Packet Utilization Services for GNURadio
PUS Tester support software

User Manual

Document: PUS-042104-UM-00200

*RELEASE: **A***

DATE: May 7th, 2024

Author: Gustavo Gonzalez

ARGENTINA

INDEX

1 INTRODUCTION 4

1.1 APPLICABLE AND REFERENCE DOCUMENTS 4

1.2 DEFINITIONS 4

2 PUS TESTER SOFTWARE INSTALLATION 5

3 PUS TESTER SOFTWARE USAGE 7

3.1 SENDING A BINARY REQUEST MESSAGE 11

3.2 SENDING A REQUEST MESSAGE FROM DATABASE 12

3.3 CREATE/EDITING A REQUEST MESSAGE 12

3.4 VIEWING REPORTS MESSAGE 15

3.5 SPECIAL GUIS 16

3.5.1 MEMORY MANAGEMENT SPECIAL GUI 17

3.5.2 TIME-BASED SCHEDULING SPECIAL GUI 19

3.5.3 LARGE PACKET TRANSFER SPECIAL GUI 22

3.5.4 EVENT-ACTION SPECIAL GUI 23

3.5.5 REQUEST SEQUENCING SPECIAL GUI 27

3.6 LOGS 31

3.6.1 GENERAL LOG ¡ERROR! MARCADOR NO DEFINIDO.

3.6.2 SERIAL TX LOG ¡ERROR! MARCADOR NO DEFINIDO.

3.6.3 SERIAL RX LOG ¡ERROR! MARCADOR NO DEFINIDO.

4 PUS TESTER SOFTWARE CONFIGURATION 33

5 PUS TESTER SOFTWARE DATABASE 38

5.1 REQUESTS ¡ERROR! MARCADOR NO DEFINIDO.

5.2 REPORTS TEMPLATES 41

5.3 PARAMETERS 43

CHANGE LOG		
ISSUE	**CHANGE DESCRIPTION**	**DATE**
A	Initial	May 7th, 2024

1 INTRODUCTION

The gr-pus OOT package incorporates the Packet Utilization Services ECSS-E-ST-70-41C [A.D.1] interfaces and services into GNURadio.

Additional ad-hoc (application specific) services could easily be added later as OOT blocks.

The PUS Tester is a software aimed to test the gr-pus functionalities con configuration. The PUS Tester is not intented to be a flight software but only a tester software.

This document is intented to summarize the PUS Tester user manual including its configuration.

1.1 APPLICABLE AND REFERENCE DOCUMENTS

Applicable Documents hereinafter are referred to as AD.X and the reference ones as RD.X.

[A.D.1] ECSS-E-ST-70-41C Telemetry and telecommand packet utilization

[R.D.1] PUS-042104-UM-00100-A gr-pus description and user manual

1.2 DEFINITIONS

According to [A.D.1], next definitions are used:

1. ***Parameters.*** A parameter is a value (often numerical) that represents a small piece of data which can be sent to or received from the satellite. Parameters can represent sensor outputs, configuration values, status indicators, or everything else.
   Parameters are mainly handled by the ParameterService, this these service has the capability to modify parameters values aimed to set mission input variables settings, but is up to the specific mission code to update any other parameter value as physical variables readings (i.e.temperatures)
   Parameters are used for the Housekeeping service to build telemetries frames, they are used for On board monitoring service to keep tracking of the satellite condition, also they are used by the Statistic service to keep tracking of parameters variation through time

2. ***Events***. Events represent expected or unexpected occurences on the spacecraft.

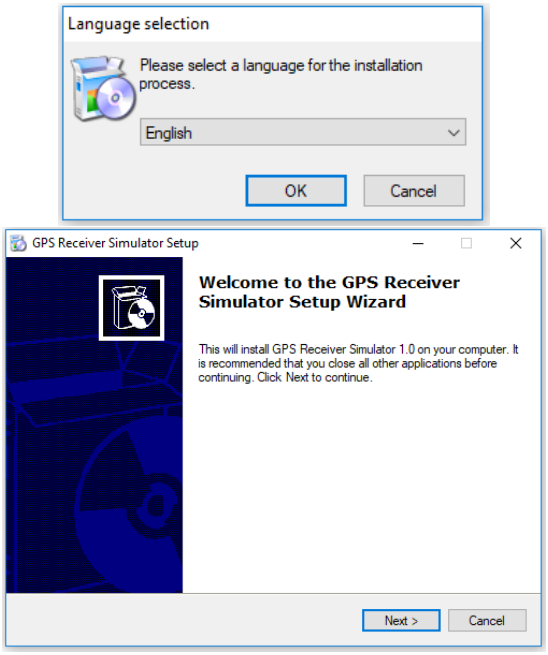
The EventReportService is mainly responsible for management of on-board events reporting while EventActionService execute actions on events detection. Other services provide the capability of generating or responding to on-board events. The events are mainly trigger by the On board monitoring service or any other service with the capability to trigger events

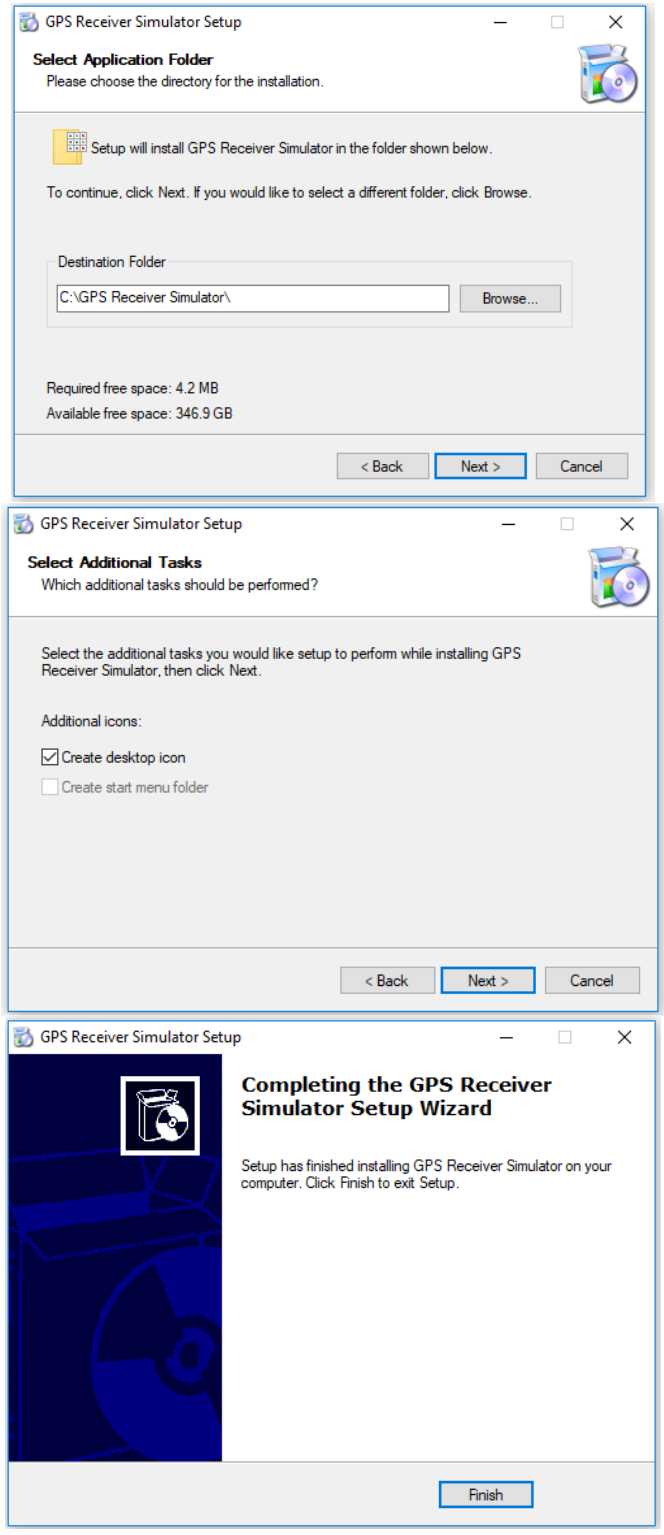
3. ***Application Processes (AP).*** An Application Process is any physical (hardware) or logical (software) entity that can handle PUS messages.
In most cases, an Application Process will be a single microcontroller or subsystem. For example, the OBC, ADCS and Ground Station may be different Application Processes.

2 PUS TESTER SOFTWARE INSTALLATION

The software requires no additional packages to be installed but a serial port available

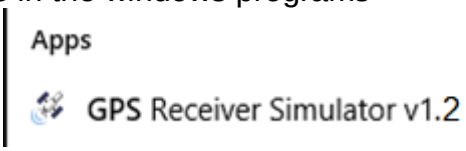
Execute the “PUSTester_setup_1.0.exe” and follow the installation on screen instruction:





After the installation, if the Create Desktop icon has been selected, the next icon will be on the windows desktop

Doble click on the Desktop icon to open the GPS Receiver software or search for the GPS Receiver Simulator v1.3 in the windows programs



3 PUS TESTER SOFTWARE USAGE

Click in the PUS tester icon or execute the PUS tester software from apps to open the software. The main software windows form will be opened:

