Horizon 2020

Project acronym:	**LASH FIRE**
Project full title:	**Legislative Assessment for Safety Hazard of Fire and Innovations in Ro-ro ship Environment**
Grant Agreement No:	**814975**
Coordinator:	**RISE Research Institutes of Sweden**

Deliverable D08.13

Overall integration with firefighting control centre

June 2023

Dissemination level: **Public**

Call identifier: H2020-MG-2018-Two-Stages Starting date: 2019-09-01 Duration: 48 months
MG-2.2-2018: Marine Accident Response, Subtopic C

Abstract

The Stowage Planning Tool (SPT) is one of the Risk Control Options envisaged in LASH FIRE from the ignition prevention perspective. The SPT is a software solution that includes fire hazard management aiming at supporting the stowage process by means of suggesting an alternative cargo distribution. The proposed cargo distribution takes advantage of a risk assessment for every single unit based on historical data with the objective of reducing the overall risk in ro-ro spaces.

Since such a software manages information about the cargo, including physical characteristics, type or accurate location of their placement in the ship, it also plays a relevant role when it comes to provide valuable support to firefighting after departure.

The present deliverable describes the implementation of a specific use case of the Stowage Planning Tool that aims at supporting the integration with the firefighting control centre, also known as Fire Resource Management Centre, by means of data sharing.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 814975

The information contained in this deliverable reflects only the view(s) of the author(s). The Agency (CINEA) is not responsible for any use that may be made of the information it contains.

Document data

Document Title:	LASH FIRE_D08.13_Overall_integration_with_firefighting_control_centre_V1.docx		
Work Package:	WP08 – Ignition		
Related Task(s):	T08.11. T08.5, T08.7, T08.8, T07.10		
Dissemination level:	Public		
Deliverable type:	R		
Lead beneficiary:	CIM		
Responsible author:	Francisco Rodero		
Co-authors:	-		
Date of delivery:	2023-06-20		
References:	D07.6, D07.11, D08.4		
Approved by	Erik Styhr on 2023-06-16	Robert Rylander on 2023-06-05	Maria Hjohlman on 2023-05-29

Involved partners

No.	Short name	Full name of Partner	Name and contact info of persons involved
16	CIM	Centre Internacional de Metodes Numerics en Enginyeria	Francisco Rodero, francisco.rodero@upc.edu

Document history

Version	Date	Prepared by	Description
[00]	2021-04-15	Ángel Priegue	Draft of structure
[01]	2023-04-24	Francisco Rodero	Final report
[02]	2023-05-25	Francisco Rodero	Final report reviewed
[03]	2023-06-20	Francisco Rodero	Final report approved

Content

1 Executive summary 4
1.1 Problem definition 4
1.2 Technical approach 4
1.3 Results and achievements 4
1.4 Contribution to LASH FIRE objectives 4
1.5 Exploitation 5
2 List of symbols and abbreviations 5
2.1 Abbreviations 5
3 Introduction 6
4 *Fire Patrol Report* feature 7
4.1 Specification 7
4.2 Inputs 7
4.2.1 Configuration parameters 7
4.3 Outputs 9
4.4 Design 9
4.4.1 Data model 9
4.4.2 Workflow 10
5 Implementation details 11
5.1 Application programming interface 11
5.2 Error management 11
5.3 Fire Patrol Report 12
6 Test methodology 13
6.1 FirePatrolReport 13
6.1.1 Correct management of the parameters 13
6.1.2 Construction of the query to the database 14
6.1.3 Generation of the output 16
7 Conclusions 17
8 References 18
9 Indexes 19
9.1 Index of tables 19
9.2 Index of figures 19

1 Executive summary

Main author of the chapter: Francisco Rodero, CIM

1.1 Problem definition

When a fire happens, several actors and technical systems simultaneously collaborate to achieve many operational goals. During these situations, it is difficult to get an overview of the status and available resources and therefore, one of the goals of LASH FIRE, in terms of inherently safe design for ships, is the design of the concept for a firefighting resource management centre.

The Fire Resource Management Centre (FRMC), described in D07.8 "Design definition and development of firefighting resource management simulator prototype", aims at supporting critical operations in case of fire to reduce the potential for human error, accelerating time sensitive tasks and providing a more comprehensive decision support.

