previous occurrence of a cronjob restores the counter value to one.
- $counter = $reference
  In this case the IF statement is respected, the SMS content is generated and the counter is updated from his current value (equal to the reference) to $counter_update = $[1 + $counter], that will be checked in the next raised Event.
  In this case the remaining fields, only the needed ones, occupying the position 1, 2, 4 and 10 in the variable $entire_line are extracted as exposed in the description of the previous IF statement:

```
requires_call=`echo $entire_line | cut -d '' -f10`

Ev_type=`echo $entire_line | cut -d '' -f1`

Ev_sub_type=`echo $entire_line | cut -d '' -f2`

event_id=`echo $entire_line | cut -d '' -f4`
```

Type	Sub Type	APID	Event Id	SPID	Event Message	SubSystem	Reference	Counter	R_C	End Line
5	3	151	36679	89901	E_EID_MSI_FS_MSI_155_FE_01	MSI	1	1	Y	E_L

The fields that are not extracted will figure only in the $entire_line variable and are not needed.
Now the SMS content can be generated merging the informations from both the Event Description and the Whitelist into the $full_event variable:

```
full_event="$Event_desc |Event: ($Ev_type, $Ev_sub_type)| ID: $event_id| $event_desc| SubSystem: $SubSystem"
```

- $counter > $reference
  In this case the IF condition is not respected and the SMS content is not generated, the counter has been already updated to a higher value.
  Since this point, the counter and the reference will never be equals and only the eventual occurrence of a cronjob, restoring the value of the counter to 1, will allow the cycle to restart.

The SMS content is then saved as a text file following the directory specified in the very first if statement (as shown in the flow chart).

```
temp_filename="$EVENT_FILE_PATTERN$domainName_$now_$RANDOM.txt"

cd $REALS_EVENTS
```

Where the variable $REALS_EVENTS contains the directory

```
echo $full_event > $temp_filename
```

and $temp_filename the name of the text file to be saved.
In the cases the IF condition is not respected the script gives back the message ‘Reference != Counter’ and terminates the script execution through exit 1.
Third indentation, IF statement:

IF [to check if the Event requires a call]
|
| <Building the standard message to be listened on the Phone-Call>
|
| <Saving the Phone-Call content as text file into the same specific directory of the SMSs >
|
END

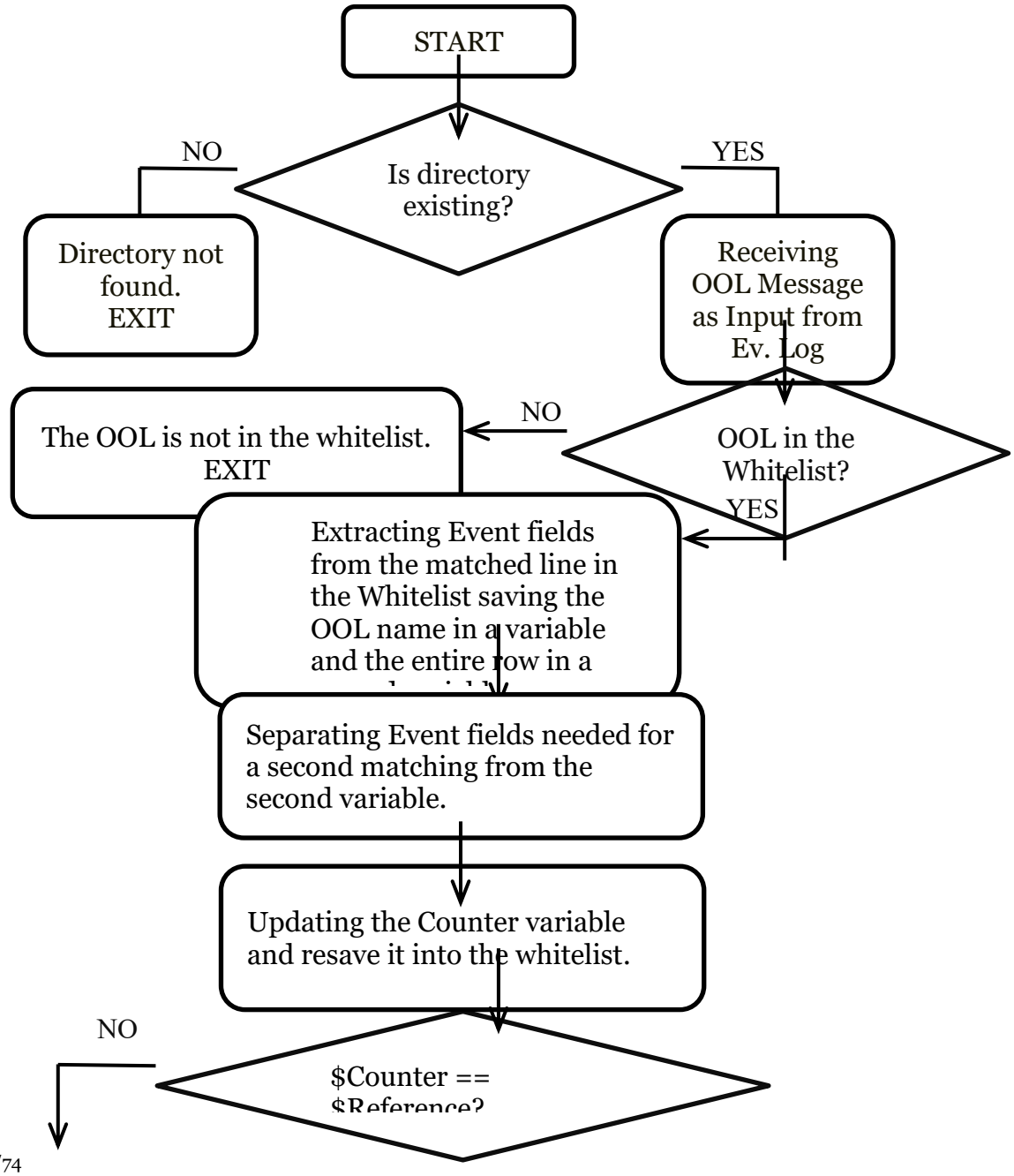
Once the right condition for the creation of the notification has been reached, we want also to check the $requires_call flag to generate the message to be listened in the Phone-Call. If the flags is set equal to “Y” (Upper case Y) than the message (whose content must match the regex in the Testmission file) can be generated and saved, with the other notification, into the generated-event folder.

```
temp_filename="$EVENT_FILE_PATTERN""$domainName""_""$handling""_""$now""_""$RAND
OM"".txt"

final_filename=$(getEventFilenameReady $temp_filename)

cd $REALS_EVENTS

echo "Sentinel requires call" > $temp_filename

chmod 777 $temp_filename

mv $temp_filename $final_filename
```

There is no ELSE condition for this statement.

8 REALS_ACTION_1_OOL.SH

The aim of this script is to generate the alerts only for those OOLs respecting specific constrains, expressed by the FCT, in terms of timing references, amount of maximum notifications and so on.
How does it work?
The script takes in input the description (from the "Message" column) of the OOL raised by SCOS in the Event Log and extracts specific meaningful elements for making a match with the content of a Whitelist in which are specified those OOLs and those constrains to be respected to allow the creation of an alert.
The logical structure
In the following is shown the logical structure that the scripts follows to accomplish its aim.



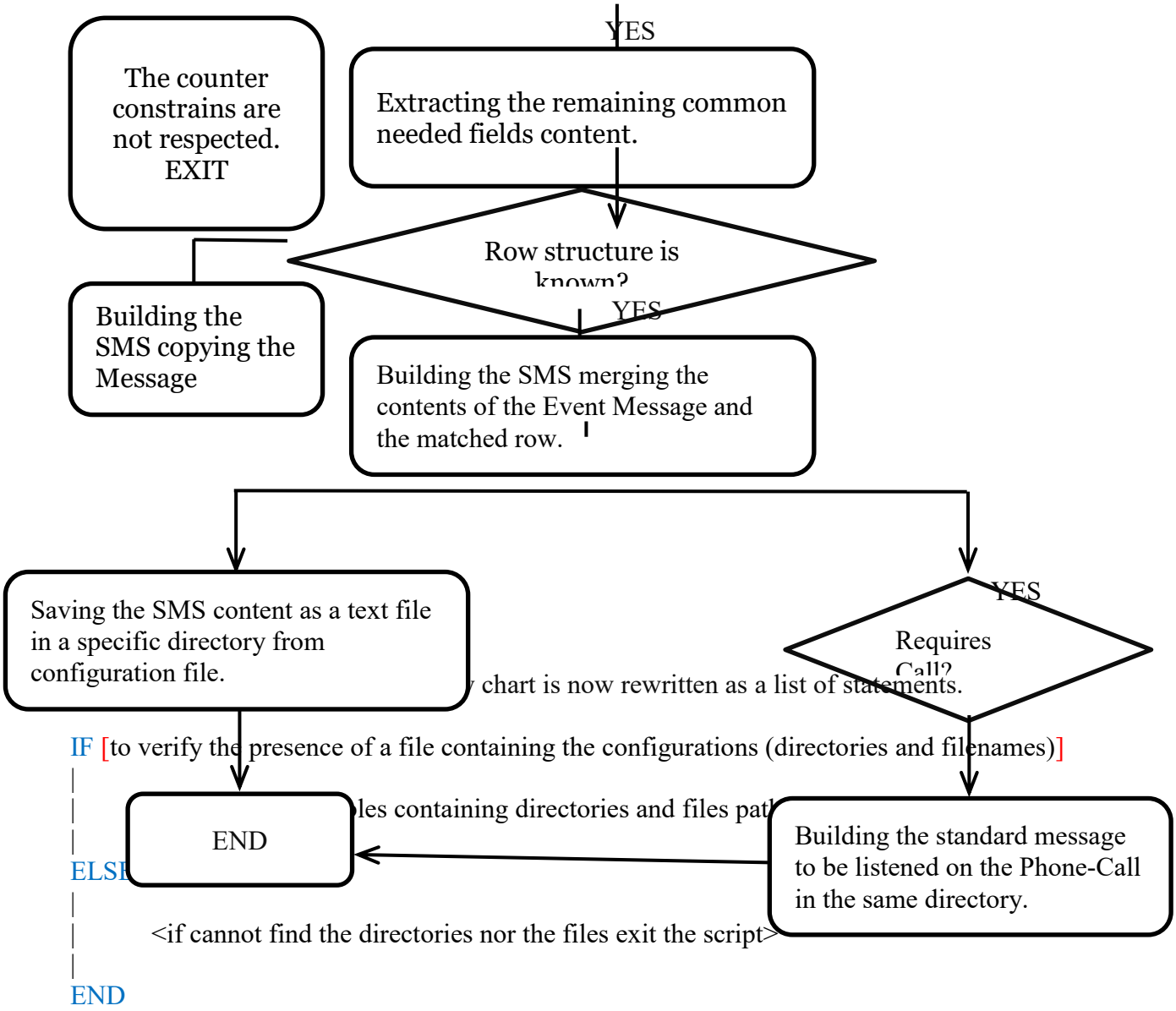


chart is now rewritten as a list of statements.