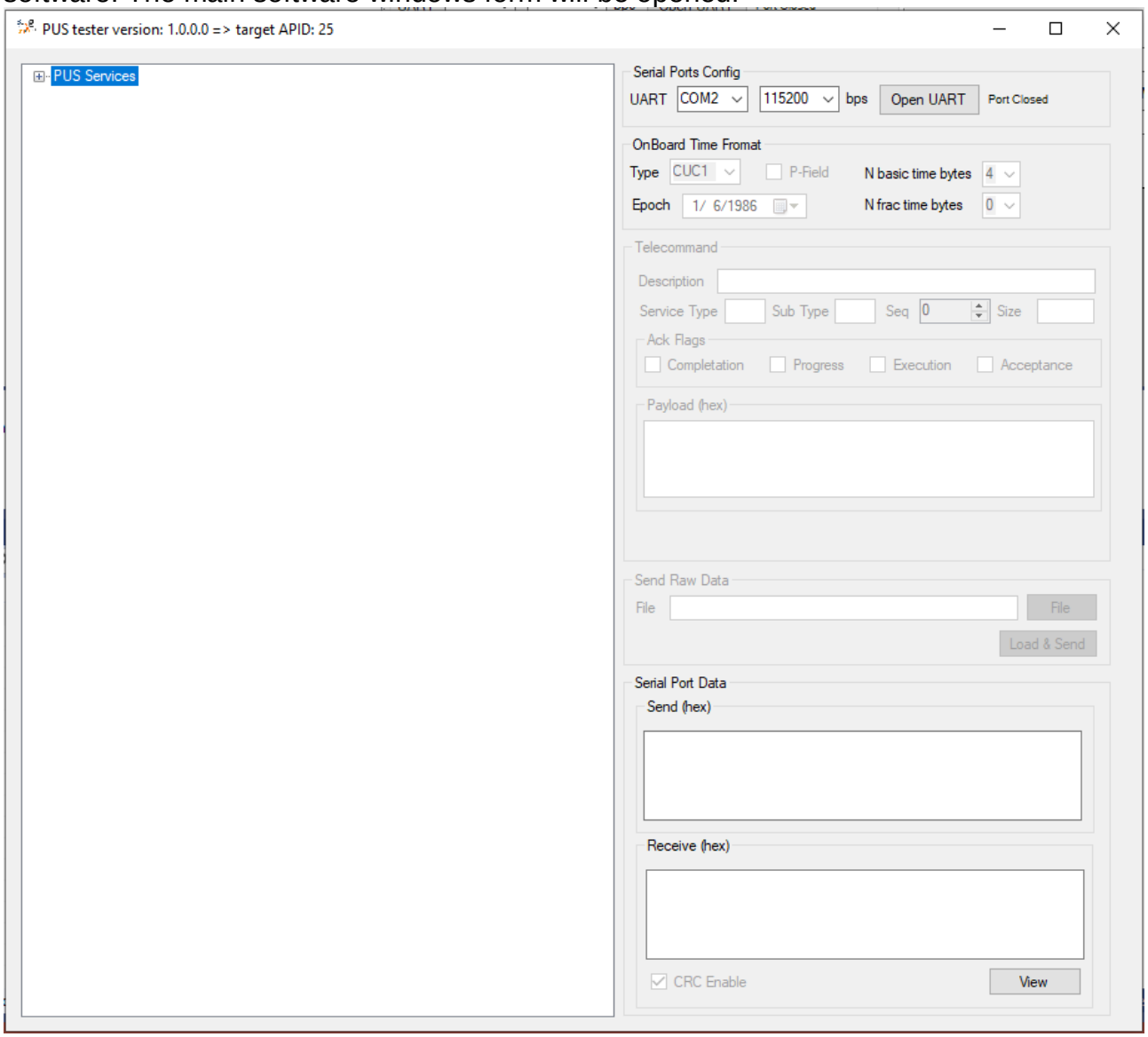
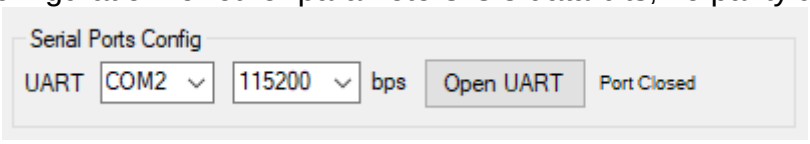


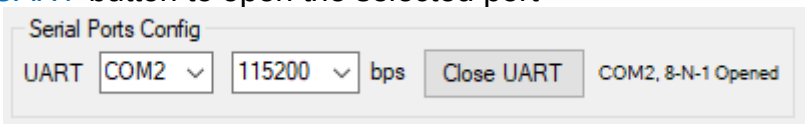
Figure 1 – PUS Tester Main Screen

At the main window top, the software will display the current version and the target APID, this ID will be used in all ***Request*** messages as application process ID, see the software configuration chapter to change this value

In the *Serial Port Config* panel, you can choose your serial port and its baud rate, the default build in configuration for other parameters is 8 data bits, no parity and 1 stop bit



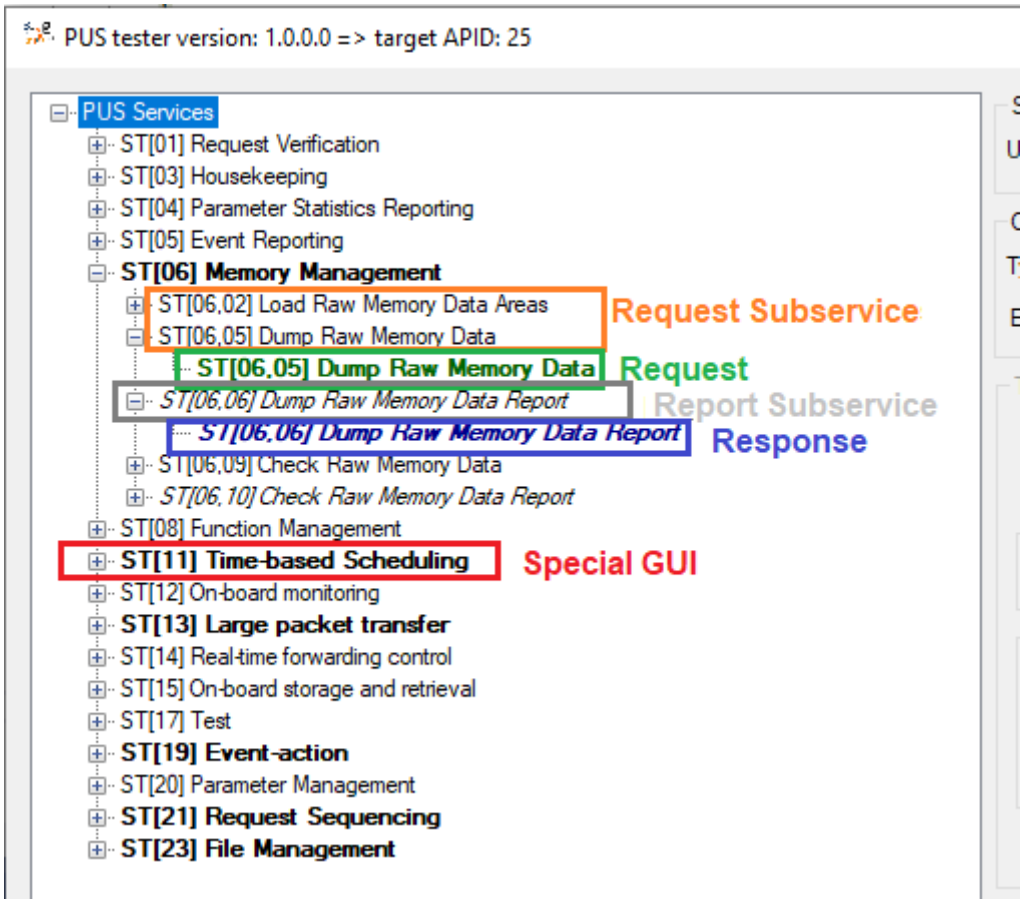
Press the *Open UART* button to open the selected port



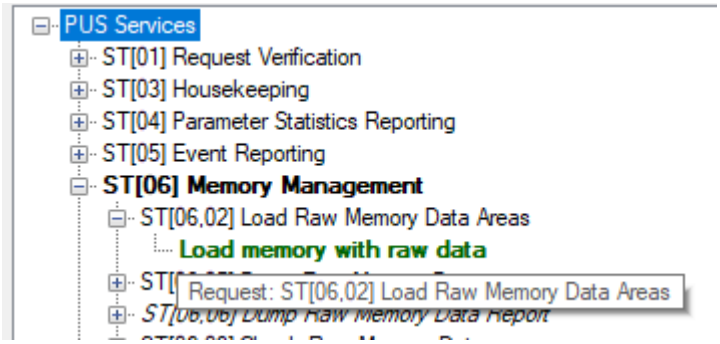
To close the connection press the *Close UART* button

The *Onboard Time Format* panel displays the current Time format expected in the ***Report*** messages. Currently only CUC1/2 with N basic = 4 and N frac = 0 forma is supported (32 bits unsegment seconds). The *Epoch* calendar displays the Epoch date when CUC2 is used. This configuration could be changed from the configuration file, see the software configuration chapter to change this value

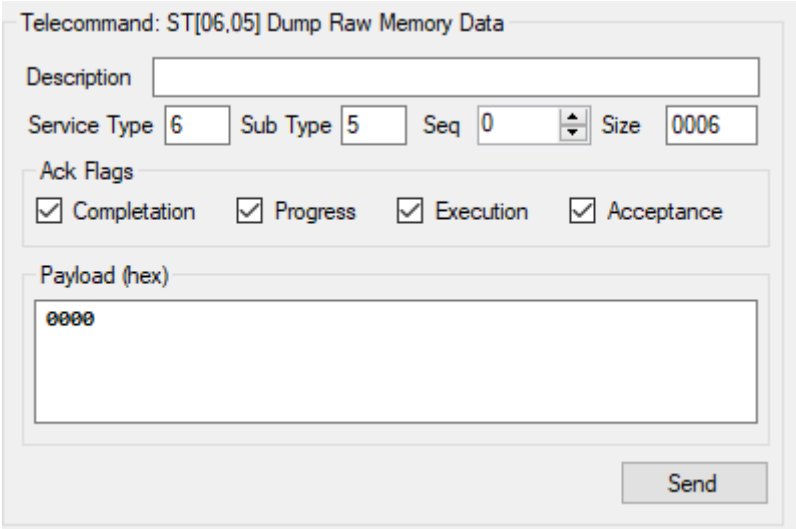
In the left panel there is a tree with all the ***Request*** and ***Report*** messages from PUS services in the data base, press the **+** symbol in each branch to deploy additional tree branches



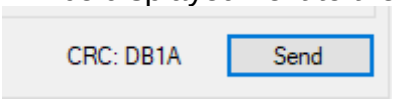
The services with its name in **black bold** font are the ones with special GUIs (see Special GUIs chapter below).
The subservices with its name in *Italic* are for ***Report*** messages, its child node's names are in *blue italic*, which are the ***Report*** messages templates in the data base.
The subservices with its name in normal fonts are the ***Request*** messages, its child node's names are in green which are the ***Request*** messages programmed in the database
Keeping the mouse cursor over a ***Request*** messages will display a tip with the Request subservice information



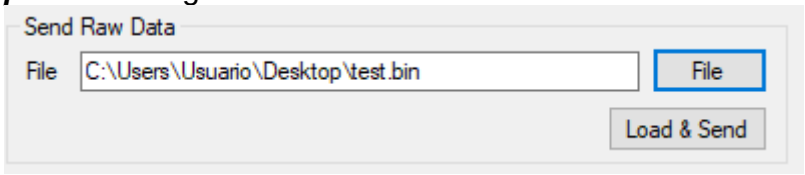
The *Telecommand* panel will remain disable until a ***Request*** messages is selected from the tree. Once a ***Request*** messages is selected, this panle will display the ***Request*** messages relevant information