This way, to enhance the effectiveness of the FRMC and increasing the capacity to provide a powerful resource for firefighting operations, is it critical to concentrate as much as meaningful information possible together with, of course, a careful design. In that sense, the SPT plays a critical role in generating the data needed for the digital fire centre to be able to create a fire patrol report.

1.2 Technical approach

Feeding the FRMC with useful data is addressed by means of the implementation of one of the envisaged use cases of the SPT as defined in deliverable D08.4 "Stowage planning optimization and visualization aid", *Fire Patrol Report*, which objective is the generation of data needed by external systems to prepare specific reports for fire patrol purposes.

1.3 Results and achievements

The Stowage Planning Tool has been successfully extended with the implementation of the above-mentioned *FirePatrolReport* feature, allowing the SW to share information about the accurate location of the cargo units along the decks, cargo type, risk score and additional references to the nearby units.

The output of the new feature contains relevant information that can be mainly used by the FRMC to create fire patrol reports as well as to easily know what is the cargo nearby an area where an eventual ignition is detected. Also, the information can be used by the Automated Guided Vehicle (AGV) to design and plan their paths according to the actual cargo. Indirect integration of automatic screening of the units and continuous monitoring of the electrical charging infrastructure is also possible, which means that systems and tools developed in WP8 can eventually be integrated in the firefighting control centre as expected.

1.4 Contribution to LASH FIRE objectives

With this feature, the SPT preserves the contribution to the project as described in D08.4, that is, fire risk is mitigation by a safety-optimized usage of deck space and eventually reduction of consequences in case of fire by a cargo distribution with lower risk score based on historical data.

Also, the integration with the FRMC extends the support of the SPT from the Ignition Prevention stage to the Extinguishment stage, in terms of the fire protection chain.

Besides this, there is also a clear contribution to the IMO Strategic Plan 2018-2023, where integration of new and advanced technologies in the regulatory framework is strongly recommended.

1.5 Exploitation

The exploitation of this functionality is closely linked to the use of the FRMC concept and with benefits for involved crew on the bridge during the management of fire situations (fire reports) or even for supporting preventive patrolling (path planning based on cargo contents) using automated guided vehicles.

2 List of symbols and abbreviations

2.1 Abbreviations

AGV	Automated Guided Vehicles
API	Application Programming Interface
DB	Database
DFC	
DG	Dangerous Goods
FRMC	Fire Resource Management Centre
IMDG	International Maritime Dangerous Goods
JSON	JavaScript Object Notation
RA	Risk Assessment
RS	Risk Score
SPT	Stowage Planning Tool
SQL	Structured Query Language
SW	Software
VHD	Vehicle Hot-Spot Detection

3 Introduction

Main author of the chapter: Francisco Rodero, CIM

Although the technical solutions that have been developed in LASH FIRE focus on specific stages of the fire protection chain, some synergies can be found from the overall perspective. The integration of the SPT with the FRMC, not a physical place but a collection of tools and methods to manage firefighting effectively, is a clear example of these synergies.

The next diagram depicts the whole architecture of the SW, including the envisaged interfaces to both external and internal components on the right side.

Figure 1 - Software components of the Stowage Planning Tool

Focusing the risk reduction in case of ignition, the SPT supports the stowage process not only during the pre-loading stage but also during the stowage process, where some situations that may alter the suggested cargo distribution can be found. For example, cargo units are not available to be loaded when they should (based on the suggested cargo distribution and the current loading status) because they have not just yet arrived to the terminal, or they will not (no-show), or even because an alarm has been triggered in the VHD when inspecting the unit; the latter uses the VHD interface to notify the SPT that the unit will not be loaded or will be loaded with special treatment after cargo office approval, which means that the risk score of the unit increases.

While the VHD interface is an input interface to the SPT, the DFC/FRMC and rolling drones (AGV) are output interfaces:

- Information about placement of cargo units and their characteristics can be shared with the DFC/FRMC to help generating fire patrol reports.
- The same information, or filtered subsets of this information, can be requested by the AGVs for path planning purposes. That is, routes followed by the rolling drones can be created based on the actual cargo distribution rather than just patrolling everywhere.