IF [to verify the presence of a file containing the configurations (directories and filenames)]
|
| les containing directories and files pat
|
ELSE
|
| <if cannot find the directories nor the files exit the script>
|
END

<Receiving the "OOL Message" as input>
.
IF [to verify if the OOL is in the Whitelist]
|
| <Extracting Event fields from the matched row in the Whitelist saving the OOL Description in a variable and the entire row in a second variable>
|
| <Separating Event fields needed for a second matching from the entire line>
|
| <Updating the Counter variable and resave it into the Whitelist>
|.

```
|       IF [to verify constraints on the Counter]
|       |
|       |       <Extracting the needed fields content>
|       |
|       |       IF [to verify if the length of the row fits the known pattern N. 1]
|       |       |.
|       |       |       <Extracting the remaining needed fields content in the specific row pattern>
|       |       |
|       |       |       <Building the SMS merging the contents of the Event Message and the
|       |       |       matched row>
|       |       |
|       |       ELIF [to verify if the length of the row fits the known pattern N. 2]
|       |       |
|       |       |       <Extracting the remaining needed fields content in the specific row pattern>
|       |       |.
|       |       |       <Building the SMS merging the contents of the Event Message and the
|       |       |       matched row>
|       |       |                                       .
|       |       |                                       .
|       |       |                                       .
|       |       |
|       |       ELSE [if the length of the row doesn`t fit any pattern]
|       |       |
|       |       |       <Building the SMS simply copying the Message content>
|       |       |
|       |       END
|       |       |
|       |       <Saving the SMS content as a text file in a specific directory>
|       |       |
|       |       IF [to check if the OOL requires a call]
|       |       |.
|       |       |       <Building the standard message to be listened on the Phone-Call>
|       |       |
|       |       |       <Saving the Phone-Call content as text file into the same specific directory of
|       |       |       the SMSs >
|       |       |
|       |       END
|       |
|       ELSE
|       |
|       |       <The counter constrains are not respected, exit the script>
|       |
|       END
ELSE
|
```

| <The event is not in the Whitelist, exit the script>
|
END

The Whitelist structure for OOLs
The REALS software relies on several Whitelists defined to specify those OOLs and (5, 4) Events that interest the FCT.
The Whitelists have a double scope:

- The Whitelists are meant to give References and Constraints to the reals_action_1_obe.sh and reals_action_1_ool.sh scripts for the filtering of the incoming messages raised by the Event Log in SCOS.

- The Whitelists, for each OOL/Event, give Informations for detailing of the notification message.

The Whitelist for the OOLs (the one we refer to, in this document) is defined for only those OOLs that should require urgent reaction by the FCT.
Both the constraints and the detailing informations can be either numerical values or characters but since UNIX does not distinguish between the two, the difference consists in the way they are handled.
The constraints are all the informations used to verify whether a notification is to be crated or not and they commonly figure in an IF statement or in a *grep* function in the script.
Under the *Constraints* of a specific OOL we find:

- The name of parameter OOL is the very first constraint since is used to check whether the OOL is or not in the Whitelist.

- The *Counter* and the *Reference_counter*, they go together because of their usage as second verification done by the script.
  The *Reference_Counter* is a fixed value and represents the times the event has to be raised to generate an alert.
  The *Counter* is updated each time the event is raised and restored to 1 by a cronjob (5 minutes).
  When its value gets equal to the *Reference_Counter*'s value, the IF statement is verified.
- The length of the row, since the rows in the Whitelist might have different pattern (whether they contain or not the both the limits of the Hard OOL range, the Unit of the parameter, etc.).
  It is used to know how to handle a row with a specific length (containing a known number of fields), if the row length doesn`t fit any known pattern, the message raised by SCOS is not handled and merged with the content of the Whitelist but simply copied into the SMS content.

  These are the 3 main IF conditions as shown in the logical Structure of the script.
  Going through all of them the SMS text is generated and its content is saved in a specific folder whose directory is taken from the configuration file.

- The Requires_Call flag is used to verify a third optional IF statement, nested into the previous ones, i.e. the need of receiving a Phone Call for the raised OOL in addition to the SMS

notification.
If the flag is set to “Y” (Upper case Y), this last IF condition is verified and a second message for the same OOL is created.
This message contains the text to be listened when the Phone Call is received.
This message will be listened when the Phone Call is received message will be handled in a second step by a specific code block into the Testmission file, described into the TN.

Under the *Detailing Informations* of a specific OOL we find:

- OOL description, to clarify the engineer the meaning of the parameter.
- Type of OOL (Hard or Soft), only the Hard ones are allowed by now.
- Range of validity of the Hard constrains.
- Unit, in International System of Units, of the value in the parameter.
- Sub System originating the OOL.

Some of them might be redundant and not all of them will figure in the content of the notification (depending on the engineer needs).
In such a way we could modify the notification content without changing the whitelist.
Here is an example of the content of the Whitelist:

Parameter Name	Parameter Description	Type	Low Limit	High Limit	Unit	SubSys.	Reference	Counter	R_C	End Line
TST00363	GPS-A_Prc_Temp	H	-25	75	degC	MAG	1	1	N	E_L
APD00001	RCS_PT1-PT2_Difference	H	-0,00002	+2		AOCS	1	1	N	E_L
CJT00605	MSI_A_SCV_HLT	H	HEALTHY			MSI	1	1	N	E_L

Since the row can contain an undefined number of words, it implies that its structure in the Whitelist can change depending on the OOL represented.
In specific, we can distinguish in a first instance between the OOLs whose value is numerical, therefore have numerical range of validity, and those OOLs whose value is a state, and the range of validity is represented by a status instead (figuring in the Low Limit column), therefore one of the two fields is missing (High Limit column).
We can also distinguish between those OOLs defined by a Unit (all of those are numerical OOLs) and those for which a Unit is not specified (all the non-numerical OOLs and some of the numerical ones).
This creates some issues in handling the row.
It is necessary to count the number of words (TAB separated) in each row before start manipulating it.
In general, each different OOL (or group of OOL) will have a defined number of words divided by a TAB “\t” character.
First indentation, IF statement:

```
IF [to verify if the OOL is in the Whitelist]
|
|       <Extracting Event fields from the matched row in the Whitelist saving the OOL Description
|       in a variable and the entire row in a second variable>
|
|       <Separating Event fields needed for a second matching from the entire line>
|
```

```
|       <Updating the Counter into the Whitelist>
|                               .
|                               .
|
ELSE
|
|       <The event is not in the Whitelist, exit the script>
|
END
```

The condition enclosed in the red brackets aims to verify IF the content of the message received as the event is raised (content that should identify the OOL) figures in one of the rows of the Whitelist.
The command:

```
MSG_param_name=` echo $@ | xargs | cut -d' ' -f2`
```

saves the name of the parameter (received as input taken from the Event Log) in the $MSG_param_name variable.
The command:

```
go_ahead=` grep -c -i -m 1 "$MSG_param_name"
/home/se2ops/REALS/scripts/WHITELIST_OOL.txt`
```

takes the name of the parameter (as a variable in $MSG_param_name) and gives the number of matched lines as output in the go_ahead variable, using for the comparison the content of the text file whitelist_1_ool.txt.
The flag "-m 1" stops the matching process at the first matched result.
Since in the whitelist there might be repetitions, in such a way the result in the $go_ahead variable is maximum equal to one "1".
In this way the IF condition will only verify the $go_ahead variable content being equal to 1.
If the condition is verified, the OOL figures in the Whitelist, this means that the FCT is interested on it.
The row matching the content of $MSG_param_name is copied in the variable $WL_entire_line.

```
WL_entire_line=` grep -A 0 -m 1 "$MSG_param_name"
/home/se2ops/REALS/scripts/WHITELIST_OOL.txt`
```

The content of $WL_entire_line has now the structure of one of the rows in the Whitelist:

Parameter Name	Parameter Description	Type	Low Limit	High Limit	Unit	SubSys.	Reference	Counter	R_C	End Line
TST00363	GPS-A_Prc_Temp	H	-25	75	degC	MAG	1	1	N	E_L
APD00001	RCS_PT1-PT2_Difference	H	-0,00002	+2		AOCS	1	1	N	E_L
CJT00605	MSI_A_SCV_HLT	H	HEALTHY			MSI	1	1	N	E_L

The following commands take the respective fields occupying position 3, 4 and from 5 on, in the variable $WL_entire_line, counting for them from the bottom of the row (reversing the row) and

save them (reversing again) into $WL_counter, $WL_reference and $WL_row_section:

```
WL_counter=
` echo $WL_entire_line | rev | cut -d ' ' -f3`

WL_reference=
` echo $WL_entire_line | rev | cut -d ' ' -f4`

WL_row_section=`echo $WL_entire_line | rev | cut -d ' ' -f5- | rev | sed "s/ /
        /g"`
```

Be aware that, reading the Whitelist (text file) UNIX interprets TAB characters as if they were SPACE characters, so the separator character into the *cut* function is a SPACE.
The $counter and the $reference are also unpacked from the header and the trailer characters to be handled.
This command is to update the value of the counter:

```
counter_update=$[1+$counter]
```

The $WL_row_section contains the content of the row preceding the counter and the reference counter, in order to update the matching row in the Whitelist substituting the variable $counter with the variable $counter_update and saving the modifications:

```
sed -i "
s/$WL_row_section\t$WL_reference\t$WL_counter/$WL_row_section\t$WL_reference\t$W
L_counter_update/g" /home/se2ops/REALS/scripts/whitelist_1_ool.txt
```

The characters used for the separation of the different fields into the Whitelist are TAB characters “\t”.
In the case the IF condition is not respected, the script give back the message ‘Miss Match’ and terminates the script execution through exit 1.

Second indentation, IF statement:

```
IF [to verify constrains on the Counter]
|
|          <Extracting needed fields content>
|
|                              .
|                              .
|                              .
ELSE
|
|          <The counter constrains are not respected, exit the script>
|
END
```

The condition enclosed in the red brackets takes as variables the $counter and the $reference to check their values to be equal.
3 different cases can happen:

- $counter < $reference
  In this case the IF condition is not respected and the SMS content is not generated, the counter has been already updated to $counter_update = $[1 + $counter].
  It will be needed the OOL to be raised again by SCOS until the value of the counter reaches the reference one, unless a previous occurrence of a cronjob restores the counter value to one.

- $counter > $reference
  In this case the IF condition is not respected and the SMS content is not generated, the counter has been already updated to $counter_update = $[1 + $counter].
  Since this point forward, the counter and the reference will never be equals, only the occurrence of a cronjob, restoring the value of the counter to 1, will allow the cycle to restart.

- $counter = $reference
  In this case the IF statement is respected, the SMS content is generated while the counter has been already updated to $counter_update = $[1 + $counter].
  In this case the needed fields, occupying the position 1 and 2 from the head of the row and the ones occupying the positions 1 and 5 from the bottom of the row in the variable $entire_line are extracted:

  ```
  MSG_time_stamp=` echo $@ | cut -d ' ' -f1`
  MSG_current_value=` echo echo $@ | rev | cut -d ' ' -f1 | rev`
  WL_SubSystem=` echo $WL_entire_line | rev | cut -d ' ' -f5 | rev`
  WL_param_desc=` echo $WL_entire_line | cut -d ' ' -f2`
  WL_num_words=` $WL_entire_line" | wc -w`
  ```

  The fields that are not extracted figure only in the $entire_line variable and are not needed to be handled.
  The third IF statement is now to be check.