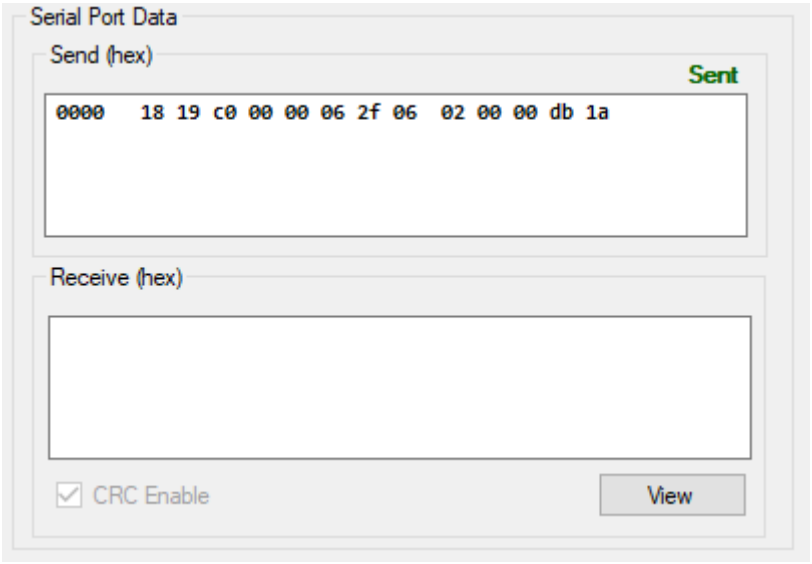
At the panel top, the subservice name will be shown, while the description box will show the ***Request*** message name given by the user if any, the service type and the service subtype will show the ***Request*** message service type and message subtype, while the size box will display the ***Request*** message size. The Seq box shows the sequence Number or ID for the ***Request*** message, this field could be changed by the user before sent the ***Request*** message.
The *Ack Flags* panel allows to set the ***Request*** message acknowledgement flags before send the ***Request*** message
The Payload box will display the ***Request*** message payload data, if any.
To send the ***Request*** message, press the *Send* button. This button will be enabled if a UART port is opened otherwise will remains disable. The PUS Tester will calculate and add the CRC to the message, this CRC will be displayed next to the *Send* button



From the *Send Raw Data* panel you are able to send a binary file through the serial port, this allows you to save binary files with ***Request*** messages and send those files without having these ***Request*** message in the data base.

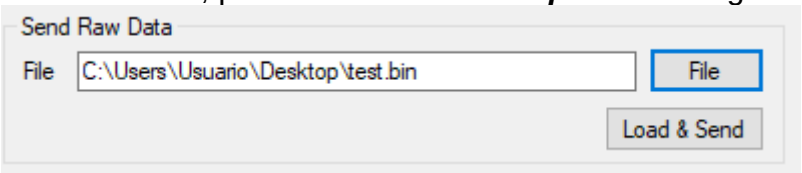


The *Serial Port Data* panel will show the binary data sent and Received throught the UART, the setd data will be update with ***Request*** messages sent while the receive data will display all the ***Report*** messages received. The CRC check box will be enable if the ***Report*** messages have CRC (see Configuration chapter to enable/disable this option). Pressing the *View* button over the received data, the software will decode the ***Report*** messages and display the information in a more friendly forms (see Vieweing Reports messages chapter below)

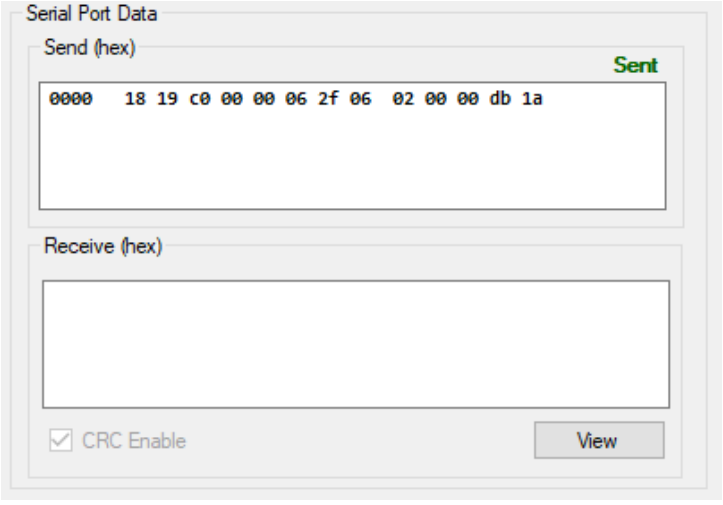


3.1 SENDING A BINARY REQUEST MESSAGE

Request messages could be sent from binary files, this is usefull for large packet transfers such as a software update. This binaries ***Request*** messages files could be create externally or from the Special GUIs some services have (see Special GUIs chapter below)
To send a binary ***Request*** message, press the *File* buton in the *Send Raw Data* panel, navigate and select the binary file with the ***Request*** message. Once a file is selected the *Load & Send* button will enable, press it to send the ***Request*** message.

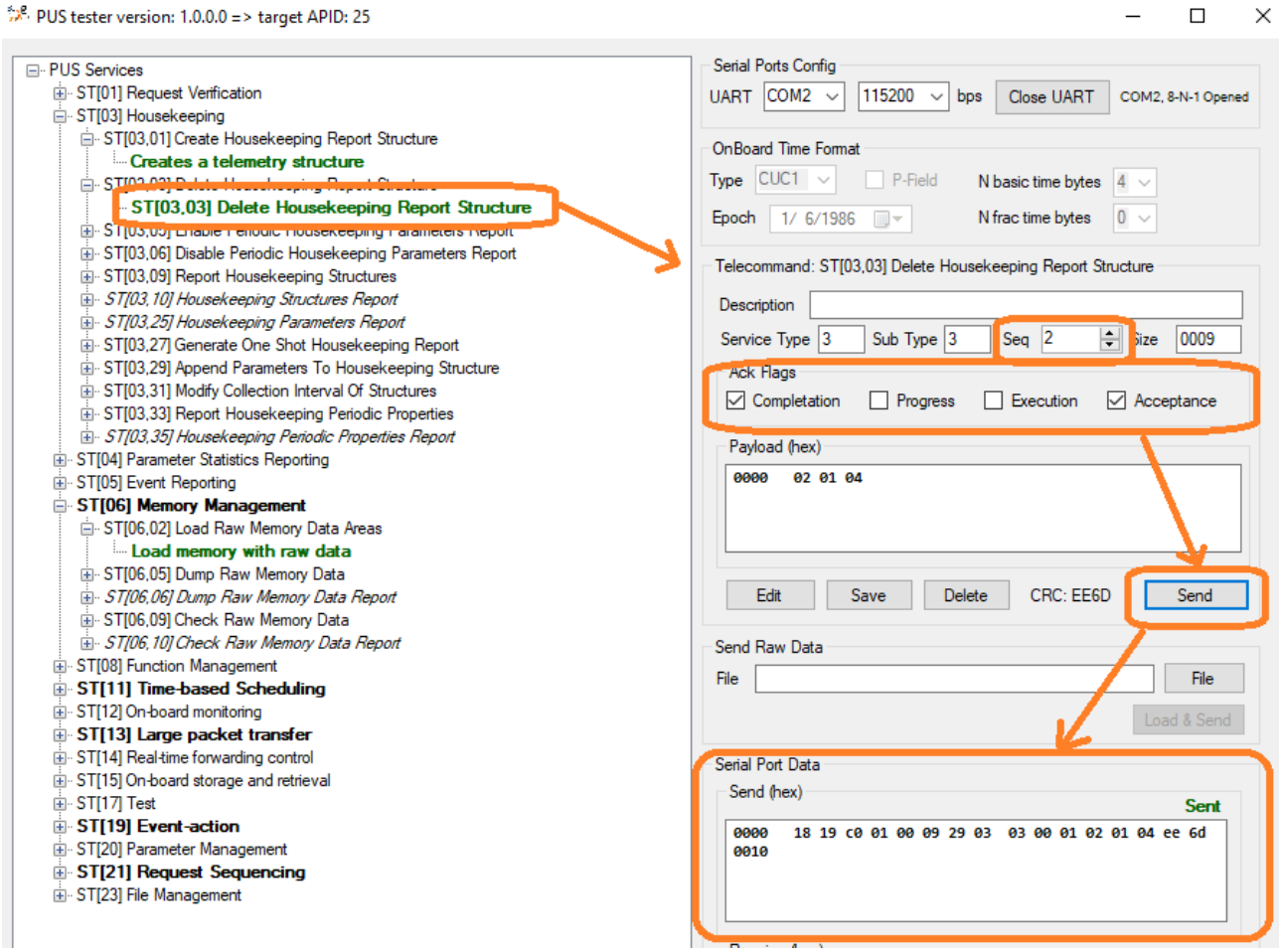


The *Send (hex)* box in the *Serial Port Data* panel will show the binary data sent and a *Sent* message over this box will be displayed
This option sends the binary data as is, this binary data should includes the full ***Request*** message to send, including its CRC
Any ***Report*** message will be displayed in the *Receive (hex)* box in the *Serial Port Data* panel



3.2 SENDING A REQUEST MESSAGE FROM DATABASE

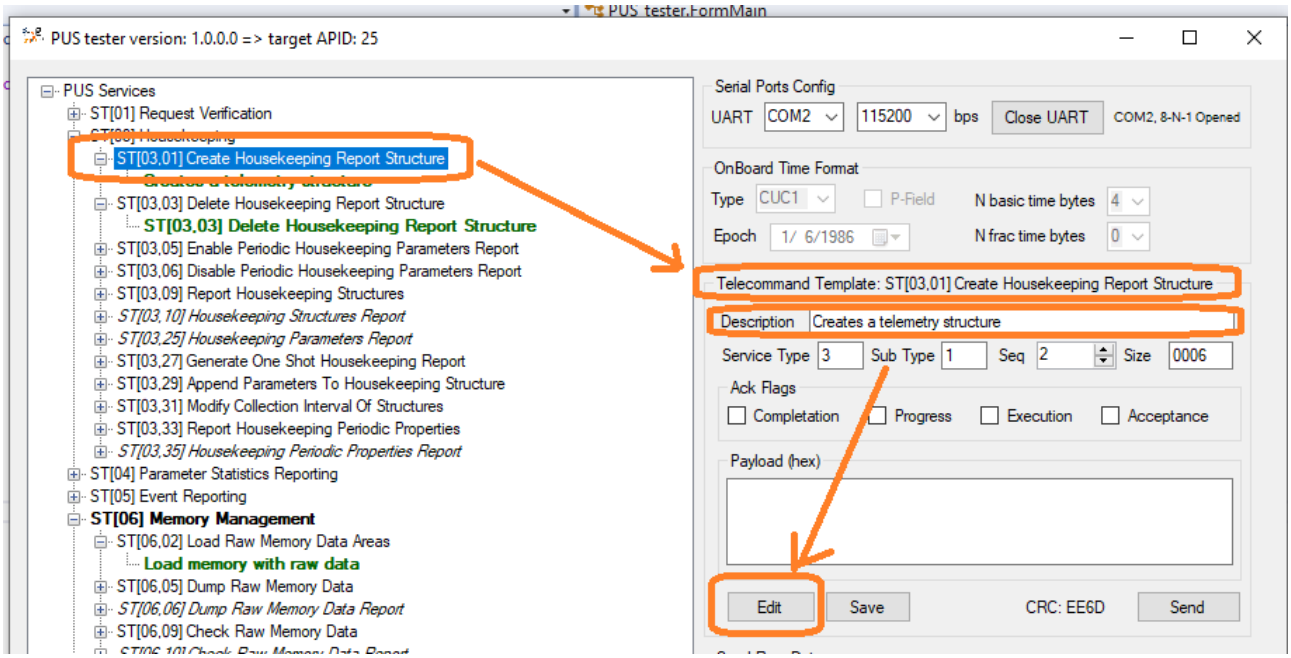
To send a ***Request*** messages from the database select the desire ***Request*** message to send from the *PUS Service* tree