These output interfaces provide accurate, up-to-date cargo stowage information to the 'Digital Fire Central' developed in D7.11 "Firefighting resource management simulator prototype" and thus ensuring that the work undertaken in the Fire Resource Management Centre is fully informed of what is burning and what may catch fire next. Fire patrol reports can include contents like:

- Inspection schedules for specific areas containing DG or other hazardous materials.
- Use of certain fire suppression systems or extinguishment methods in specific areas.

- Additional safety measures for cargo with high-risk score, e.g., if special firefighting equipment, personal protection equipment or response procedures should be considered due to the cargo.

4 *Fire Patrol Report* feature

Main author of the chapter: África Marrero and Francisco Rodero, CIM

The feature being described here concerns the implementation of the use case UC#13 which generates the data needed by an external system to prepare a specific report for fire patrol purposes.

It is important to remark that this implementation is not a stand-alone development but an extension of the SPT. This means that this deliverable only documents the modifications made on the original software while all technical details of the SPT as defined in D08.4 still apply.

4.1 Specification

There is just one additional explicit requirement that extends the current definition of the SPT.

Table 1.List of requirements

Identifier	Description
REQ300	The system will implement an interface to external software in order to export up-to-date information about the location, type (including DG class if needed), risk score and nearby units for each single unit of an existing cargo distribution.

4.2 Inputs

4.2.1 Configuration parameters

The following table contains updated information, whenever is necessary, respect to the parameters defined in D08.4. Again, to highlight that definitions are cumulative in the sense that which is not override here, still applies.

Table 2.List of additional considerations about configuration parameters respect to D08.4

Name	Description and valid values
Service	An additional use case is defined: • *FirePatrolReport*: The system returns information as defined in UC#13 / REQ300.
IdService	Numerical parameter (greater than 1) that identifies one service that has been already executed. This is the identifier stored in the database.
deck	Filters the result by a specific deck
lane	Filters the result by a specific lane
frame_start	Filters the result by a frame value greater or equal than this value
frame_end	Filters the result by a frame value lower or equal than this value
type	Filters the result by a cargo type (from MT_FIREORIGIN2 table as defined in D08.4)
dg_class	Filters the result by a DG class (from MT_DG_CLASS table as defined in D08.4)

The following parameters: *deck, lane, frame_start, frame_end, type* and *dg_class,* are cumulative when it comes to filtering. That means that the resulting subset of units must satisfy all the filters. For example, if the service is executed with *deck=3*, *frame_end=187* and *dg_class=4.1*, the list of returned units must satisfy that they are located in *deck=3* AND in a slot defined by *frame_start* and *frame_end* where the value for the *frame_end* is 187 as a maximum AND all of them are flammable solids 4.1 dangerous goods class.

The following table shows what parameters are required for this service to run properly:

Table 3.List of required parameters for FirePatrolReport service

Parameter	***Service***
	FirePatrolReport
Service	Yes
ServiceDescription	Optional
IdService	Yes
Ship	No
Layout	No
Route	No
SlotError	Yes
Sep_X	Yes
Sep_Y	Yes
timeout	No
Improvement	No
IsTest	-
IdTest	-
Deck	Optional
lane	Optional
frame_start	Optional
frame_end	Optional
type	Optional
dg_class	Optional

4.3 Outputs

This is the information which is shared with the FRMC (via the DFC) or AGV for them to create fire patrol reports or path plans, respectively.

- Cargo unit identifier: Unique identifier of the unit itself.
- Type: Value for the cargo type as defined in MT_FIREORIGIN2 table.
- DG class: Optional. In case of a DG, it contains the IMDG classification.
- Deck, lane, frame_start/frame_end: Accurate location of the unit (placement slot).
- Score values: Two values defining the initial risk score (type dependent) and the final risk score (location dependent) as per the risk assessment based on historical data.
- List of nearby units: According to *Sep_X* and *Sep_Y*, the list includes the nearby units in the same deck.

Please note that although not included in the demonstration and testing, both *alarm* (triggered by the unit when passing the VHD) and *id_connection* (identifier for the electrical connection) fields can also be easily added to the output since they are actually defined as part of the SERVICE_UNITS table against which the query is executed.