Third indentation, IF statement:

```
IF [to verify if the length of the row fits the known pattern N. 1]
|
|        <Extracting the remaining needed fields content in the specific row pattern>
|
|        <Building the SMS merging the contents of the Event Message and the
|        matched row>
|
ELIF [to verify if the length of the row fits the known pattern N. 2]
|
|        <Extracting the remaining needed fields content in the specific row pattern>
```

```
|
|       <Building the SMS merging the contents of the Event Message and the
|       matched row>
|                                   .
|                                   .
ELSE [if the length of the row doesn`t fit any pattern]
|
|       <Building the SMS simply copying the Message content>
|
END
```

Depending on the length of the row, that can vary between 8 and 10 words, we are allowed to know whether the “High Limit” and the “Unit” fields are filled.
In case the row has length equal to 8, the value of the OOL is not numerical, the command:

```
WL_exp_state=` echo $WL_entire_line | cut -d ' ' -f4`
```

saves in the variable $WL_exp_state the field in the 4th position from the head of the row, that contains the expected state of the parameter.
The command:

```
full_event="$MSG_time_stamp| $MSG_param_name| $WL_param_descr| Expected State:
$WL_exp_state| Current State: $MSG_current_value| SubSystem: $WL_SubSystem"
```

builds the SMS content merging the informations from both the Message Content and the specifically structured row in the Whitelist and saves it into the $full_event variable.

In case the row has length equal to 9, the value of the OOL is numerical but the Unit is not specified, the command:

```
WL_exp_state=` echo $WL_entire_line | cut -d ' ' -f4,5`
```

saves in the variable $WL_exp_state the field in the 4th and 5th positions from the head of the row, that contain the expected boundaries of the hard range of validity of the parameter.
The command:

```
full_event=" $MSG_time_stamp| $MSG_param_name| $WL_param_descr| Hard Limits:
<$WL_exp_state> | Current Value: $MSG_current_value| SubSystem: $WL_SubSystem"
```

builds the SMS content merging the informations from both the Message Content and the specifically structured row in the Whitelist and saves it into the $full_event variable.

In case the row has length equal to 10, the value of the OOL is numerical and all the fields are filled, the command:

```
WL_unit=`echo $WL_entire_line | cut -d ' ' -f6`
```

Saves into $WL_unit the content of the field in the 6th position from the head of the row, which contains the Unit, while:

```
WL_exp_state=` echo $WL_entire_line | cut -d ' ' -f4,5`
```

saves in the variable $WL_exp_state the field in the 4th and 5th positions from the head of the row, that contain the expected boundaries of the hard range of validity of the parameter.
The command:

```
full_event=" $MSG_time_stamp| $MSG_param_name| $WL_param_descr| Hard Limits:
<$WL_exp_state> [$WL_unit]| Current Value: $MSG_current_value| SubSystem:
$WL_SubSystem"
```

builds the SMS content merging the informations from both the Message Content and the specifically structured row in the Whitelist and saves it into the $full_event variable.

In case the length of the row is equal to none of these values, the Message Content raised from SCOS, received as input, is simply copied into $full_event

```
full_event=`echo $*`
```

The SMS content is then saved as a text file following the directory specified in the very first if statement (as shown in the flow chart).
The commands follow:

```
cd $REALS_EVENTS
```

Where the variable $REALS_EVENTS contains the directory

```
echo $full_event > $temp_filename
```

and $temp_filename the name of the text file to be saved.
In the cases the IF condition is not respected the script gives back the message ‘Reference != Counter’ and terminates the script execution through exit 1.
Fourth indentation, IF statement:

```
IF [to check if the OOL requires a call]
|
|          <Building the standard message to be listened on the Phone-Call>
|
|          <Saving the Phone-Call content as text file into the same specific directory of the SMSs >
|
END
```

Once the right condition for the creation of the notification has been reached, we want also to check the $requires_call flag to generate the message to be listened in the Phone-Call.

If the flags is set equal to “Y” (Upper case Y) than the message (whose content must match the regex in the Testmission file) can be generated and saved, with the other notification, into the generated-event folder.
There is no ELSE condition for this statement.

```
temp_filename="$EVENT_FILE_PATTERN""$domainName""_""$handling""_""$now""_""$RAND
OM"".txt"

final_filename=$(getEventFilenameReady $temp_filename)

cd $REALS_EVENTS

echo "Sentinel requires call" > $temp_filename

chmod 777 $temp_filename

mv $temp_filename $final_filename
```

The reals_action_1_ool.sh is commented.

9 REALS_FCT_OBE_OOL_LIST

Be aware that the Excel interface, created for an easier use of the REALS software, is not a part of it.
The aim of making an interface and writing a macro for the generation of the Whitelists is double:

- Let the FCT a faster operation, since the Excel interface is built for taking as input a Copy&Paste of the Events and OOL parameters directly from the WODB.
- The macro takes the standard input (as from the WODB) and generates a standardized output (the whitelists) as text file in a format on which the bash scripts can operate.

9.1 Excel General Tasks

The Excel files contains several WorkSheets:

- README, containing general instructions for the use of the interface.
- 1_obe, must contain the lists of on board events, detailed for S2A (domain 1).
- 1_ool, must contain the lists of on Out Of Limits, detailed for S2A (domain 1).
- …etc for each domain.
- Macro, containing the buttons to call the macro and create the whitelist for each different worksheet.
- Subsystems and Originator, containing detailing information.

9.2 Worksheets Containing Lists

The names of the worksheets that will contain the lists of parameter or the on-board Events, defined for each mission, must respect the pattern:

#MissionDomain_#handling

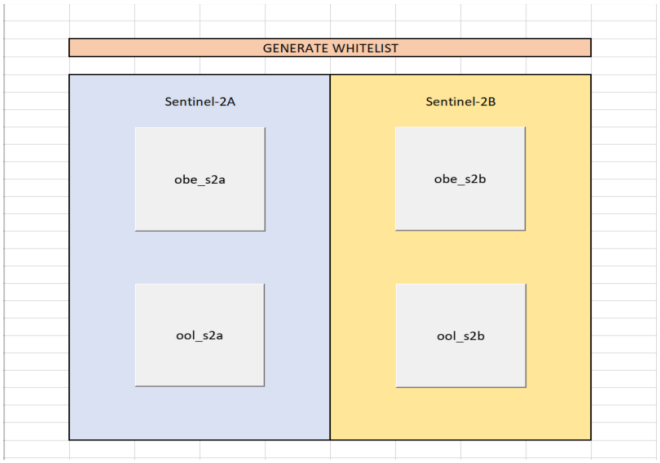
The *mission domain* represents the domain (integer number) associated to the specific mission within the mission's server.
The *handling* represents the acronym to identify whether the list contains ool rather than to obe (lower case ool/obe).
The two fields within the worksheet name must be *Underscore* "_" separated.
For each domain in the server (in our case, by now, two domains), will figure a worksheet for the ool and a worksheet for the obe.

In this case, *1_obe* represents the list of on board events referred to S2A mission (domain 1 into the se2amca/se2amce server).

In case a new whitelist is needed for a new domain (mission), it will be sufficient to follow this name standardization for the new worksheets.

9.3 Macro

The macro worksheet contains the buttons associated to the VBA functions that call the macro itself.
The functions give as input the worksheet name to which refer for the creation of the whitelist.



To the button obe_s2a will be associated the input (worksheet name) 1_obe, to ool_s2a will be associated the input 1_ool and so on.
If a new worksheet is created, also a new function must be defined to call the macro using the proper input.
Those functions are defined into the *Sheet5 (Macro) under the Developer window extension*.

9.4 Subsystem & Originator

The *Subsystem & Originator* worksheets contains two different lists (list of S/C subsystems and engineers names), to be used as source for the scroll-down menu to detail the events/ools into the previous worksheets described in the paragraph above.

9.5 Macro Output

As one of the button is clicked, the new relative whitelist is generated converting the content of the relative worksheet into a structured text file, readable by the bash scripts.
The output is double:

- A first copy of the whitelist with the formatted name proper for its handling (whitelist_#domain_#handling.txt) is saved into the folder ***Latest_Whitelists*** created in the same Excel directory.
  This folder must be copied into the mission's server under the path:

  /home/se2ops/REALS/scripts/***Latest_Whitelists***/

  This folder will be the BackUp used by the ***reals_gronjob.sh*** for restoring the whitelist, each time it is called.

- A second folder is created (***Whitelist_Archive_1_ool/*** and ***Whitelist_Archive_1_obe/***) as archive to keep a copy of all the generated lists. The copies within this archive will differ by the name, which contains a time stamp to identify the creation date.
  There is no need to copy this folder into the mission's server.

9.6 VBA Macro

The aim of the macro is to take the structured list into either the ***obe*** or the ***ool*** Sheet to create and format a whitelist in a specific, different and more complex structure readable by the reals_action_1_obe.sh and reals_action_1_ool.sh scripts being part of the REALS software.
The macro is saved into the *Sheet5 (Macro) under the Developer window extension*.
The logical structure of the macro:

1. The first step for the macro is to build the path and the name of the whitelist text file, specific for its content (depending whether it contains the OOLs or the Events).
   This step is performed taking as reference the WorkSheet Name itself (passed by the functions described above), that will follow the Text standardization as described in ***Paragraph 2.1***.
2. The second step is to check for the existence of the file paths and, if none already exists, they are created.
   To be handled by the bash script, the Whitelist need to be named as:
   a. whitelist_1_obe.txt
   b. whitelist_1_ool.txt

   As mentioned, they figure under the path:
   \\escwindfs01\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\Latest_Whitelists\whitelist_1_obe.txt
   And
   \\escwindfs01\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\Latest_Whitelists\whitelist_1_ool.txt
   Moreover, a backup of all the generated Whitelists is saved under the sub-folders:
   \\escwindfs01\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\Whitelist_Archive_1_obe\
   And
   \\escwindfs01\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\Whitelist_Archive_1_ool\
   Adding to each new text-file name the time stamp of its generation.
3. The third step is to open the text file and write into the files following a specific pattern.
   For the OBE:

5	4	199	54690	23824	E_EID_PLCT_FMON_3_OCP_COM	MAG	1	1	Y	E_L

   Since the "Event Description" field might contain an undefined number of words, its content is taken to substitute each SPACE character " " with an underscore