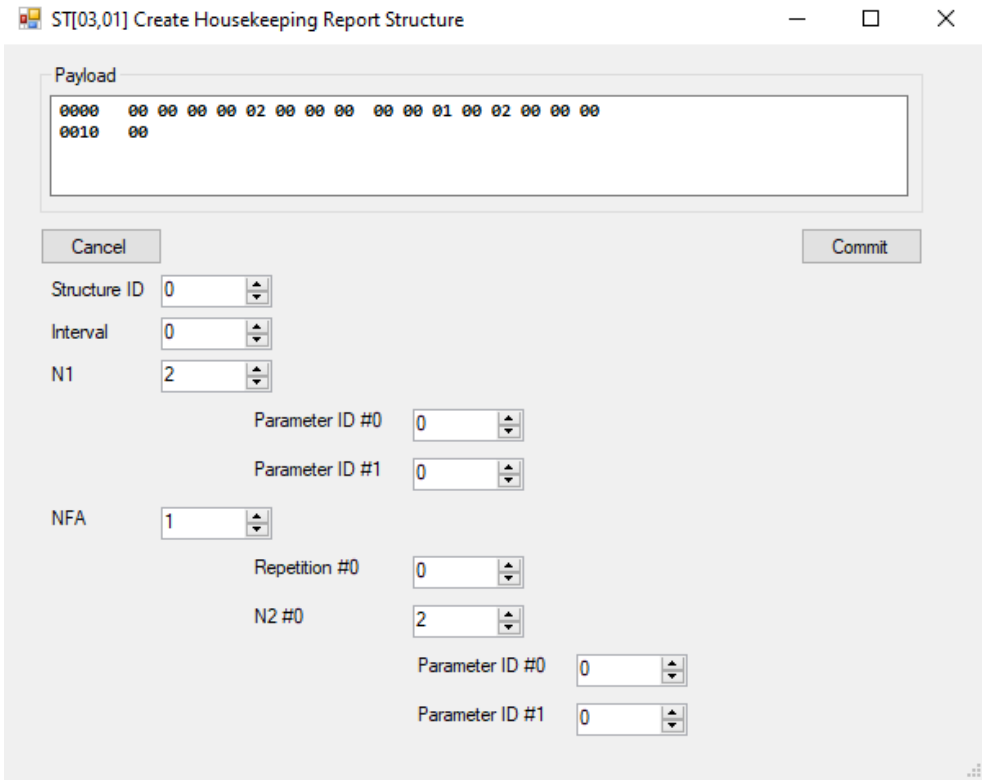
In the *Telecommand* panel set the Seq box with the proper sequence Number or ID for the ***Request*** message and the *Ack Flags* you wish to include, if any.
Then press the *Send* button, remember that this button will be enabled if a UART port is opened otherwise will remains disable. The ***Request*** message, the CRC for the message will be displayed next to the *Send* button. The *Send (hex)* box in the *Serial Port Data* panel will show the binary data sent and a *Sent* message over this box will be displayed
Any ***Report*** message will be displayed in the *Receive (hex)* box in the *Serial Port Data* panel

3.3 CREATE/EDITING A REQUEST MESSAGES

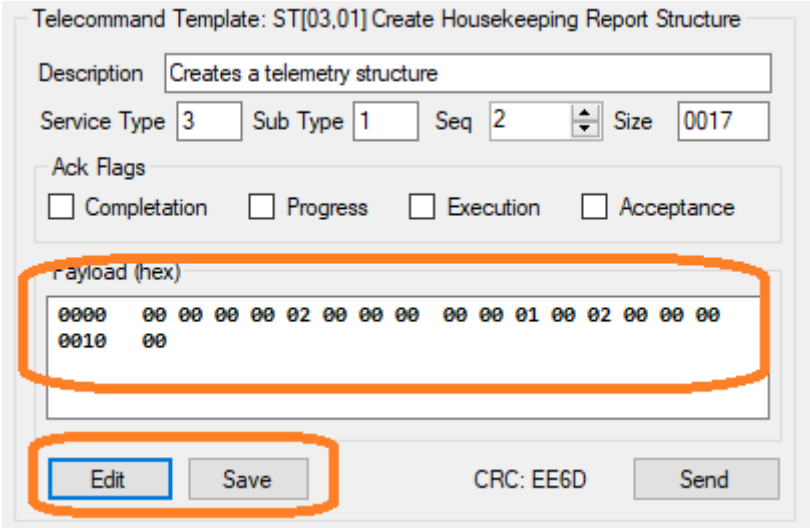
To create and add into the database a new ***Request*** message you could edit the database with a text editor, see Database chapter below, or you could use the PUS tester software, this last method is the recommended one to avoid database corruption or problems
To create a new ***Request*** message, select the subservice branch where you want to create the ***Request*** message from the PUS Service tree



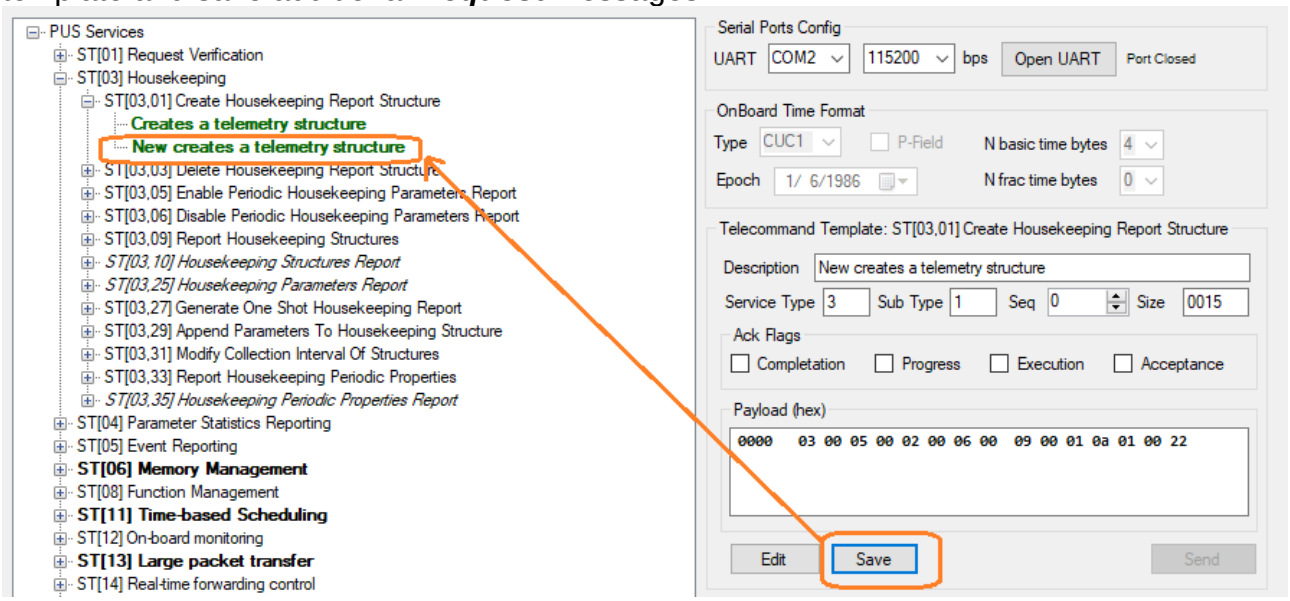
In the *Telecommand* panel, the title will display the Telecommand Template message with the subservice name, in the Description box enter the ***Request*** message name you will use to identify the ***Request*** message in the PUS Service tree and then press the button *Edit*
A new form will open with the ***Request*** message template associate to the subservice. Check [A.D.1] to identify each field name



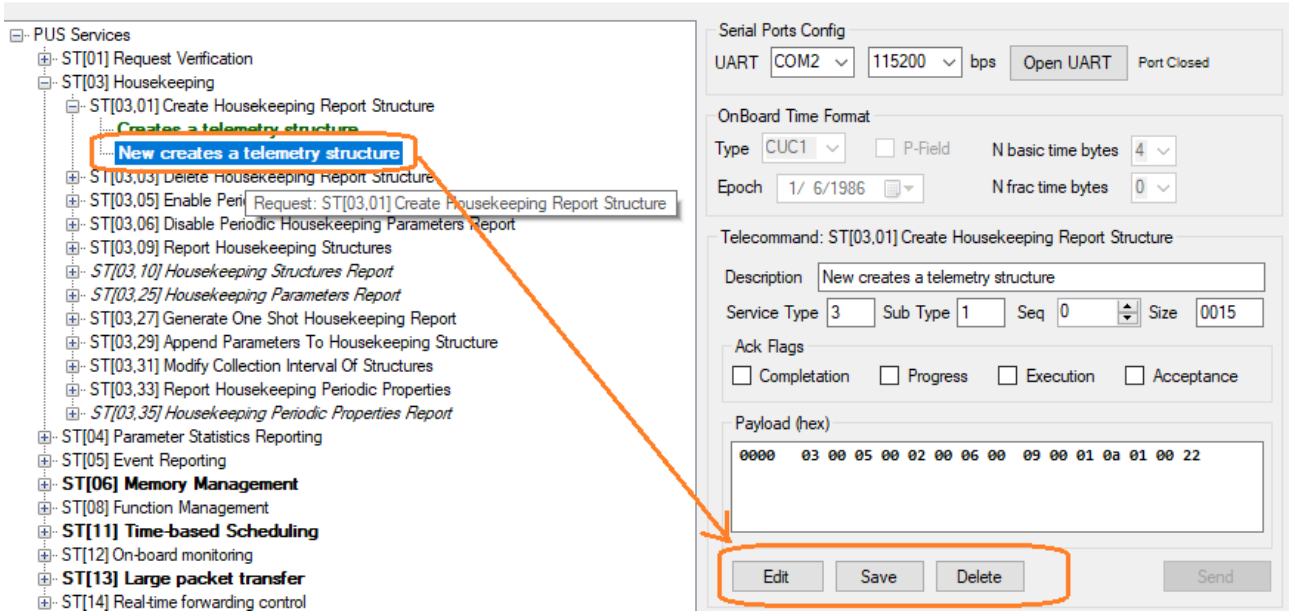
Enter the ***Request*** message's field data you wishfollowing [A.D.1], the *Payload* box will display the payload message raw data. To accept the payload, press *Commit* button, otherwise press *Cancel* button to close the form without accept the changes.
If you press the *Commit* button, the Payload (hex) box in the *Telecommand* panel will be updated. Press the *Edit* button if you wish to change the payload or the *Save* button if you wish to save the ***Request*** message into the database. Press the *Send* button to send the ***Request*** message, you can send the ***Request*** message without saving it into the database