Notice that the last 6 optional parameters in the above table allow to customize the request for specific locations. Since the service can be called many times with different values for parameters, whoever request the information can combine the results as needed. Two examples could be:

- The DFC in the FRMC receives an alarm from the connected Fire Detection System, gets the location of the sensor which has triggered the alarm and then it creates a request based on this location to get all the units in a surrounding area of about 50 meters.
- An AGV, equipped with a specific gas sensor, requests for the units of classes 2.1, 2.2 and 2.3 that are inside a limited area because it is not able to navigate beyond these limits unless it recharges battery over 50%.

The list of nearby units is provided to add value in terms of potential fire propagation beyond the specific area requested.

4.4 Design

4.4.1 Data model

The underlying database has not changed but a small modification has been implemented in order to improve the performance of the execution. More concretely, two new fields have been added in the SERVICE_UNITS table:

Table 4. Additional fields of SERVICE_UNITS table

Attribute	Type	References	Description
type	TXT	MT_FIREORIGIN2.uid	Value based on FIRE_ORIGIN2
dg_class	TXT	MT_DG_CLASS.uid	Value based on MT_DG_CLASS

4.4.2 Workflow

4.4.2.1 FirePatrolReport

Diagram in Figure 2 depicts the main actions taken during the execution of the service before sending back the generated output to the requester.

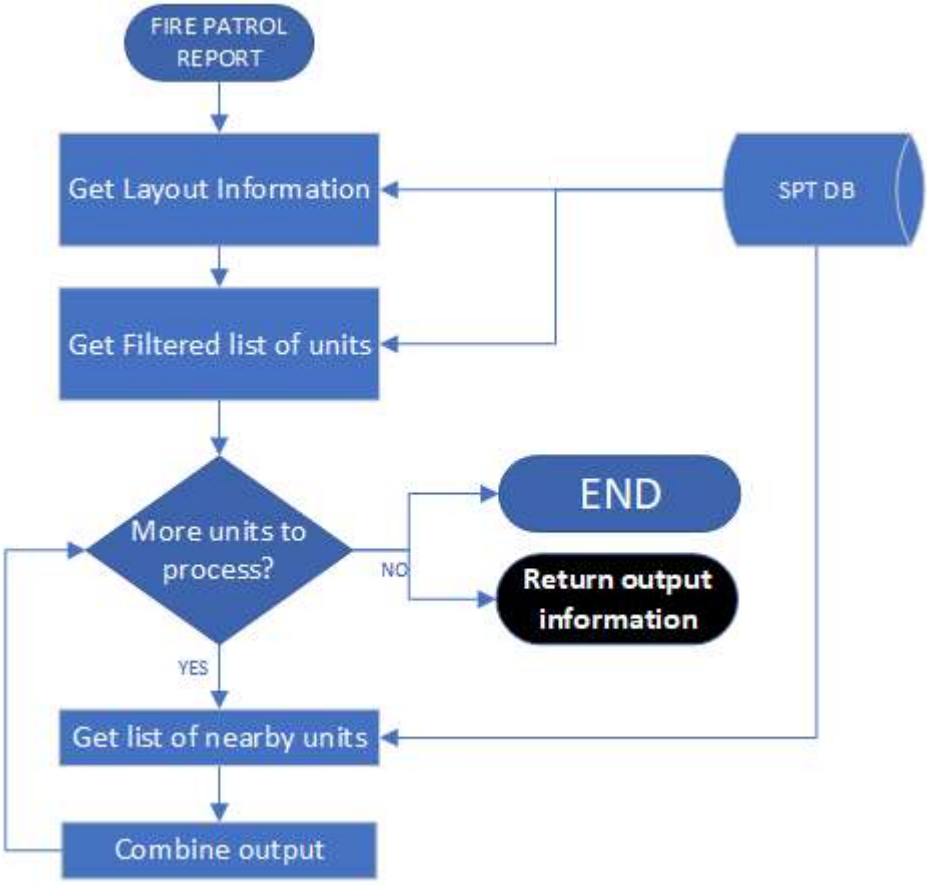

Figure 2. Workflow of FirePatrolReport service

4.4.2.1.1 Get Layout Information

Output contains information that depends on physical layout of the ship, so the first step retrieves this information from the database taking into account the service identifier passed as one of the configuration parameters.