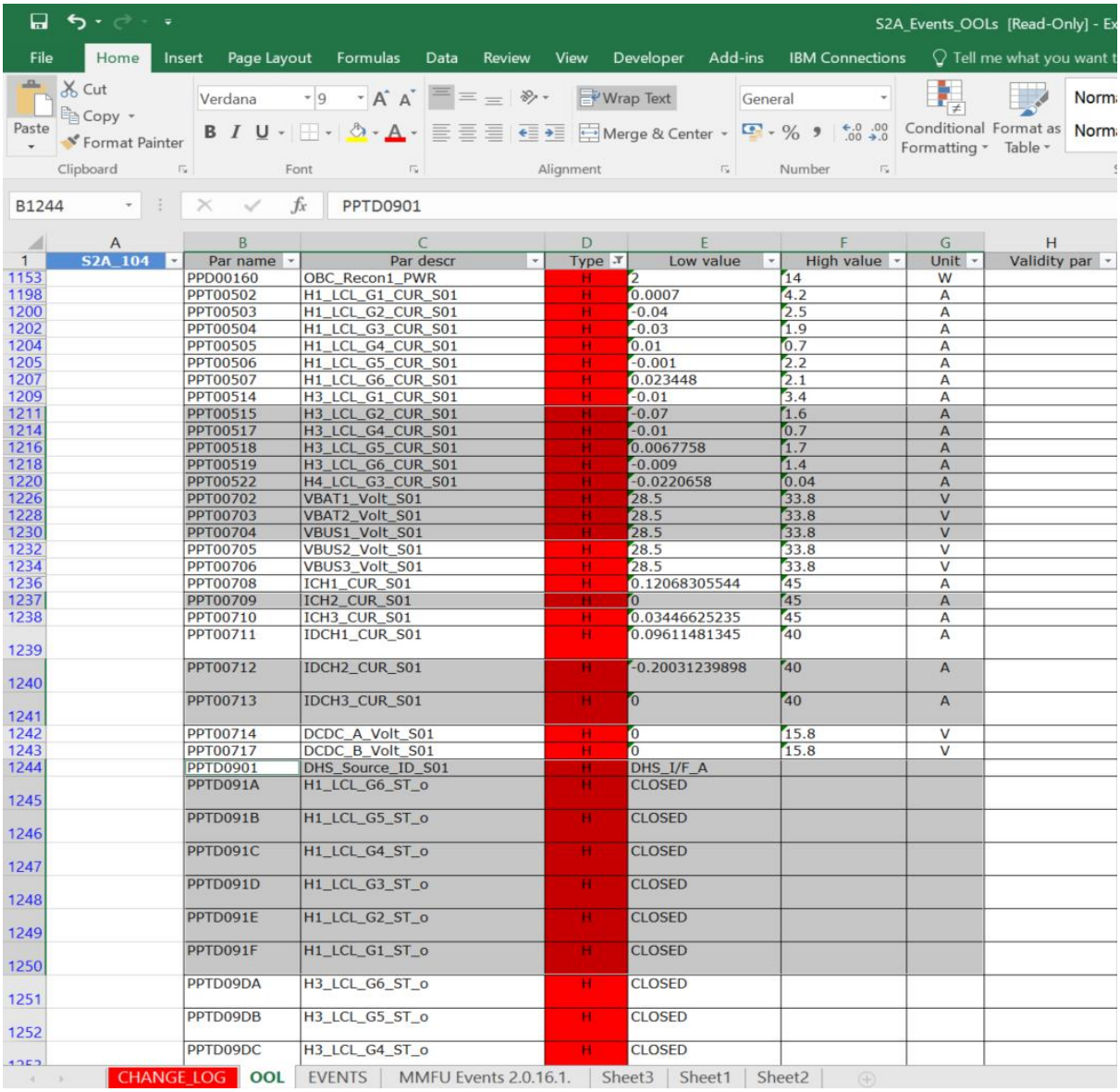
character "_".
The same operation is done by the bash script receiving this same content directly from the Event Log.
In such a way the match will be performed between the two modified strings
For the OOLs

TST00382	MSI_VFPA_TH_SURV_A	H	10	30	degC	MSI	1	1	N	E_L
CLT12166	XTRP2A_EHS	H	1	1		XBS	1	1	N	E_L
CJT60072	MSI_REFUSE_MODE	H	False			MSI	1	1	N	E_L

All of the fields are then separated by a space Tab character "\t" when translated in text file in order to be handled by the scripts.
A detailed description of both the Whitelists is given into the description of the reals_action_1_obe.sh and reals_action__1_ool.sh word files.
The macro VBA script is commented and understandable.

9.7 ool interface

The very first step for the engineer is to open the WODB (at the following link) and choose the parameters for which he wants to be notified, in case they go Out Of Limits.
\\escwindfs01\GMES Sentinels FOS$\Sentinel-2\WODB\S2A\S2A_Events_OOLs.xlsx
Into the OOL Sheet, the engineer has to copy, for each selected row, the columns from "B" to "G" (going from the *Parameter Name* to the *Unit*), as shown in the following example image.



S2A_Events_OOLs [Read-Only] - Ex

File | Home | Insert | Page Layout | Formulas | Data | Review | View | Developer | Add-ins | IBM Connections | Tell me what you want t

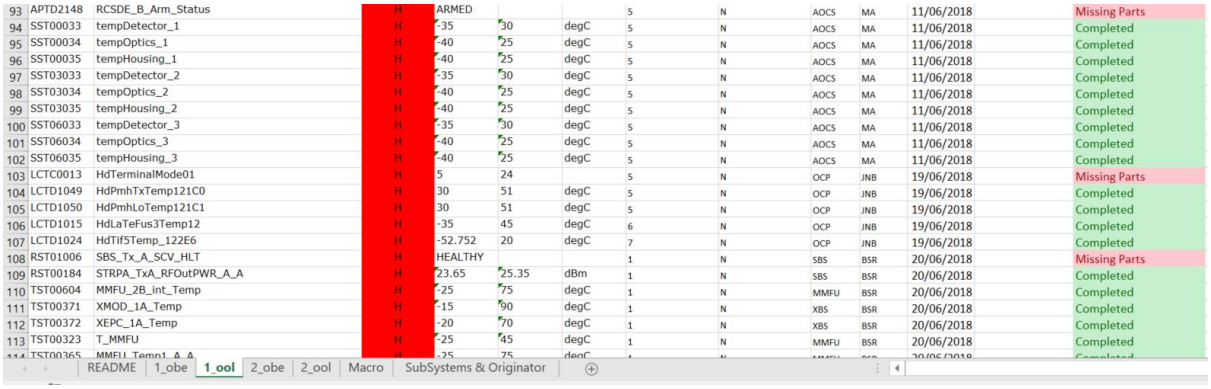
Cut | Copy | Format Painter | Paste | Clipboard | Verdana | 9 | Font | Wrap Text | Merge & Center | Alignment | General | Number | Conditional Formatting | Format as Table | Norm | Norm

B1244 | PPTD0901

	A	B	C	D	E	F	G	H
1	S2A_104	Par name	Par descr	Type	Low value	High value	Unit	Validity par
1153		PPD00160	OBC_Recon1_PWR	H	2	14	W	
1198		PPT00502	H1_LCL_G1_CUR_S01	H	0.0007	4.2	A	
1200		PPT00503	H1_LCL_G2_CUR_S01	H	-0.04	2.5	A	
1202		PPT00504	H1_LCL_G3_CUR_S01	H	-0.03	1.9	A	
1204		PPT00505	H1_LCL_G4_CUR_S01	H	0.01	0.7	A	
1205		PPT00506	H1_LCL_G5_CUR_S01	H	-0.001	2.2	A	
1207		PPT00507	H1_LCL_G6_CUR_S01	H	0.023448	2.1	A	
1209		PPT00514	H3_LCL_G1_CUR_S01	H	-0.01	3.4	A	
1211		PPT00515	H3_LCL_G2_CUR_S01	H	-0.07	1.6	A	
1214		PPT00517	H3_LCL_G4_CUR_S01	H	-0.01	0.7	A	
1216		PPT00518	H3_LCL_G5_CUR_S01	H	0.0067758	1.7	A	
1218		PPT00519	H3_LCL_G6_CUR_S01	H	-0.009	1.4	A	
1220		PPT00522	H4_LCL_G3_CUR_S01	H	-0.0220658	0.04	A	
1226		PPT00702	VBAT1_Volt_S01	H	28.5	33.8	V	
1228		PPT00703	VBAT2_Volt_S01	H	28.5	33.8	V	
1230		PPT00704	VBUS1_Volt_S01	H	28.5	33.8	V	
1232		PPT00705	VBUS2_Volt_S01	H	28.5	33.8	V	
1234		PPT00706	VBUS3_Volt_S01	H	28.5	33.8	V	
1236		PPT00708	ICH1_CUR_S01	H	0.12068305544	45	A	
1237		PPT00709	ICH2_CUR_S01	H	0	45	A	
1238		PPT00710	ICH3_CUR_S01	H	0.03446625235	45	A	
1239		PPT00711	IDCH1_CUR_S01	H	0.09611481345	40	A	
1240		PPT00712	IDCH2_CUR_S01	H	-0.20031239898	40	A	
1241		PPT00713	IDCH3_CUR_S01	H	0	40	A	
1242		PPT00714	DCDC_A_Volt_S01	H	0	15.8	V	
1243		PPT00717	DCDC_B_Volt_S01	H	0	15.8	V	
1244		PPTD0901	DHS_Source_ID_S01	H	DHS_I/F_A			
1245		PPTD091A	H1_LCL_G6_ST_o	H	CLOSED			
1246		PPTD091B	H1_LCL_G5_ST_o	H	CLOSED			
1247		PPTD091C	H1_LCL_G4_ST_o	H	CLOSED			
1248		PPTD091D	H1_LCL_G3_ST_o	H	CLOSED			
1249		PPTD091E	H1_LCL_G2_ST_o	H	CLOSED			
1250		PPTD091F	H1_LCL_G1_ST_o	H	CLOSED			
1251		PPTD09DA	H3_LCL_G6_ST_o	H	CLOSED			
1252		PPTD09DB	H3_LCL_G5_ST_o	H	CLOSED			
		PPTD09DC	H3_LCL_G4_ST_o	H	CLOSED			

CHANGE_LOG | OOL | EVENTS | MMFU Events 2.0.16.1. | Sheet3 | Sheet1 | Sheet2

After having copied the selected (greyed) rows, the engineer has to paste them into the respective OOL Scheet into REALS_FCT_Event_OOL_list.xlsm file, being the Excel interface for the FCT.
\\escwindfs01\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\REALS_FCT_Event_OOL_list.xlsm
As shown in the following example image.



93	APTD2148	RCSDE_B_Arm_Status	H	ARMED			5	N	AOCS	MA	11/06/2018		Missing Parts
94	SST00033	tempDetector_1	H	-35	30	degC	5	N	AOCS	MA	11/06/2018		Completed
95	SST00034	tempOptics_1	H	-40	25	degC	5	N	AOCS	MA	11/06/2018		Completed
96	SST00035	tempHousing_1	H	-40	25	degC	5	N	AOCS	MA	11/06/2018		Completed
97	SST03033	tempDetector_2	H	-35	30	degC	5	N	AOCS	MA	11/06/2018		Completed
98	SST03034	tempOptics_2	H	-40	25	degC	5	N	AOCS	MA	11/06/2018		Completed
99	SST03035	tempHousing_2	H	-40	25	degC	5	N	AOCS	MA	11/06/2018		Completed
100	SST06033	tempDetector_3	H	-35	30	degC	5	N	AOCS	MA	11/06/2018		Completed
101	SST06034	tempOptics_3	H	-40	25	degC	5	N	AOCS	MA	11/06/2018		Completed
102	SST06035	tempHousing_3	H	-40	25	degC	5	N	AOCS	MA	11/06/2018		Completed
103	LCTC0013	HdTerminalMode01	H	5	24		5	N	OCP	JNB	19/06/2018		Missing Parts
104	LCTD1049	HdPmhTxTemp121C0	H	30	51	degC	5	N	OCP	JNB	19/06/2018		Completed
105	LCTD1050	HdPmhLoTemp121C1	H	30	51	degC	5	N	OCP	JNB	19/06/2018		Completed
106	LCTD1015	HdLaTeFus3Temp12	H	-35	45	degC	6	N	OCP	JNB	19/06/2018		Completed
107	LCTD1024	HdTif5Temp_122E6	H	-52.752	20	degC	7	N	OCP	JNB	19/06/2018		Completed
108	RST01006	SBS_Tx_A_SCV_HLT	H	HEALTHY			1	N	SBS	BSR	20/06/2018		Missing Parts
109	RST00184	STRPA_TxA_RFOutPWR_A_A	H	23.65	25.35	dBm	1	N	SBS	BSR	20/06/2018		Completed
110	TST00604	MMFU_2B_int_Temp	H	-25	75	degC	1	N	MMFU	BSR	20/06/2018		Completed
111	TST00371	XMOD_1A_Temp	H	-15	90	degC	1	N	XBS	BSR	20/06/2018		Completed
112	TST00372	XEPC_1A_Temp	H	-20	70	degC	1	N	XBS	BSR	20/06/2018		Completed
113	TST00323	T_MMFU	H	-25	45	degC	1	N	MMFU	BSR	20/06/2018		Completed
114	TST00365	MMFU_Temp1_A_A	H	-25	75	degC	1	N	MMFU	BSR	20/06/2018		Completed