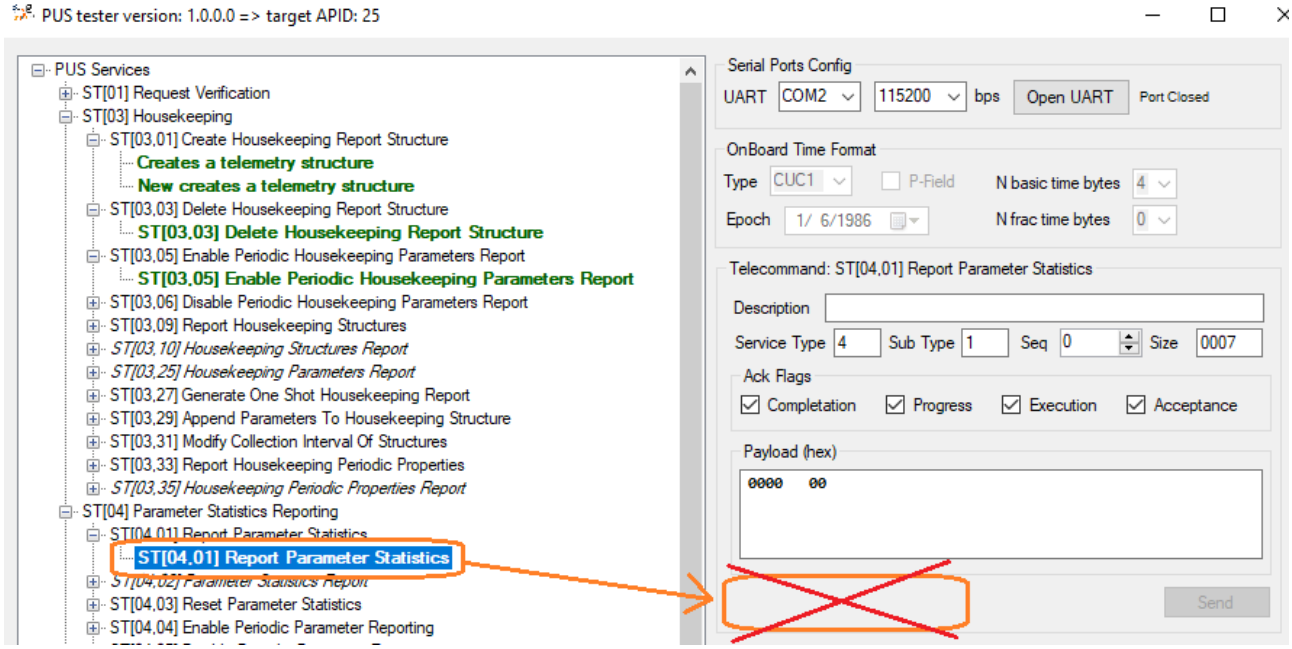
Once the Save button is pressed a new ***Request*** message will appears in the subservice tree branch. Remember the selected PUS Service will remains the template until the new ***Request*** message is selected from the PUS Service tree, you will able to modify the template and save additional ***Request*** messages



Selecting the new ***Request*** message you will be able to modify it pressing the *Edit* button, save the modification (update) pressing the *Save* button or delete it from the database pressing the *Delete* button

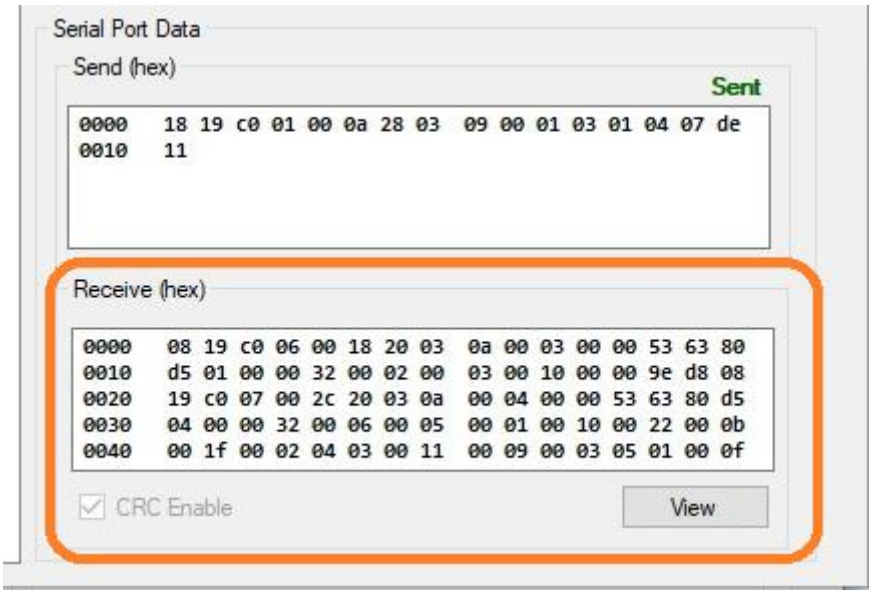


Some ***Request*** messages are write protected, then selecting it will no display the *Edit*, *Save* and *Delete* buttons, this can be done editing the database from a text editor, see the Database chapter

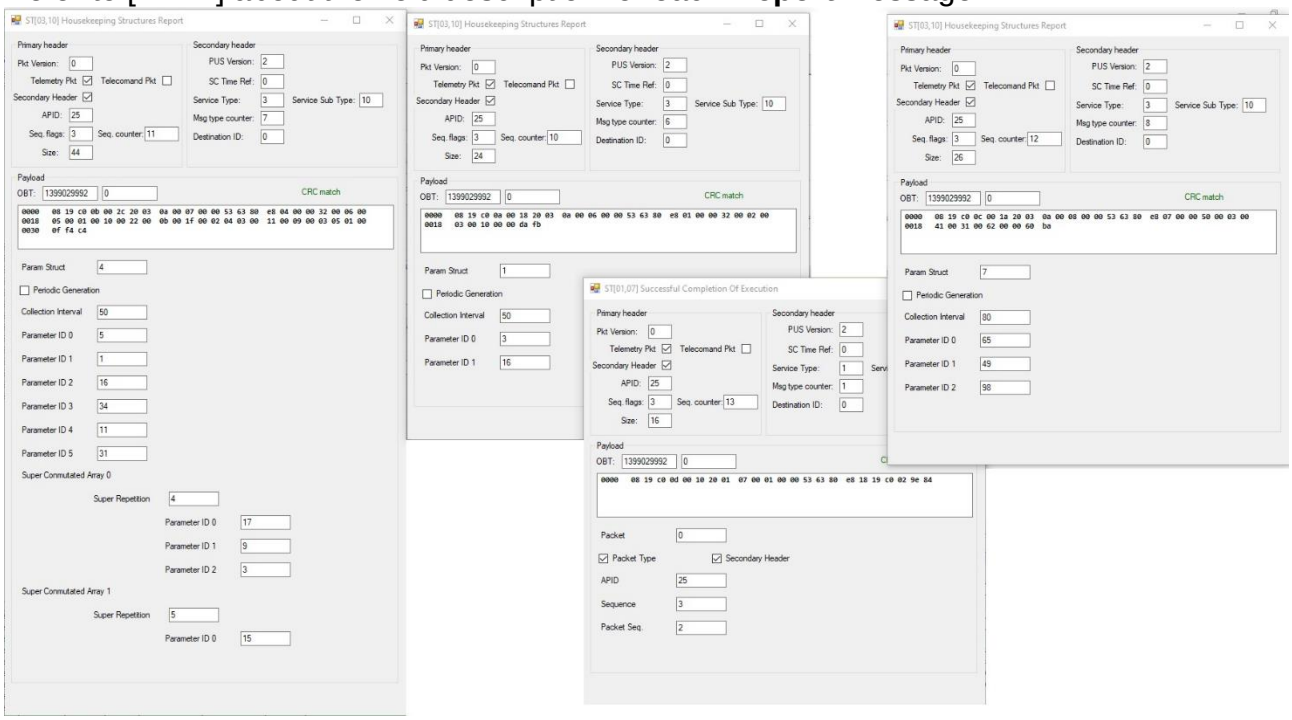


3.4 VIEWING REPORT MESSAGES

Received **Report** messages data is displayed in the *Serial Port Data* panel, *Receive (hex)* box, this data is the binary received data either as response to a **Request** message or as periodic **Report** message.
To display the information in a more user's friendly way, press the **View** button



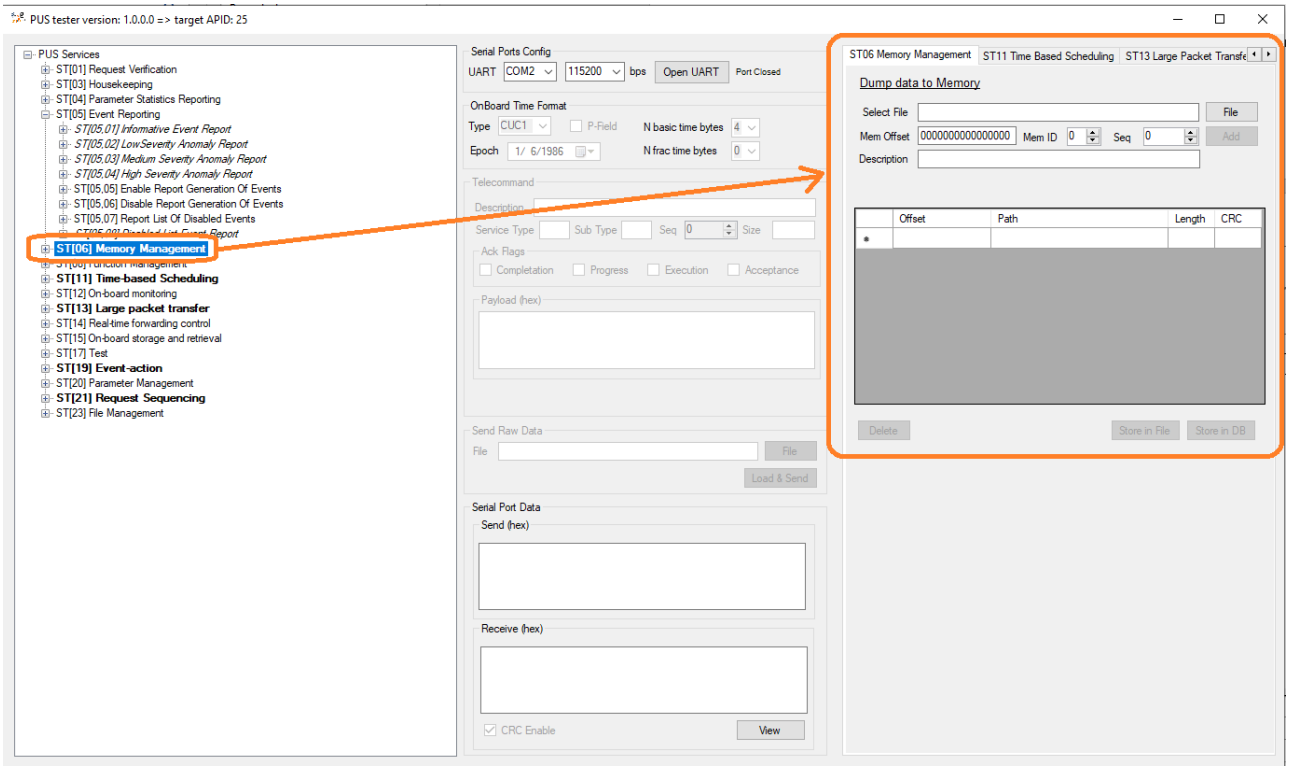
The **View** button will open additional windows forms, one per each **Report** message showing the information according to each **Report** message template (see chapter Reports template below)
Refer to [A.D.1] about the field description for each **Report** message