4.4.2.1.2 Get Filtered list of Units

Filters used as arguments for the request, if any, are combined with the full list of units to generate the subset the user is interested on.

4.4.2.1.3 Get list of nearby units / Combine output

For each filtered unit, the software parses the physical layout and all the existing units to create a list of nearby units.

5 Implementation details

Main author of the chapter: Francisco Rodero, CIM

The next table includes an update of the files that compose the software:

Table 5. Folders and files of the software development

Path	File name	Description
/uc	firepatrolreport.py	Specific implementation for *FirePatrolReport* use case

The next sections include the most relevant tips concerning the development using parts of the code:

5.1 Application programming interface

In the same way that the visual interface communicates with the implementation of the core components of the SPT using a lightweight RESTful API with *Flask* and the *Python* code, both FRMC and AGV developments can do the same.

The API has been extended with the entry point to the implementation:

```
@app.route('/firepatrolreport', methods = ['GET'])
def firepatrolreport():
```

This way, interaction with the *FirePatrolReport* service via the API uses HTTP requests like the following:

```
http://XXX.YYY.ZZZ.TTT:5000/firepatrolreport?IdService=2&SlotError=0.1&Sep_X=6&Sep_Y=3
```

The previous example is calling the service just with the mandatory parameters. A full request using all parameters could be as follows:

```
http://XXX.YYY.ZZZ.TTT:5000/firepatrolreport?IdService=2&SlotError=0.1&Sep_X=6&Sep_Y=3&deck=5&lane=93&frame_start=6&frame_end=84&type=CAR&dg_class=4.3
```

5.2 Error management

The below table includes the only error message that has been added to support the service:

Table 6.Implemented errors/warnings messages

Type	Group	ID	Description
E	0	200	Parameter {deck, lane} must be integer greater than 0

5.3 Fire Patrol Report

Since it is an extension of the actual SPT the implementation of the *FirePatrolReport* uses lot of existing code concerning the list of nearby units (which was implemented to support the Score service) and two new functions: the one combining all the information retrieved from the database to create the output (two *Python* dictionaries are used to write the output file which is, then, parsed by the *API* to send back the information) and the query to the database, which looks like as follows:

```
# returns list of units that have been processed by service_id in a format
# which is focused on the Fire Resource Management Centre and/or the Path
# Planning for the AGVs
def RS_getServiceUnitsFirePatrolReport(cfg):
    result = {}
    CURSOR = CONNDB.cursor()
    if CURSOR != None:
        # Location attributes get from SERVICE_UNITS since it is the table
        # that stores the final distribution
        query = "SELECT id_cargo_unit, type, dg_class, id_deck, id_lane,
frame_start, frame_end, RS0, RS"
        query += " FROM SERVICE_UNITS WHERE"
        query += " id_service=" + str(cfg["Parameters"]["IdService"])
        try:
            deck = cfg["Parameters"]["deck"]
            query += " AND id_deck=" + str(deck)
        except:
            None
        try:
            lane = cfg["Parameters"]["lane"]
            query += " AND id_lane=" + str(lane)
        except:
            None
        try:
            frame_start = cfg["Parameters"]["frame_start"]
            query += " AND frame_start>=" + str(frame_start)
        except:
            None
        try:
            frame_end = cfg["Parameters"]["frame_end"]
            query += " AND frame_end<=" + str(frame_end)
        except:
            None
        try:
            type = cfg["Parameters"]["type"]
            query += " AND type='" + str(type) + "'"
        except:
            None
        try:
            dg_class = cfg["Parameters"]["dg_class"]
            query += " AND dg_class='" + str(dg_class) + "'"
        except:
            None
        query += ";"
        CURSOR.execute(query)
        for record in CURSOR:
            try:
                result[record[0]]
            except:
                result[record[0]] = []
```

```
            result[record[0]]  =  [record[1],  record[2],  record[3],  record[4],
record[5], record[6], record[7], record[8]]
    else:
        print("RS_getServiceUnitsFirePatrolReport::Error creating cursor!")
    return result
```

The function creates a SQL query where parameters are used to filter the results of the query.