README | 1_obe | 1_ool | 2_obe | 2_ool | Macro | SubSystems & Originator

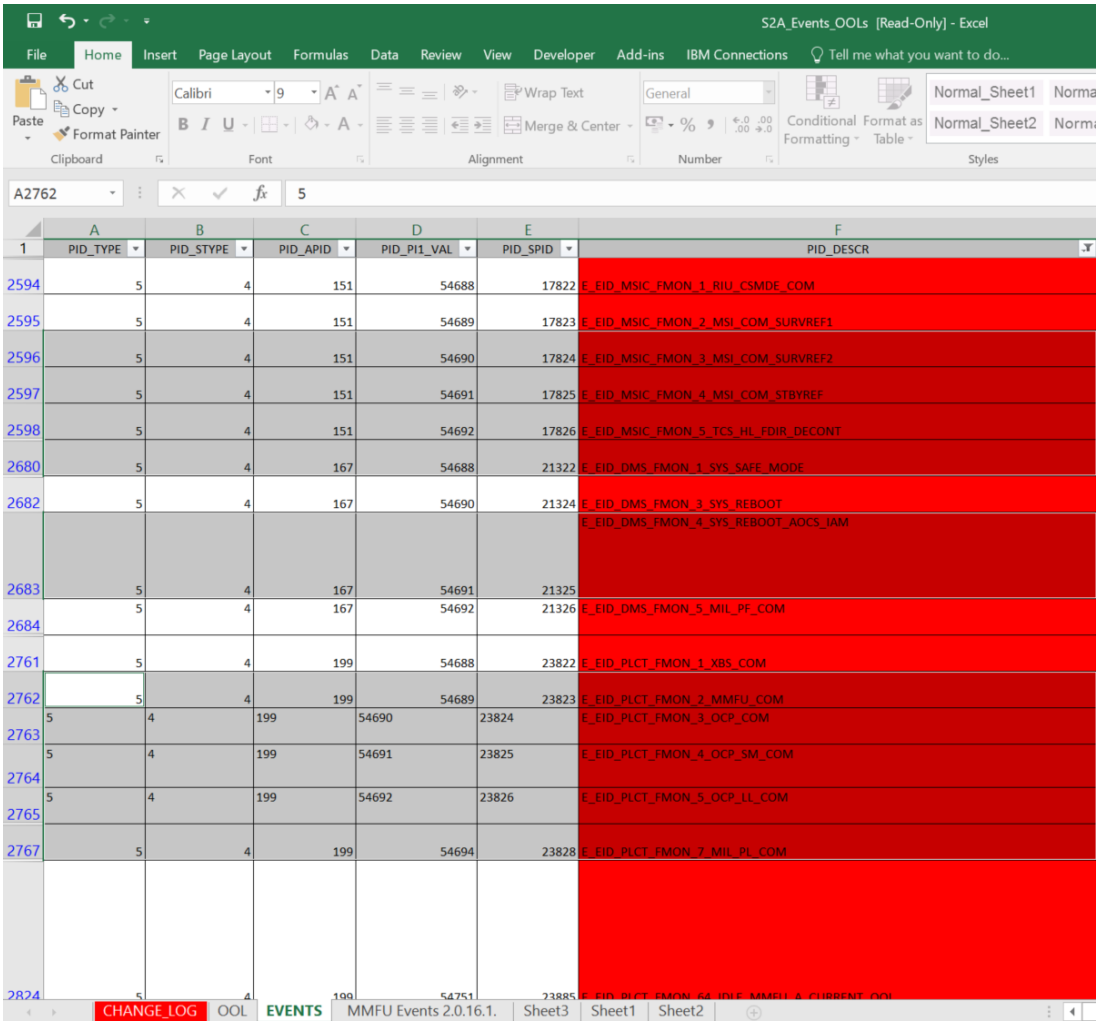
The rows copied from the WODB have to figure from column “A” to column “F”.
The remaining ones, columns from “G” to “K”, have to be filled manually by the engineer and represent:

g. Reference Counter: being the OOL raising repetition at which the notification is sent.
h. Requires Call flag: Must be set equal to “Y” (Upper case Y) if the raising of that specific OOL requires a Phone-Call notification (Upper case N otherwise).
i. SubSystem to which the parameter is referred (to be selected from a scroll-down list).
j. Originator: the initials of the engineer requesting the notification (to be selected from a scroll-down list).
k. Date in the format DD/MM/YYY when the list is updated with the selected rows.

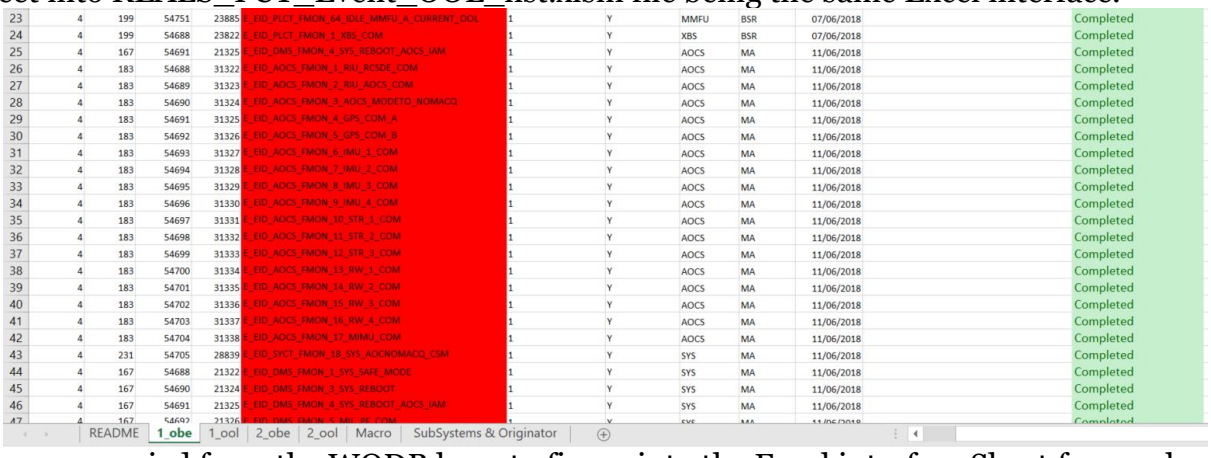
The “L” column is left for eventual comments by the SOE and a check for missing informations.
Be aware that not all the fields under the “E” and “F” columns necessarily need to be filled. Some parameters might not represent a physical measure (missing the Unit field under the “F” column) or must respect a status instead of a numerical range (missing the HIGH Value under the “E” column).
The “M” column is RED highlighted in case some fields are missing but, in this case, since there might be no need to fill all of them, the highlighted cell will keep being red without representing any error in the format of the Whitelist..
A visual check is anyway recommended for all the new red alerted rows.

9.8 obe interface

For the On Board Events the procedure to follow is the same.
Under the OBE Sheet of the WODB, the engineer has to copy, for each selected row, the columns from “A” to “F” (going from the Event Type to the Event Description), as shown in the following example image.



After copying the selected rows, the engineer has to paste them into the respective OBE Sheet into REALS_FCT_Event_OOL_list.xlsm file being the same Excel interface.



The rows copied from the WODB have to figure into the Excel interface Sheet from column “A” to column “F”.
The remaining ones, columns from “G” to “K”, have to be filled manually by the engineer and, as for the OOLs, represent:

g. Reference Counter: being the OOL raising repetition at which the notification is sent.
h. Requires Call flag: Must be set equal to “Y” (Upper case Y) if the raising of that specific Event requires a Phone-Call notification (Upper case N otherwise).
i. SubSystem to which the parameter is referred (to be selected from a scroll-down list).
j. Originator: the initials of the engineer requesting the notification (to be selected from a scroll-down list).
k. Date in the format DD/MM/YYY when the list is updated with the selected rows.

The “L” column is left for eventual comments by the SOE and a check for missing informations.
In this case, *all the fields must be filled* and the “L” column must be GREEN for all the pasted rows.

10 TESTS PERFORMED

The REALS system, the scripts and the database changes were tested using the simulators on the DEVLAN servers.
End-to-end tests were conducted to prove the capability of the software to manage with all the parameter's OOLs within the lists.
As a first step the parameters have been divided into different lists depending whether their constraints were Boolean or float (i.e. the modification needed was affecting the Raw Value rather than the Engineering Value, from the ScTmPrameterPool figuring in the simulator interface).
Different scripts were created to interface with the simulator and with the expressed functions to handle the parameters.
The path followed by the scripts to cause the OOLs starts from the list of parameters furnished manually for the tests.
The script (Any of them) reads the given list row-by-row to collect each parameter, points to the object defining the read parameter's properties and calls the simulator's function to force the parameter's value (Eng rather than Raw value, defined as properties of the object).
For analysing the list of parameters defined by Boolean value, the script switches the Raw value from 0 to 1 or vice versa depending on its current status.
If the list analysed contains Engineering parameters, the function defined manually within the script takes as input also the limits (defined within the list) and adds a standard quantity (+1) to the higher, relying on the simulator's function to force the Engineering value.
Once this operation has been performed, the script calls a delay (set to 5 seconds) to allow the OOL figuring into the Event Log (on which the REALS software relies).
As the delay has passed, the script calls the simulator's function defined to restore the forced parameter to its nominal value.
This cycle repeats for each parameter in the lists.
It has been noted that there is a substantial difference between the two classes of parameters.
The Engineering values are not generated by the OBSW.
The function used to force their value acts directly on their source, keeping the parameter out of the limits.
For the Raw parameters, their value is generated by the OBSW and cannot be forced using the simulator's function but must be caused by configuring the software itself, therefore such a script cannot be used for their testing.
A list of the results of the tests figures in the Excel interface for the FCT:

\\ESAAD\esoc\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\REALS_FCT_Event_OOL_list.xlsm

Into the "*1_ool*" Sheet, under the "TESTING (O)" column.
The scripts used for the testing and the relative lists figure in the simulator machine *simi022* under:

/home/s2simdev/USERS/FCT/LMGagliardini/

The files are also saved into the shared drive:
\\ESAAD\esoc\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\REALS\REALS_Testing

A bash script takes as input several files containing the specific lists and the different defined functions (mentioned above) to compose the java scripts as a single file.
A first bash scripts, used to create a specified list of parameters, takes source from the whitelist_1_ool.txt (generated by the Excel interface) and selects the rows depending on their length to determine the parameter type, and groups them, i.e. Raw value or Engineering value.

The analysis takes as a reference the same rules used in the reals_action_1_ool.sh for selecting how to handle the events, whose description is linked in the present document as Annex named reals_action_1_ool_whitelist_description
The functions mentioned above are defined manually for the different kind of parameters.
The other input to the bash script are the java functions built specifically to cause the OOLs acting on the Raw value rather than on the Engineering value of the parameter.
The structure of the final java scripts (***javascript_lhu.js***, ***javascript_lh.js***, ***javascript_l.js***) consists of two functions (specific for the length of the rows in the list, saved as ***get_ool_##.txt*** and ***restore_ool_##.txt***) for causing and restoring the OOL and the listed parameters with the respective limits (saved as ***array_##.txt***), formatted as a java script.
All the files can be found, with the results, in the server ***se2amce*** under:
/home/se2ops/REALS/Javascript_testing_S2A/

The files bash scripts are configured to run on the server ***se2amce*** and to give, as output (under the ***java_script_to_simulator*** folder) the java scripts configured to run into the simulator.
Each time script_composer.js in called, a new java script is created with the latest arrays and functions content.