3.5 SPECIAL GUIS

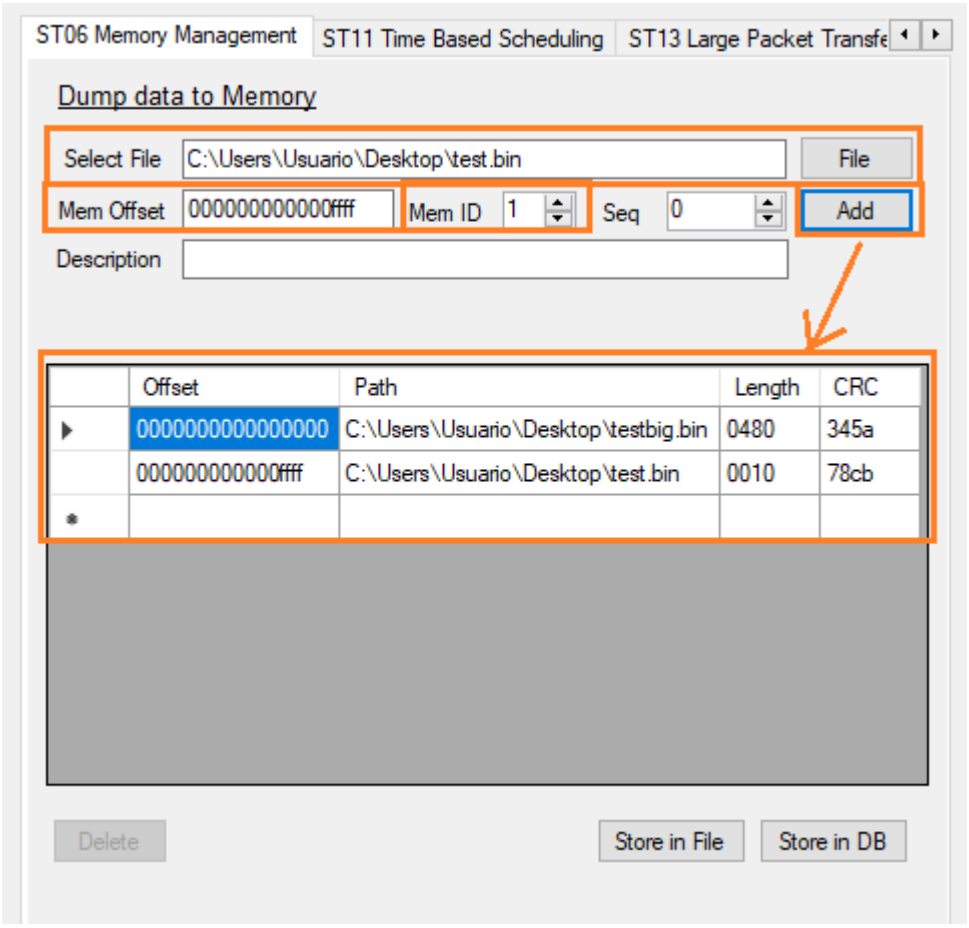
Some PUS Services have special GUIs to help user with complex **Request** Messages creation. These GUIs offers a simpler interface than create **Request** Messages filling each field by hand.

The services with special GUIs are displayed in the PUS services tree with **black bold** font. To display the special GUI for a service, select this service from the PUS service tree, the special GUI will be displayed at the right side of the main windows

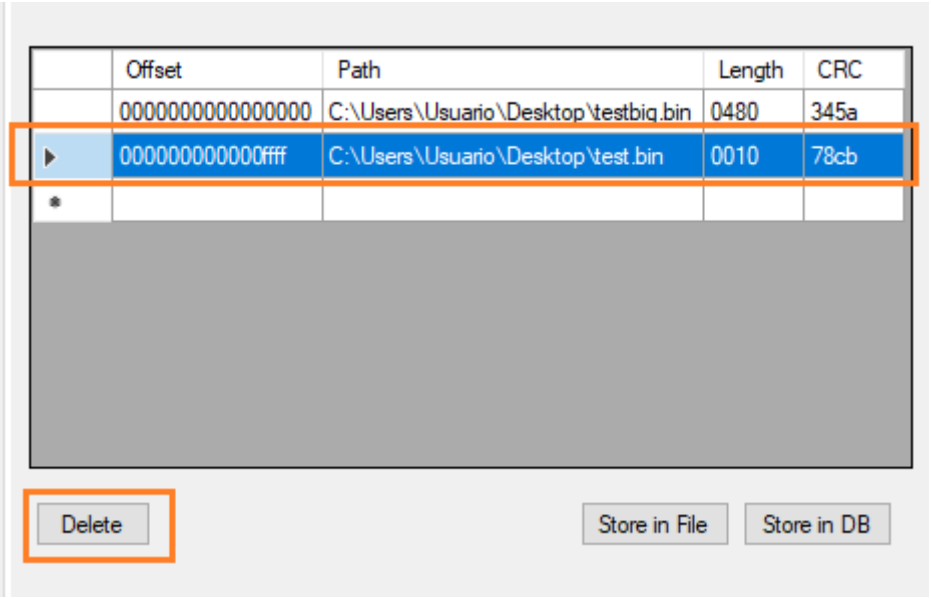


3.5.1 MEMORY MANAGEMENT SPECIAL GUI

The ST[06] Memory Management service has a special GUI allowing the user to create a **Request** Message to load memory information.
The user is able to create **Request** Messages to load binary files into specific memory locations.
Press the *File* button then navigate and select the binary file to load in memory. Select the Memory ID from *Mem ID* updown selector where upload this file binary data, then the memory position from *Mem. Offset* box and press add to load this configuration into the ***Request*** message table. You could add additional entries into the table

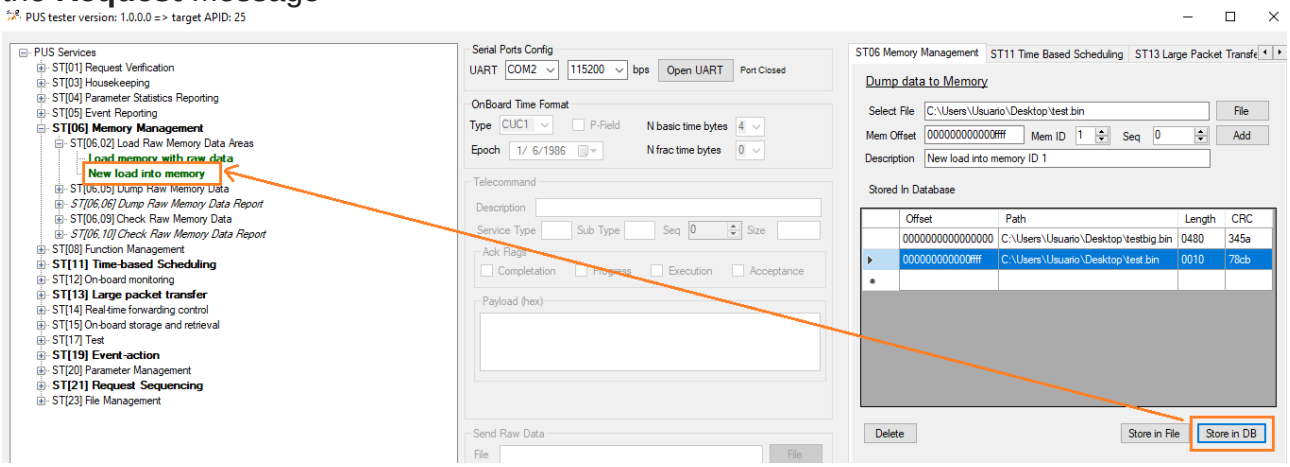


To delete an entry from the table, select the row you wish to delete and press the *Delete* button.

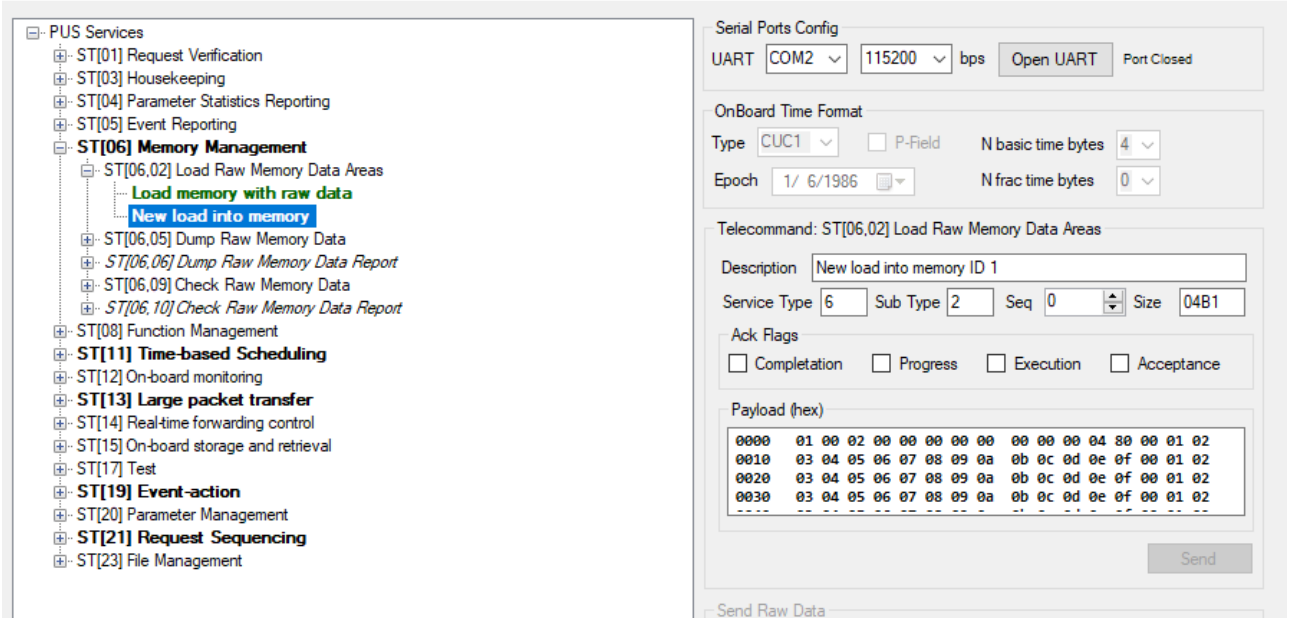


Once the table is filled with all the data you wish to load into memory, select the **Request** Message Sequence ID from *Seq* box if you want to save the **Request** Message into a file and change it and fill the *Description* box with the new **Request** Message name, otherwise the default subservice name will be used