6 Test methodology

Main author of the chapter: Francisco Rodero, CIM

6.1 FirePatrolReport

The next steps have been followed to test that the feature works properly:

1. Verification of the correct management of parameters
2. Verification of the correct construction of the query to the database
3. Verification that the output contains expected contents and format

The way the SPT has been designed and implemented makes that most of the tests mentioned above have been indirectly verified during the testing of both *Score* and *Distribution* services.

6.1.1 Correct management of the parameters

Two tests have been conducted, first omitting one the mandatory parameters (”*IdService*”) and then setting a wrong value for one of the optional parameters (”*frame_start*”).

```
Execution finished in 0.00034046173095703125 seconds
PS G:\TEMP\LASHFIRE\SW> & "C:/Program Files/Python311/python.exe" g:/TEMP/LASHFIRE/SW/lashfire_
spt.py
initialization of module: db
Connection successfully
end initialization of module: db
initialization of module: error
end initialization of module: error
initialization of module: cfg
ERROR 3.1 : IdService is mandatory for Services {RemoveService, FirePatrolReport} [None]
end initialization of module: cfg
initialization of module: input
end initialization of module: input
DO NOT CONTINUE
Execution finished in 0.0009989738464355469 seconds
PS G:\TEMP\LASHFIRE\SW> & "C:/Program Files/Python311/python.exe" g:/TEMP/LASHFIRE/SW/lashfire_
spt.py
initialization of module: db
Connection successfully
end initialization of module: db
initialization of module: error
end initialization of module: error
initialization of module: cfg
ERROR 1.17 : Invalid value for frame_start or frame_end [3ff]
end initialization of module: cfg
initialization of module: input
end initialization of module: input
DO NOT CONTINUE
Execution finished in 0.0010058879852294922 seconds
PS G:\TEMP\LASHFIRE\SW>
```

Figure 3. Output for tests checking for a correct management of the parameters

As shown above, the execution of the SW detects both errors and stops running, which is the expected behaviour.

6.1.2 Construction of the query to the database

Two tests have been conducted, first omitting submitting only mandatory parameters and the second using all available parameters. The source code has been modified temporarily to print the string to the console.

```
PS G:\TEMP\LASHFIRE\SW> & "C:/Program Files/Python311/python.exe" g:/TEMP/LASHFIRE/SW/lashfire_
spt.py
initialization of module: db
Connection successfully
end initialization of module: db
initialization of module: error
end initialization of module: error
initialization of module: cfg
{'Parameters': {'Service': 'FirePatrolReport', 'SlotError': 0.4, 'IdService': 6, 'Sep_X': 6, 'S
ep_Y': 3, 'ServiceDescription': 'Not provided', 'IsTest': False}}
end initialization of module: cfg
initialization of module: input
end initialization of module: input
Now accessing entrypoint for use cases...
Implementation of FIRE PATROL REPORT use case...
insertService::Transaction executed successfully!
Service: FirePatrolReport created with id: 9
SELECT id_cargo_unit, type, dg_class, id_deck, id_lane, frame_start, frame_end, RS0, RS FROM SE
RVICE_UNITS WHERE id_service=6;
updateService::Transaction executed successfully!
Generating output files...
Execution finished in 0.42888545989990234 seconds
PS G:\TEMP\LASHFIRE\SW> & "C:/Program Files/Python311/python.exe" g:/TEMP/LASHFIRE/SW/lashfire_
spt.py
initialization of module: db
Connection successfully
end initialization of module: db
initialization of module: error
end initialization of module: error
initialization of module: cfg
{'Parameters': {'Service': 'FirePatrolReport', 'SlotError': 0.4, 'IdService': 6, 'Sep_X': 6, 'S
ep_Y': 3, 'lane': 43, 'deck': 3, 'frame_start': 100, 'frame_end': 200, 'type': 'TRAILER', 'dg_c
lass': '2.2', 'ServiceDescription': 'Not provided', 'IsTest': False}}
end initialization of module: cfg
initialization of module: input
end initialization of module: input
Now accessing entrypoint for use cases...
Implementation of FIRE PATROL REPORT use case...
insertService::Transaction executed successfully!
Service: FirePatrolReport created with id: 10
SELECT id_cargo_unit, type, dg_class, id_deck, id_lane, frame_start, frame_end, RS0, RS FROM SE
RVICE_UNITS WHERE id_service=6 AND id_deck=3 AND id_lane=43 AND frame_start>=100 AND frame_end<
=200 AND type='TRAILER' AND dg_class='2.2';
updateService::Transaction executed successfully!
Generating output files...
Execution finished in 0.45557332038879395 seconds
PS G:\TEMP\LASHFIRE\SW>
```