11 STEPS FOR NEW MISSION'S ROLLOUT

Let us describe the steps needed for the deployment, starting from the top entries of the block diagram, shown in ***Paragraph 5.1***, following a logical cascade of configurations.

11.1 REALS Folder Moving

In case the deployment involves a new mission's server, the first step consist on moving the entire ***REALS*** folder and the Action.dat file (that must figure under a different directory) to the new mission's server.
It is important to leave the files within the REALS main folder in their location.
The ***REALS*** folder must be located under a specific users folder (se2ops/se2user/etc..).
Each time the Action.dat file is updated the EVENT and ACTION application of SCOS must be stopped and restarted.
In case the server is common for several missions, we can directly jump to the next step.

11.2 reals_config.cfg

Into the server's folder:

/home/se2ops/REALS/

The ***reals_config.cfg*** was configured with the proper mission's directory and filenames.
In order to do this it was enough to adapt the first two rows of the file:
REALS_USER="se2ops"
REALS_HOME="/home/$REALS_USER/REALS".

11.3 Whitelists Configuration

The second configuration step concerns the whitelists.
In this case, the documentation must be followed.
Refer to the ***macro_whitelist_description.pdf*** linked here as Annex.

11.4 Bash Scripts Configuration

The bash scripts (reals_action_1_ool.sh, reals_action_1_obe.sh, reals_cronjob.sh) must be changed following the documentation linked into this same documents under Annexes.
In any case, be careful to change the proper directories in all the scripts to adapt to the mission's server.

11.5 Action.dat Configuration

The ***Action.dat*** file makes a first general specification on the input coming from SCOS, as described in ***Paragraph 5.2*** in this same document.
In the same document is specified where to copy the duplicate file and start the configuration.
The Action.dat file also specifies the bash scripts (in terms of Directory_Path/fileName.sh) specific for handling the input, therefore be aware of their adaptation to the mission (for details look to the linked Annexes named
reals_action_1_obe_whitelist_description.pdf and

reals_action_1_ool_whitelist_description.pdf.
In our case, the structure of the TM coming from the Event Log for S2A and S2B is the same.
Therefore, it will be only needed to change the file names (using the proper domain specification) since the script itself reads its own standardized name to refer to the proper whitelist and create the proper output following the Text standardization (as specified in ***Paragraph 1.4***).

11.6 Testmission_1_Action.properties Configuration

A copy and paste of the existing version for domain 1 (S2A) can be adapted to a new domain as in described in ***Paragraph 5.6***.

11.7 crontab –e

When accessed the server using the proper account, the command ***crontab –e*** allows to specify the frequency at which a specific script is called.
In our case we want to define the reals_cronjob.sh to be called each 5 minutes.
The line of command for this is as follows:

```
*/5   *   *   *   *   /home/se2ops/REALS/scripts/reals_cronjob.sh
```

11.8 Starting the Software

Once everything is in position, the following steps must be performed:

- The ***generated-event*** folder must be emptied.
- Be aware that the file ***/home/se2ops/RELAS/config/cronjob_status.txt*** figures a "0" (zero, corresponding to *cronjob not running*).
- From command line, let's locate into the ***/home/se2ops/REALS/*** folder to check the status of the software, calling the script there figuring:
  ./status.sh
  The software should be NOT RUNNING.
- The software can be started calling the start script:
  ./start.sh
  If everything is properly configured, the script should start giving back the instances for each domain.
- If we check the status of the software calling again the status.sh script, we should receive back the software RUNNING with the instances of each domain.
- The software can be stopped calling the stop.sh command:
  ./stop.sh

- As started the software, call the ***reals_cronjob.sh*** manually (from command line: ./reals_cronjob.sh).
  The script will copy and paste the whitelists from the ***Latest_Whitelists*** backup folder to the operational folder (/home/se2ops/REALS/scripts/).

12 ISSUES

12.1 Testmission_1_Action.properties file (03/07/2018)

The Testmission_1_Action.properties file on the server se2amce

/home/se2ops/REALS/config/1/Testmission_1_Action.properties

is composed of a series of regular expressions to match the structure of the incoming messages sent by the reals_cronjob.sh file directly from the generated-event folder to the REALS server.
Two different miss-behaviour have been found handling this file:

1. The Testmission_1_Action.properties contains regular expressions also for those messages notifying the status of a parameter back to a nominal value but, since those same OOLs have already been filtered out at a previous level (by the Action.dat file), this part of code was commented out.
   The result was that all the notifications both for the hard OOLs and for the FMONs were missing.

2. It was also found that the structure to be matched by the regular expressions for those messages requiring notifications were not filtered.
   Instead, the Testmission_1_Action.properties allowed all the messages with any structure to be sent to the REALS server.
   It was noticed, in the specific, that the content of the field filled in the Testmission_1_Action.properties was not affecting the parsing of the messages.

It is concluded (up today) that all the fields filled in the Testmission_1_Action.properties are necessary for the proper routing of the notifications but none of them is useful for their parsing.
Since all the needed filtering steps were accomplished by previous operations (in the Action.dat and in the reals_action_1_obe.sh and reals_action_1_ool_sh files) the Testmission_1_Action.properties file is not affecting the proper working of the software if left as figures in this same folder (G:\Sentinel-2\Operations\Automation\REALS\Documents\Described_scripts).

The same file for the Cryosat-2 mission has a customized structure.
Andrea Papa is the Engineer who modified that file for Cryosat (and created the REALS Cryosat documentation).
A further modification could be needed in case of a more complex routing of the notifications and might need his help.

See ***Paragraph 7.5*** to follow on.

13 CHANGES

The REALS client version for Sentinel-2 required a few modification to be used operationally, these include:

13.1 reals_action_1_obe.sh & reals_action_1_ool.sh

The reals_action.sh script was separated into two scripts, reals_action_obe.sh for (5, 4) Events and reals_action_1_ool.sh for OOLs.
Moreover, into those was added code to filter specific events and OOLs.
For description go to ***Paragraph 5.4***.

13.2 Whitelists

Creation of Whitelists – Only for those Events and OOLs requiring the attention of the on-call engineer/SOM.

13.3 reals_cronjob.sh

The reals_cronjob.sh was modified to restore to overwrite the modified Whitelists at each cronjob occurrence.

13.4 Pid table

As mentioned in the paragraph 2.1, the Event ID field in the Event Log is a field used by only a few applications (e.g. the ACTION application), therefore by default this field is not populated.
Any EV.ID figuring into the PID table has been expressly modified and so was in our case. Since the ACTION application cannot handle an input without the specification of both the Application and the EV.ID, a modification was needed.
The PID table can be found in the latest WODB Access file, and the relative Event Identifier under the PID_EVID column.

\\ESAAD\esoc\GMES Sentinels FOS$\Sentinel-2\WODB\S2A\WODB_Archive\MERGED

In this table, the fields on which the EV.ID for SCOS can be defined figures under the 16th column, whose name is “EV_ID”, as intuitive.
From this table have been selected ***all and only the (5, 4) events*** and to them was assigned the same Event ID (1104).
It is not needed to create a different Event Id for each of the (5, 4) events since they would be handled in the same way by the Action.dat.
In such a way all the (5, 4) events are allowed to the next filter step.

13.5 Testmission_1_action.properties

Referring to the ***Paragraph 12.1***, it has been understood that all the field from all of the three blocks of code (parsing the HARD OOLs, the SOFT OOLs and the OBE) are necessary

for the handling of the notification by the REALS server.
The very first block contained the regular expression that follows:

(\\d{4}\\.\\d{3}\\.\\d{2}\\.\\d{2}\\.\\d{2}\\.\\d{3}) (\\w+) Current state:((HIGH HIGH HIGH)|(LOW LOW LOW)) value: ([-+]?\\d+([.]{1}\\d+)?)

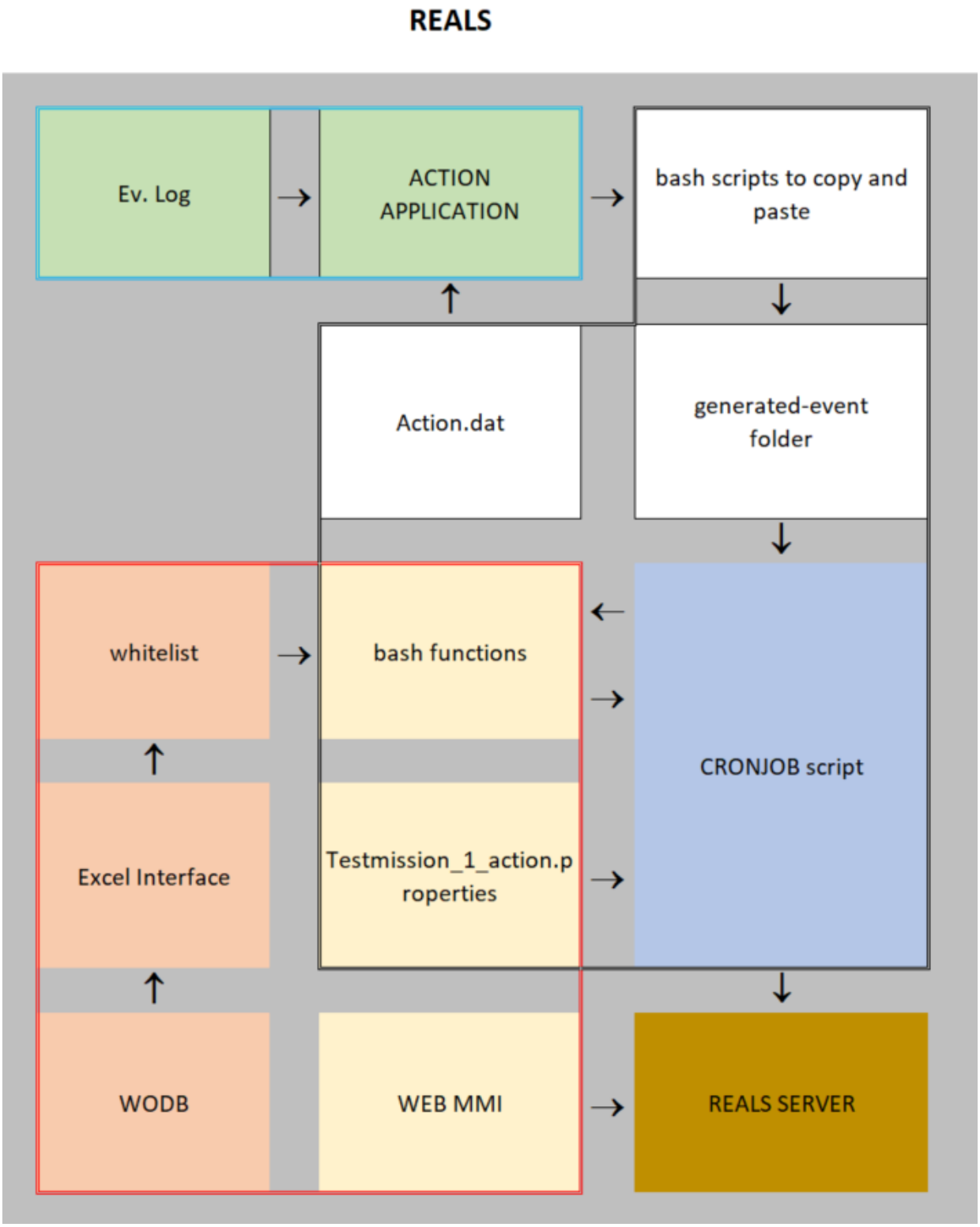
in order to match the incoming messages respecting the following structure:
2018.173.09.13.17.866 APT00265 Current state:HIGH HIGH value: 99.9975
The second block of code used to contain a regular expression for the matching of the messages notifying the parameters back to their nominal value.
As explained in paragraph 6.1 (1.) those kind of messages were already filtered out by the Action.dat file.
This second block was then modified in order to match a message generated by the scripts as a second output (the first is properly what will be the content of the SMS with the Event description) to be used as readable text for the Phone Call notification.
Since there is not the Possibility of notifying the alerts through a Mobile Phone Call, it has been decided to use the local ESOC LAN to receive these SMSs.
The local line will handle the SMSs as Phone Calls read by synthetic vocalizer.
This Phone Call is then forwarded to a Mobile Phone.
The content of this message is not needed to contain the Event information since works only as an alarm to the On-Call SOE, the Event description will be received to the same number as an SMS.
The third block is made to parse all the incoming messages except for those containing “SCC FAIL”.
Given those assumption, in the case the regular expression of the first block is not matching the incoming message this is sent to be parsed by the second block.
If neither this second block is matching the message it is finally sent to the third block accepting nearly everything.
Here it is easy to understand that the Testmission_1_action.properties is not a proper filter but is used to make a last composition of the notification depending on the structure of the incoming message.
In case of a further customization of the file, a deeper investigation is needed.