To create a new **Request** Message in the database under ST[06,02], press the *Store in DB* button, alternatively you are able to create a binary file with the **Request** Message pressing the *Store in File* button, this is usefull either to sent the **Request** Message as binary request without saving it into the database or if the **Request** Message is large enough to required the use of the Large Packet transfer service, which could use this binary file as input to split the **Request** Message



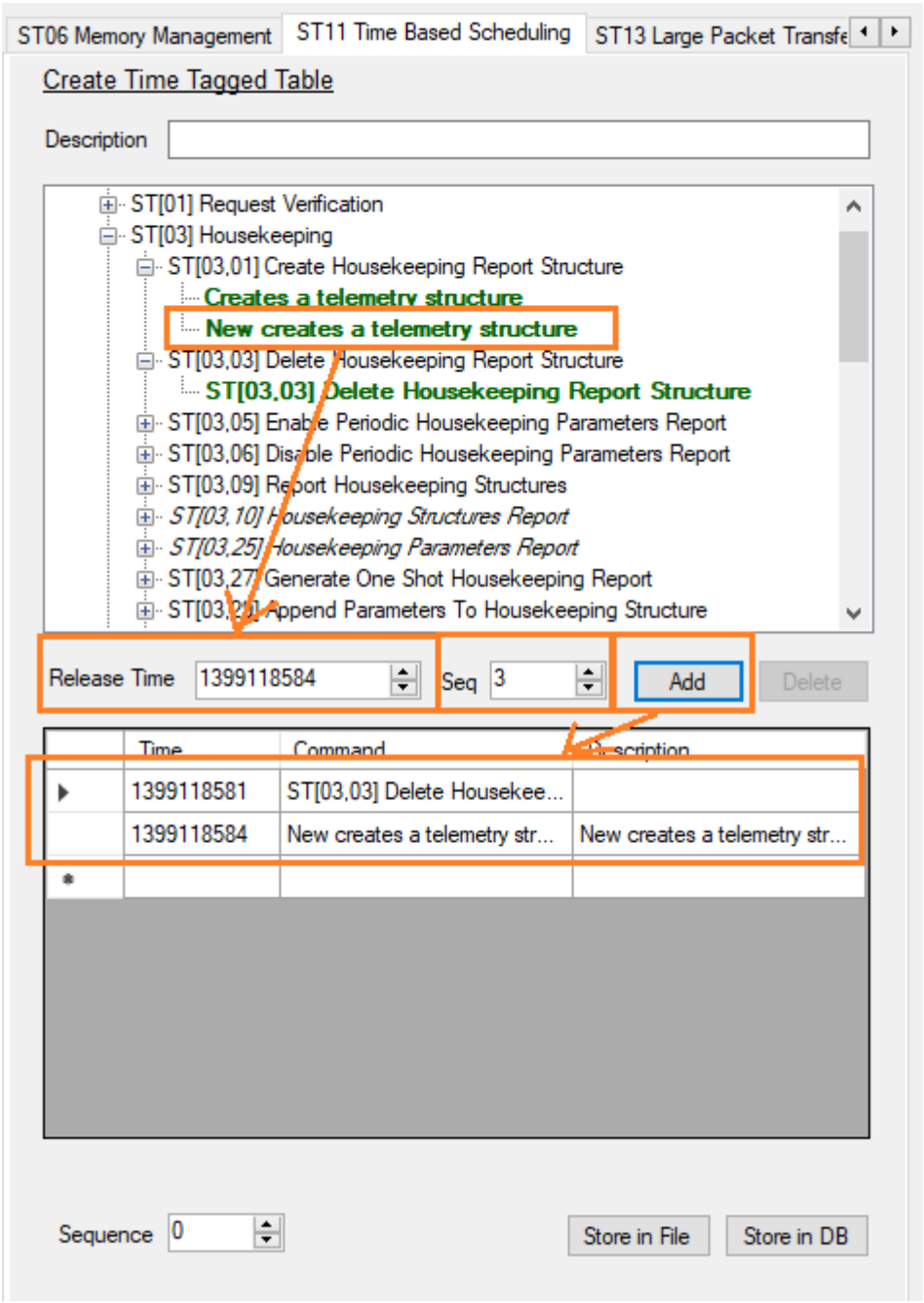
Once the **Request** Message is store into the database it could be use, see the Sending a Request Message from Database chapter



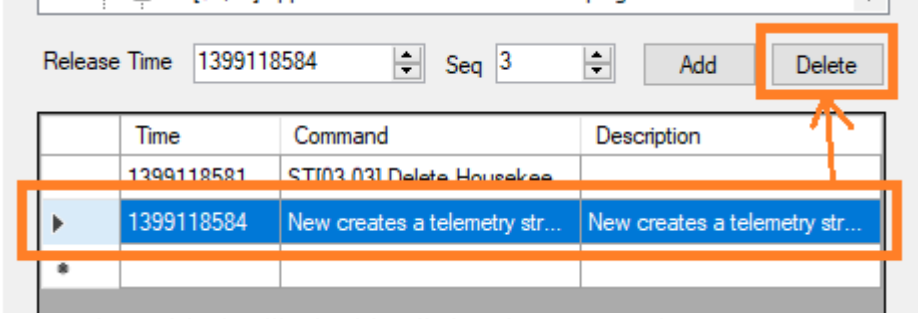
3.5.2 TIME-BASED SCHEDULING SPECIAL GUI

The ST[11]Time-based Scheduling service has a special GUI allowing the user to create a **Request** Message to upload time tagged **Request** Messages.
A PUS service's **Request** Messages database tree is displayed, select the **Request** Message you wish to add into the time tagged table from the tree, then select the release time from *Release Time* box, the sequence ID identification for this **Request** Message from *Seq* box and press the Add button to add it into the table

.



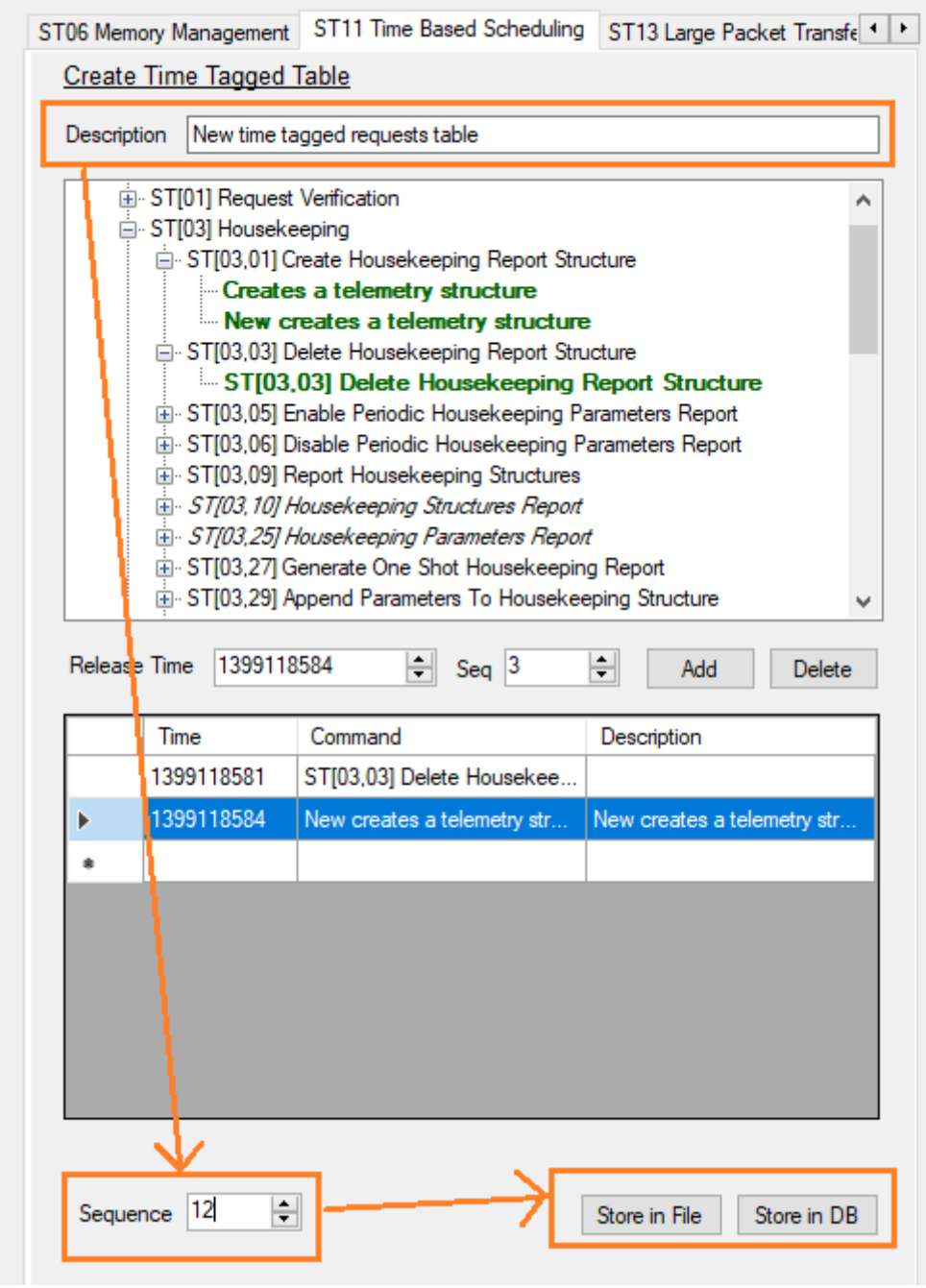
To delete a table entry select the row to delete and rpess the *delete* button