Figure 4. Output for tests checking a correct construction of the query

As shown above, the execution of the SW correctly manages the parameters to construct the next two SQL queries, which is the expected behaviour:

```
SELECT id_cargo_unit, type, dg_class, id_deck, id_lane, frame_start, frame_end, RS0,
RS FROM SERVICE_UNITS WHERE id_service=6;
```

```
SELECT id_cargo_unit, type, dg_class, id_deck, id_lane, frame_start, frame_end, RS0,
RS FROM SERVICE_UNITS WHERE id_service=6 AND id_deck=3 AND id_lane=43 AND
frame_start>=100 AND frame_end<=200 AND type='TRAILER' AND dg_class='2.2';
```

6.1.3 Generation of the output

To verify that output is generated as expected, the contents of the output files are checked after the same tests of the previous section. First, using only mandatory parameters results on a list of all available units included in the reference input file:

```
2001;TRAILER;;2;18;102.0;110.5;1;1.0;2002,2003,2004,2005,2006,2007,2008,2009
2002;TRAILER;;2;16;100.0;109.0;1;1.0;2001,2003,2004,2005,2006,2007,2008,2009
2003;TRAILER;;2;16;110.0;119.0;1;1.0;2001,2002,2005,2006,2009
2004;TRAILER;;2;16;90.0;99.0;1;1.0;2001,2002,2005,2007,2008
2005;TRAILER;;2;17;100.0;109.0;1;1.0;2001,2002,2003,2004,2006,2007,2008,2009
2006;TRAILER;;2;17;110.0;119.0;1;1.0;2001,2002,2003,2005,2009
2007;TRAILER;;2;17;90.0;99.0;1;1.0;2001,2002,2004,2005,2008
2008;CAR;;2;15;86.0;95.0;2;2.0;2001,2002,2004,2005,2007
2009;V;;2;18;111.0;120.0;3;3.25;2001,2002,2003,2005,2006
3001;TRAILER;;3;44;115.0;124.0;1;1.0;3004,3005,3007,3008,3009
3002;TRAILER;;3;40;86.0;95.0;1;1.0;3003,3004,3006,3008,3009
3003;TRAILER;;3;41;86.0;95.0;1;1.0;3002,3004,3006,3008,3009
3004;TRAILER;;3;43;105.0;114.0;1;1.0;3001,3002,3003,3005,3006,3007,3008,3009
3005;TRAILER;2.2;3;43;115.0;124.0;2;2.0;3001,3004,3007,3008,3009
3006;V;;3;39;86.0;95.0;3;3.0;3002,3003,3004,3008,3009
3007;CAR;;3;45;115.0;124.0;2;2.5;3001,3004,3005,3008,3009
3008;TRAILER;;3;45;105.0;114.0;1;1.0;3001,3002,3003,3004,3005,3006,3007,3009
3009;TRAILER;;3;44;105.0;114.0;1;1.0;3001,3002,3003,3004,3005,3006,3007,3008
4001;TRAILER;;4;74;90.0;99.0;1;1.0;4003,4005,4006,4007,4008
4002;CAR;;4;80;116.0;125.0;2;2.0;4003,4004,4007,4008,4009
4003;TRAILER;;4;79;106.0;115.0;1;1.0;4001,4002,4004,4005,4006,4007,4008,4009
4004;CAR;;4;79;116.0;125.0;2;2.0;4002,4003,4007,4008,4009
4005;V;;4;76;90.0;99.0;3;3.0;4001,4003,4006,4007,4008
4006;TRAILER;;4;75;90.0;99.0;1;1.0;4001,4003,4005,4007,4008
4007;TRAILER;;4;78;106.0;115.0;1;1.0;4001,4002,4003,4004,4006,4008,4009
4008;TRAILER;;4;80;106.0;115.0;1;1.0;4001,4002,4003,4004,4005,4006,4007,4009
4009;TRAILER;;4;78;116.0;125.0;1;1.0;4002,4003,4004,4007,4008
```