13.6 On-Call Flag

Since it has been raised the need of receiving a Phone-Call for specific (5, 4) Events and specific OOLs, it was decided to add a flag to each line of the Whitelists (for each Event/OOL), specifying whether or not the Phone-Call is required.
This flag specifies the yes-condition “Y” (Upper case Y) for the requiring of a Phone-Call rather a no-condition “N”.
It is been located under the second to last column of each row into the Whitelist, before the End_Line “E_L” character.
The files describing the Whitelists and the bash scripts handling them have been modified and are now updated.

13.7 Change of bash scripts architecture

The too frequent triggering of the Events/OOLs causing the re-calling of the reals_action scripts while they are still running raised the need to revise the architecture of the bash scripts.
The new architecture replaces the first demanding scripts activities with more basic functions, faster completed and thus avoiding a re-call while still running.
The filtering operations (i.e. the longest) will take place recursively at the occurrence of a cronjob, having the time to handle, one by one, all the inputs.
Such an architecture allows all the Events/OOLs respecting the conditions specified in the Action.dat file to be saved into the generated-event folder (e.g., up to 55 entries during testes were passed to be saved into the generated-event folder).
This architecture uses short scripts (few lines, whose due time takes a few milliseconds) to quickly save the incoming message directly into a text file into the generated-event folder.
Depending on the alert raised from SCOS, a different script is called respecting the text standardization.
This means that in case of the raising of an OOL from S2A, the script figuring into the Action.dat file to save the incoming message is: ***reals_saving_1_ool.sh***.
In case of an OBE for S2A the script called is: ***reals_saving_1_obe.sh***.
Those scripts give as output a file, containing the incoming message as it is, whose name is the same as for the current architecture and respects the standardization as well.
Each 5 minutes the reals_cronjob.sh is called to handle the entire folder content.
Respecting the standardization of the file opened (i.e. eventREALS_1_obe_DOY###_etc.. rather than eventREALS_1_ool_DOY###_etc..), the reals_cronjob.sh calls the reals_action_1_obe.sh rather than the reals_action_1_ool.sh as functions giving as input to them the content of the text files.
The scripts called by the reals_cronjob.sh have been modified to give as output the SMS content detailed from the whitelists.
Is such a way the reals_cronjob.sh receives back directly the SMS content for the REALS server.
In case the Event/OOL requires a Phone-Call, the script called by the reals_cronjob.sh saves also a file containing the Phone-Call text.
The reals_cronjob.sh will handle this new text file as a common input and will give the Phone-Call text as input to the reals_action scripts.
Those scripts, receiving this input, will give as output the text itself that will be sent to the REALS server passing through the Testmission_1_action.preoperties.
The new architecture is structured as follows:


REALS
Ev. Log
ACTION APPLICATION
bash scripts to copy and paste
Action.dat
generated-event folder
whitelist
bash functions
CRONJOB script
Excel Interface
Testmission_1_action.properties
WODB
WEB MMI
REALS SERVER

14 RULES & CONSTRAINTS STRUCTURE

The content within the R&Cs file is the one of an XML, grouped in four main nodes:

14.1 R&C Header

Containing detailing information about the file itself (not updated).

14.2 List_of_Rules

made to assign to each event its sequences and relative location (SIOS, SIGR).
The content of the List_of_Rules, as it has been structured, is made of Rules specifying the sequences of commands for four different activities:
Planview_S_AOSD_StationName_TM
Planview_S_AOSD_StationName_TC
Planview_S_LOSD_StationName_TC
Planview_S_LOSD_StationName_TM
Each activity can be performed through each station available for the passes (look into the R&Cs file to have an idea) and contains nested requests, named (under the RQ_Name field field) as the sequence of commands to be performed for the activity.
The request contains other detailing fields:

- RQ_Type: to specify whether the sequence of commands is a “Reals-time sequence”, for Command Sequence Requests to be inserted in the Ground Schedule Increment (SIGR) or a “Time-tagged Sequence”, for Command Sequence Requests to be inserted in the MTL-tagged On-Board Schedule Increment (SIOS).
- RQ_Offset: to give an additive time gap to the sequence from the execution time\release time reference.
- Other fields standardly filled.

14.3 List_of_Constraints

Specifying details concerning the input files (SPF, PLANVIEW, NPPF, OPF).

14.4 List_of_PlanView_Events

Aimed to relate the scheduled timing from the PlanView file with the Rules (above).
To each Event in the List_of_PlanView_Events, whose structure is the same of the List_of_Rules, is related one (and only one) Rule having its same name:
Planview_S_AOSD_StationName_TM
Planview_S_AOSD_StationName_TC
Planview_S_LOSD_StationName_TC
Planview_S_LOSD_StationName_TM

15 PLANVIEW

The aim of the R&Cs file is to relate each activity, expressed in the PlanView file with a time reference, to the proper sequence(s) of commands to perform in order to accomplish that activity.
For each pass available for commanding, the PlanView file contains the time references of the starting and ending activity of TM and TC, specifying for them the relative ground station (e.g. SG20):

```
<service_session>

<osg_dn>Sentinel2B_1.SN2B_kiruna_contact.nominal_definition.SN2B</osg_dn>
   <satellite_id>SN2A</satellite_id>
   <ground_station>SG20</ground_station>
   <activity_start>2018-07-26T11:17:28.000Z</activity_start>
   <activity_end>2018-07-26T11:35:45.000Z</activity_end>
   <tracking_start>2018-07-26T11:19:28.000Z</tracking_start>
   <tracking_end>2018-07-26T11:34:45.000Z</tracking_end>
   <oss_ref>EVENT-2018.183.17.37.16.459103-1797455</oss_ref>
   <service_instance>
    <ssi_id>EVENT-2018.207.09.30.28.375164-713639</ssi_id>
    <service_type>telemetryReception</service_type>
    <activity_start>2018-07-26T11:19:35.000Z</activity_start>
    <activity_end>2018-07-26T11:34:30.000Z</activity_end>
   </service_instance>
   <service_instance>
    <ssi_id>EVENT-2018.207.09.30.28.375259-713640</ssi_id>
    <service_type>telecommandUplink</service_type>
    <activity_start>2018-07-26T11:20:44.000Z</activity_start>
    <activity_end>2018-07-26T11:33:29.000Z</activity_end>
   </service_instance>
  </service_session>
```

The MPS looks into the R&Cs file for the Event relative to the specific activity and ground station given by the PlanView.
Once found, the Event redirects the MPS to the Rule, having the same name of the Event itself.
As described into the previous chapter, the Rules contain the name of the procedures within the nested requests.
In such way, the MPS relates each activity timely defined to its proper sequence(s), destining it/them to the proper file (SIOS, SIGR).
Since in the very first step of the Mission Planning, the only input files taken in account to generate the SIOS and SIGR are the R&Cs, the PlanView and the WODB, the two timing references (“Reals-time sequence” and “Time-tagged Sequence”) are the only that can be inserted into the R&Cs file for each request.
Otherwise, e.g. for an “Orbital-angle tagged Sequence” (for Command Sequence Requests

to be inserted in the OPS-tagged On-Board Schedule Increment (SIOP)) also the NPPF and the OEV file would be needed.

16 SEQUENCES SCHEDULES

16.1 AOS TM

As mentioned, the sequences are specified in the <RQ_Name></RQ_Name> tag into the Rules.
The algorithm to be followed for each pass ha the same steps, specific for the ground station used for the link.
The very first step to perform is the acquisition of telemetry, whose rule has the following structure:

```
<Rule>
        <Rule_Name>Planview_S_AOSD_KIR1_TM</Rule_Name>
        <Rule_Description>
                Connect NIS Links
                Switch ON S-Band (STRPSET_SBY_TO_OPER)
        </Rule_Description>
        <List_of_RQs count="2">
                <RQ>
                        <RQ_Name>KR14_O</RQ_Name>
                        <RQ_Type>Real-time Sequence</RQ_Type>
                        <RQ_Description>Connect NIS Links</RQ_Description>
                        <RQ_Time_Offset>-60</RQ_Time_Offset>
                        <RQ_Frequency>1</RQ_Frequency>
                        <RQ_Expiration_Window>00T00:00:00</RQ_Expiration_Window>
                        <RQ_Repetition_Period>00T00:00:00</RQ_Repetition_Period>
                        <RQ_Subsystem>NIS</RQ_Subsystem>
                        <List_of_RQ_Parameters count="0">
                        </List_of_RQ_Parameters>
                </RQ>
                <RQ>
                        <RQ_Name>MTRFSBOP</RQ_Name>
                        <RQ_Type>Time-tagged Sequence</RQ_Type>
                        <RQ_Description>Switch on S-Band</RQ_Description>
                        <RQ_Time_Offset>-30</RQ_Time_Offset>
                        <RQ_Frequency>1</RQ_Frequency>
                        <RQ_Expiration_Window>00T00:00:00</RQ_Expiration_Window>
                        <RQ_Repetition_Period>00T00:00:00</RQ_Repetition_Period>
                        <RQ_Subsystem>S-BAND</RQ_Subsystem>
                        <List_of_RQ_Parameters count="0">
                        </List_of_RQ_Parameters>
                </RQ>
        </List_of_RQs>
</Rule>
```

The rule is firstly tagged with the name specifying the file where the activity comes from (PlanView), the S band through which the link is performed, the kind of activity (AOSD), the ground station available for the link and whether it is concerning an Up-link (TC) or a Down-link (TM).
The Rule description is only used to let the Rule being easier understandable.
In the list of request, the fields that have been set to a specific value specify the name of the sequences, the type of time reference, a time offset from the scheduled time from PlanView.

Other fields have been set to a standard value or filled with describing info for the user.
For AOSD TM the activity to perform is double:

1. The first part sets the sequence for the NIS link connection with the antenna to a release time with an offset of 60 seconds before the expected AOSD TM.
2. The second part sets the sequence to change the status of the S-Band transponder from stand-by to Operative to an execution time with an offset of 30 seconds before the expected AOSD TM.

Those two sequences are the only having the AOSD TM as reference to be performed at the occurrence of each pass.
Since there is a different sequence for connecting the NIS link with each different ground station, a Rule is defined for each ground station (in the example above, Kiruna1).