The contents when using all parameters results on the next contents:

```
3005;TRAILER;2.2;3;43;115.0;124.0;2;2.0;3001,3004,3007,3008,3009
```

It is important to highlight that there are many combinations of parameters that give the last output. For example, if only parameter *dg_class* with value *2.2* is used, since there is only one unit satisfying this condition.

Finally, before sending back the response, the *API* formats the output in JSON, which has been also verified with both tests as shown in the below pictures:

Figure 5. Output in JSON format (full list of units)

Figure 6. Output in JSON format (filter for dg_class=2.2 used)

7 Conclusions

Main author of the chapter: Francisco Rodero, CIM

Overall integration of systems and tools developed in the context of WP08 with the firefighting control centre is made through the implementation of an interface between the DFC in the FRMC and the SPT, the main reason being that the SPT already concentrates relevant activity of these systems or has been designed to include in short-term without significant impact at coding level.

Focusing on the objectives, the extension of the SPT with this feature allows the software going beyond the limits of the ignition prevention and firefighters can now create more accurate patrol reports including up-to-date information concerning the actual cargo of a ship:

- Cargo scanning and identification system: The Stowage Planning Tool supports the stowage process not only during the pre-loading but also during the loading stage. If an alarm is triggered by the VHD, the SPT keeps track of the alarm for the specific unit and recalculates the risk score accordingly. So, units that have been raised an alarm and have been finally loaded in the ship will also be included in the output of the *FirePatrolReport* feature.
- Automatic screening with rolling drones: AGVs can take advantage of the *FirePatrolReport* feature to get valuable information about cargo distribution for path planning purposes. Since vehicles can request information as many times as needed, patrolling routes can be dynamically adapted to battery charging status, preventive monitoring of certain areas or to control specific units. This way, in case of abnormal temperature detected by their thermal cameras, direct association between the warning and the cargo information can be made.
- Electrical charging monitoring: There is an indirect integration between the FRMC and the system responsible of the continuous monitoring of the charging infrastructure through the SPT. The SW has been designed to manage unique connections for these vehicles that require to be connected during the voyage (reefer units or EV). These connections are selected during the cargo distribution depending on the final placement of the units. This way, output information gives information about which connection is being used by every single unit, which can be send to the FRMC through the DFC, deeply described in D07.6, which features a digital fire plan with visualisations of the spread of heat and smoke. In addition, the historical data about triggered alarms and emergency-related events, like electrical connections, are plotted on a timeline on the display to provide historical data about the emergency, as well as predictions of near future developments.

8 References

Deliverable D07.11 of LASH FIRE, "*Firefighting resource management simulator prototype*", pending to be published in the sLASH FIRE site: https://lashfire.eu

Deliverable D08.4 of LASH FIRE, "*Stowage planning optimization and visualization aid*", pending to be published in the sLASH FIRE site: https://lashfire.eu

9 Indexes

9.1 Index of tables

Table 1.List of requirements 7
Table 2.List of additional considerations about configuration parameters respect to D08.4 8
Table 3.List of required parameters for FirePatrolReport service 8
Table 4. Additional fields of SERVICE_UNITS table 9
Table 5. Folders and files of the software development..... 11
Table 6.Implemented errors/warnings messages..... 11

9.2 Index of figures

Figure 1 - Software components of the Stowage Planning Tool 6
Figure 2. Workflow of FirePatrolReport service..... 10
Figure 3. Output for tests checking for a correct management of the parameters..... 14
Figure 4. Output for tests checking a correct construction of the query..... 15
Figure 5. Output in JSON format (full list of units) 16
Figure 6. Output in JSON format (filter for dg_class=2.2 used) 16