16.2 AOS TC

After the acquisition of telemetry, the sudden step is to achieve the command link with the S/C.
A Rule is defined for this activity but, until now, not a request was defined within it since the commands to perform at the AOSD TC, the Test command and the download of the System Log and the PSs, are operated manually.

```
<Rule>
        <Rule_Name>Planview_S_AOSD_KIR1_TC</Rule_Name>
        <Rule_Description></Rule_Description>
        <List_of_RQs count="0"/>
</Rule>
```

16.3 LOS TC

As well for the LOSD TC, the commands to perform, Stop of all the PS download and their deletion, are operated manually and not a request is created.

```
<Rule>
        <Rule_Name>Planview_S_LOSD_KIR1_TC</Rule_Name>
        <Rule_Description></Rule_Description>
        <List_of_RQs count="0"/>
</Rule>
```

16.4 LOS TM

At the LOSD TM the activity is symmetric:

1. The NIS link with the ground stations are closed giving to the proper sequence a release time reference with an offset of 15 seconds after the expected LOSD TM.
2. The S-Band transponder switches from the operational mode to stand-by mode giving to the sequence an execution time reference with an offset of 30 seconds after the expected LOSD TM.

```
<Rule>
    <Rule_Name>Planview_S_LOSD_KIR1_TM</Rule_Name>
    <Rule_Description>
        Connect NIS Links
        Switch ON S-Band (STRPSET_SBY_TO_OPER)
    </Rule_Description>
    <List_of_RQs count="2">
        <RQ>
            <RQ_Name>KR14_C</RQ_Name>
            <RQ_Type>Real-time Sequence</RQ_Type>
            <RQ_Description>Disconnect NIS Links</RQ_Description>
            <RQ_Time_Offset>15</RQ_Time_Offset>
            <RQ_Frequency>1</RQ_Frequency>
            <RQ_Expiration_Window>00T00:00:00</RQ_Expiration_Window>
            <RQ_Repetition_Period>00T00:00:00</RQ_Repetition_Period>
            <RQ_Subsystem>NIS</RQ_Subsystem>
            <List_of_RQ_Parameters count="0">
            </List_of_RQ_Parameters>
        </RQ>
        <RQ>
            <RQ_Name>MTRFOPSSB</RQ_Name>
            <RQ_Type>Time-tagged Sequence</RQ_Type>
            <RQ_Description>Switch on S-Band</RQ_Description>
            <RQ_Time_Offset>30</RQ_Time_Offset>
            <RQ_Frequency>1</RQ_Frequency>
            <RQ_Expiration_Window>00T00:00:00</RQ_Expiration_Window>
            <RQ_Repetition_Period>00T00:00:00</RQ_Repetition_Period>
            <RQ_Subsystem>S-BAND</RQ_Subsystem>
            <List_of_RQ_Parameters count="0">
            </List_of_RQ_Parameters>
        </RQ>
    </List_of_RQs>
</Rule>
```

Those two sequences are the only having the LOSD TM as reference to be performed at the occurrence of each pass.
Since there is a different sequence for disconnecting the NIS link with each different ground station, a Rule is defined for each ground station (in the example above, Kiruna1).

17 MRSLPSDP & MRSMMSUS

CSS_SQNAME	CSS_COMM	CSS_ENTRY	CSS_TYPE	CSS_ELEMID	CSS_NPARS	CSS_MANDISP	CSS_RELTYPE	CSS_RELTIME	CSS_EXTIME	CSS_PREVREL	CSS_ILSCOPE	CSS_ILSTAGE	CSS_DYNPTV	CSS_STAPTV	CSS_CEV	SDB_IMPORTED
MRSLPSDP	Perform Connect Test	1	C	CSC00078	0	Y	R	00.00.01		R	L	C	N	N	N	No
MRSLPSDP	Downlink System Log	2	C	CSC00035	1	N	R	00.00.01		R	L	C	N	N	N	No
MRSLPSDP	StartPS_1PLP_1 downlink	3	C	ZSC10165	0	N	R	00.00.05		R	L	C	N	N	N	No
MRSLPSDP	StartPS_3PLP_1 downlink	4	C	ZSC10365	0	N	R	00.00.05		R			N	N	N	No

CSS_SQNAME	CSS_COMM	CSS_ENTRY	CSS_TYPE	CSS_ELEMID	CSS_NPARS	CSS_MANDISP	CSS_RELTYPE	CSS_RELTIME	CSS_EXTIME	CSS_PREVREL	CSS_ILSCOPE	CSS_ILSTAGE	CSS_DYNPTV	CSS_STAPTV	CSS_CEV	SDB_IMPORTED
MRSMMSUS	Stop pk Stores downlink	1	C	CSC00064	0	N	R		00.00.00	R			N	N	N	No

MRSLPSDP contains 4 commands to be uplinked at the beginning of the ground station pass:

17.1 CSC00078, Test command

Remarks:

- The command has 1-second-delay between the absolute Release-Time coming from the Planview file set 30 seconds after the AOS TC. i.e. The command will be released 31 seconds after AOS TC.
- The command is Inter-locked with the following one.

17.2 CSC00035, Downlink of the System Log

Remarks:

- The command has 1-second-delay between the Release-Time of the previous command.
- At the occurrence of the Release-Time of the command it is not released until the reception of the packet (1, 7) of CSC00078.
  The Validity-Release-Window is 10 seconds, after that time, if the Interlock of command CS00078 is not verified the command CSC00035 will fail.
  The command is also Inter-Locked with the following one.
- The parameter within the command (CSP00007 = 01) identifies the primary Safe Guard Memory (SGM_A)

17.3 ZSC10165, Start of PS1 Downlink

Remarks:

- The command has 5-second-delay after the Release-Time of the previous command.
  This value is greater than the maximum average time needed for the downlink of the System Log.
- At the occurrence of the Release-Time of the command it is not released until the reception of the packet (1, 7) of CSC00035.
  The Validity-Release-Window is 10 seconds as well (MCS setting).
  The command is Inter-Locked with the following one.

17.4 ZSC10365, Start of PS3 Downlink

Remarks:

- The command has 5-second-delay after the Release-Time of the previous command.
  This value is greater than the maximum average time needed for the downlink of the PS3.
- At the occurrence of the Release-Time of the command it is not released until the reception of the packet (1, 7) of ZSC10165.

MRSMMSUS contains 1 TT command to be executed 30 seconds before the LOS TC.

17.5 CSC00064, Stop PS downlink.

Remarks:

- The command stops any current downlink of PSs/Sys Log at the epoch of its execution.
- The command is not inter-Locked.
- Even if no PS downlink is occurring at the epoch of the execution, the command will be executed successfully.

The absolute release time for MRSLPSDP and the absolute execution time for MRSMMSUS are defined into the Planview file and set with delay coming from the R&C file.

18 R&C MODIFICATION

The modifications that affected the R&C files, for both S2A and S2B, included the two sequences MRSLPSDP (Test command, downlink PS1, PS3 and Sys. Log) and MRSMMSUS (Stop PSs downlink).
A detailed description of the two sequences:

\\escwindfs01\GMES Sentinels FOS$\Sentinel-2\Operations\Automation\MPS\R&Cs_Modification\Summary_sequences.pdf

18.1 MRSLPSDP

The MRSLPSDP sequence is released 30 seconds after the AOS TC.
This condition is achieved including the sequence under each AOS TC Rule (defined above) and setting the Request Type field to:

<RQ_Type> Time-tagged Sequence</RQ_Type>

The "Time-tagged Sequence" is interpreted by the MPS as the instruction to handle the request as a Release-Time-Sequence that will be included into the SIGR.
Furthermore, the Request Time Offset is set at *+30* seconds to add a delay to the absolute time from the Planview:

<RQ_Time_Offset>30</RQ_Time_Offset>

18.2 MRSMMSUS

The MRSMMSUS sequence is released 30 seconds before the LOS TC.
This condition is achieved including the sequence under each LOS TC Rule (defined above) and setting the Request Type field to:

<RQ_Type> Time-tagged Sequence</RQ_Type>

The "Time-tagged Sequence" is interpreted by the MPS as a sequence to be included into the SIOS.
Furthermore, the Request Time Offset is set at *-30* seconds to the absolute time from the Planview:

<RQ_Time_Offset>-30</RQ_Time_Offset>

19 CONCLUSIONS

The first and main objective of the study was to configure a software that would allow engineers to be informed in real time about the specific anomalies present on board the spacecraft in order to monitor and, if necessary, suddenly react to repair the conditions of the spacecraft. The work carried out in the months spent at the European Space Operations Centre (ESOC) has been a challenge from various points of view and has helped to study, experiment and make operational a concrete and reliable solution for the supervision of Sentinel-2. The activity, carried out in strong synergy with the engineers of the Sentinel missions (then of the Copernicus Program) and of the missions involved in the implementation of REALS software (in particular the GAIA mission), progressed step by step because of the complexity of the issues to be solved.
The first phase took several weeks to become familiar with the Mission Control System, as the source of the inputs, as well as for the use of the simulator, for testing and validation, before being able to implement the software. All this in order to acquire a wide and clear vision of the characteristics and languages, and consequently of the potential that such software owns, before implementing it in the mission, in order to make the real customization effective and optimal. The last phase, but not less complex than the previous ones, required to implement the software into a highly detailed and complex environment, such as the operations. This required a further effort to understand all the elements involved, both directly and indirectly.
Before the introduction of REALS software in fact, the engineer On-Call was aware of the anomalies on board the spacecraft, downlinked within the telemetry, only after having reached the workstation. This could lead to operational difficulties since, in the first analysis, the engineer had to search among the telemetry those events that represented anomalies and, among these, identify the most important and most interesting that required urgent interventions. Only later the engineer would have been able to react for solving of the anomalies. The introduction of REALS allowed making a selection of the telemetry sent by the spacecraft and then to identify and extrapolate from a very broad list the most relevant elements on the base of their content.
The most logical approach followed was to create, almost from scratch, a new telemetry filtering methodology to meet the specific needs of the Flight Control Team.
It was therefore necessary to identify a standard inputs structure, so to adapt the scripts to the structure they were supposed to handle.
Two different patterns were identified: the structure of the messages referring to the Out Of Limits and the one referring to the services (5, 4).
According to this, two different scripts were created for the respective filtering.
This phase was decisive and not easy to solve, as it required a knowledge of all the entities present and only the work in synergy with the engineers allowed create a list of the information of interest.
In a first phase was considered an SMS notification, but soon became clear that such a lonely notification was not reliable for a safe awareness of the On-Call Engineer. The need for telephone notification for specific events became then necessary.

To satisfy this need, it was decided to create a double notification (Telefonica + SMS) in order to alert the engineer and provide, at the same time, a clear text description of the problem.
A long testing campaign has proven the reliability of the software.
One of the problems that rised during the project, which perhaps represented a purely technical challenge, is that of the "toggling" (oscillations) of the parameters: their repeated passage into / out of the nominal limits. This oscillation would have led to repeated and superfluous notifications of the same occurrence generating overlaps and loss of time.
The solution has been entrusted in part also to the engineer's knowledge about the specified parameters. It would have been the engineer to specify how each parameter was affected by toggling, introducing this measure as a reference for its notification.
At the end of October the project has become operational and is allowing the supervision of the mission by engineers rather than the Spacecraft Controller beyond the limits previously linked to the physical presence of the operator in the control room. This allows an immediate intervention; does not involve an increase in management costs; makes the operation smoother and reduces investigation and problem solving times. This system can now be deployed to the future Sentinel missions.
Only after having become familiar with the operations environment was it possible to develop the second part of the project. That is to say: the automation of the commands that required a deeper knowledge of the Mission Planning System and of the files involved in the generation of the command stacks in order to make changes. [To be completed]