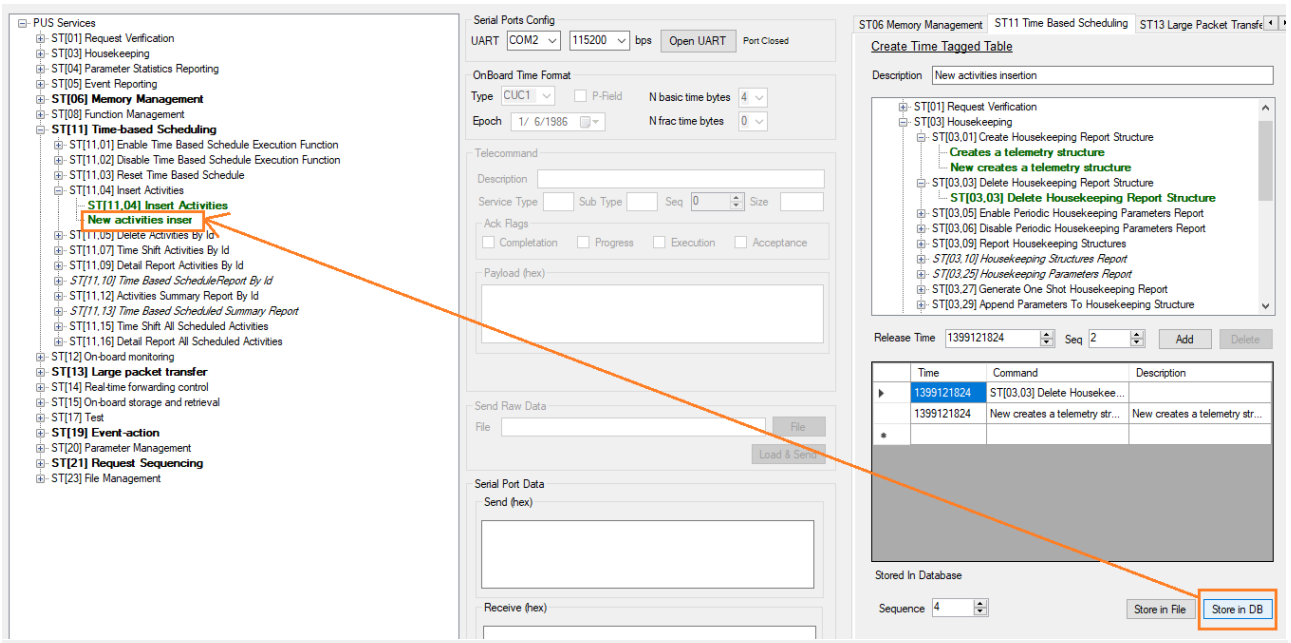
Once the table is filled with all the time tagged **Request** messages you wish to insert into activities, select the **Request** Message Sequence ID from *Sequence* box if you want to save

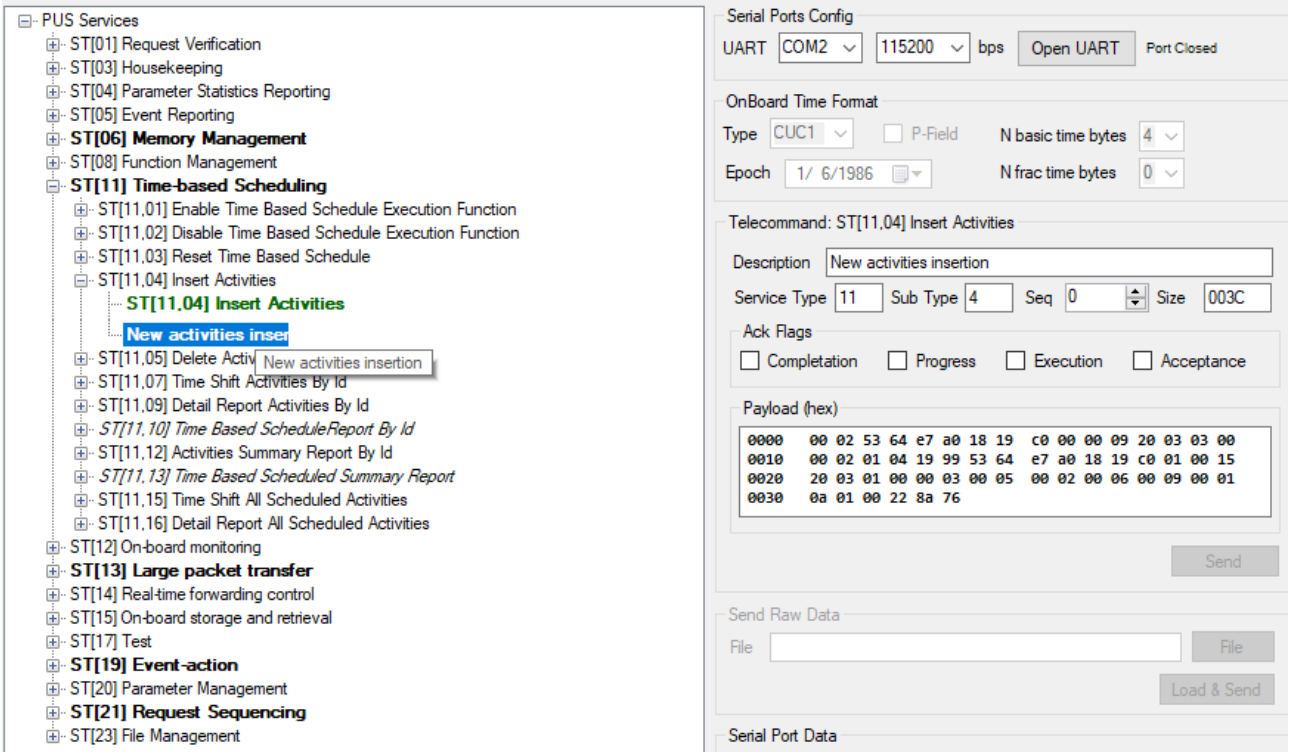
the **Request** Message into a file and change it and fill the *Description* box with the new **Request** Message name, otherwise the default subservice name will be used

To create a new **Request** Message in the database under ST[11,04], press the *Store in DB* button, alternatively you are able to create a binary file with the **Request** Message pressing the *Store in File* button, this is usefull either to sent the **Request** Message as binary request without saving it into the database or if the **Request** Message is large enough to required the use of the Large Packet transfer service, which could use this binary file as input to split the **Request** Message



Once the **Request** Message is store into the database it could be use, see the Sending a Request Message from Database chapter

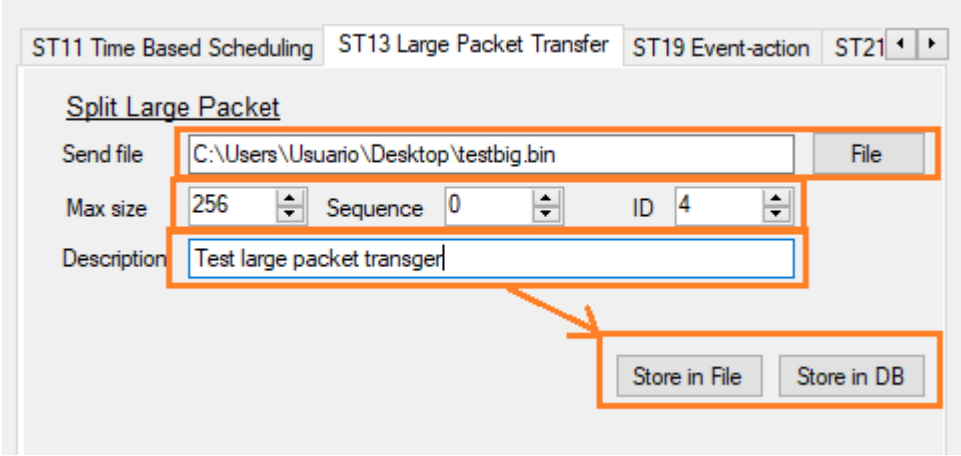




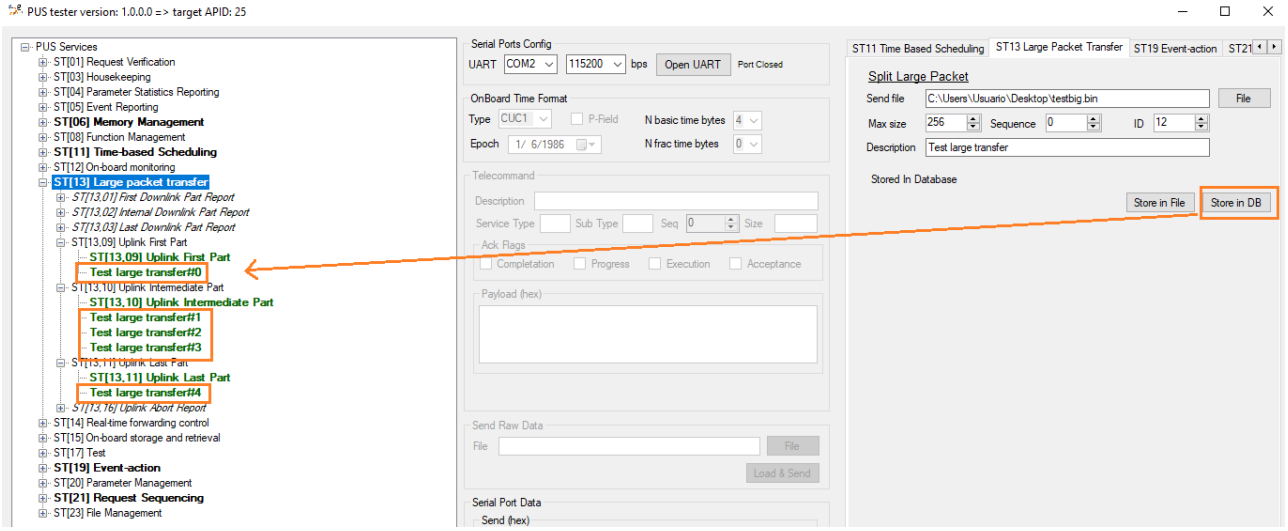
3.5.3 LARGE PACKET TRANSFER SPECIAL GUI

The ST[13] Large Packet Transfer service has a special GUI allowing the user to create a set of **Request** Messages to send large **Request** message packets as small ones.
The user is able to create **Request** Messages to load binary files into specific memory locations.

Press the *File* button then navigate and select the binary file to split into small packets. Select the maximum packet size from *Max size* updown selector, select the transfer ID from the ID box, then select the **Request** Message Sequence ID from *Sequence* box if you want to save the **Request** Message into a file, this sequence ID will applied to the first **Request** message packet, sucesive **Request** messages will incremente by 1 each new **Request** message create with the large information to upload. Fill the *Description* box with the new **Request** Message name, all packet will share the same name with an incremental number added, otherwise the default subservice name will be used



To create a new **Request** Message in the database under ST[13,09/10/11], press the *Store in DB* button, alternatively you are able to create a binary file with the **Request** Messages pressing the *Store in File* button.



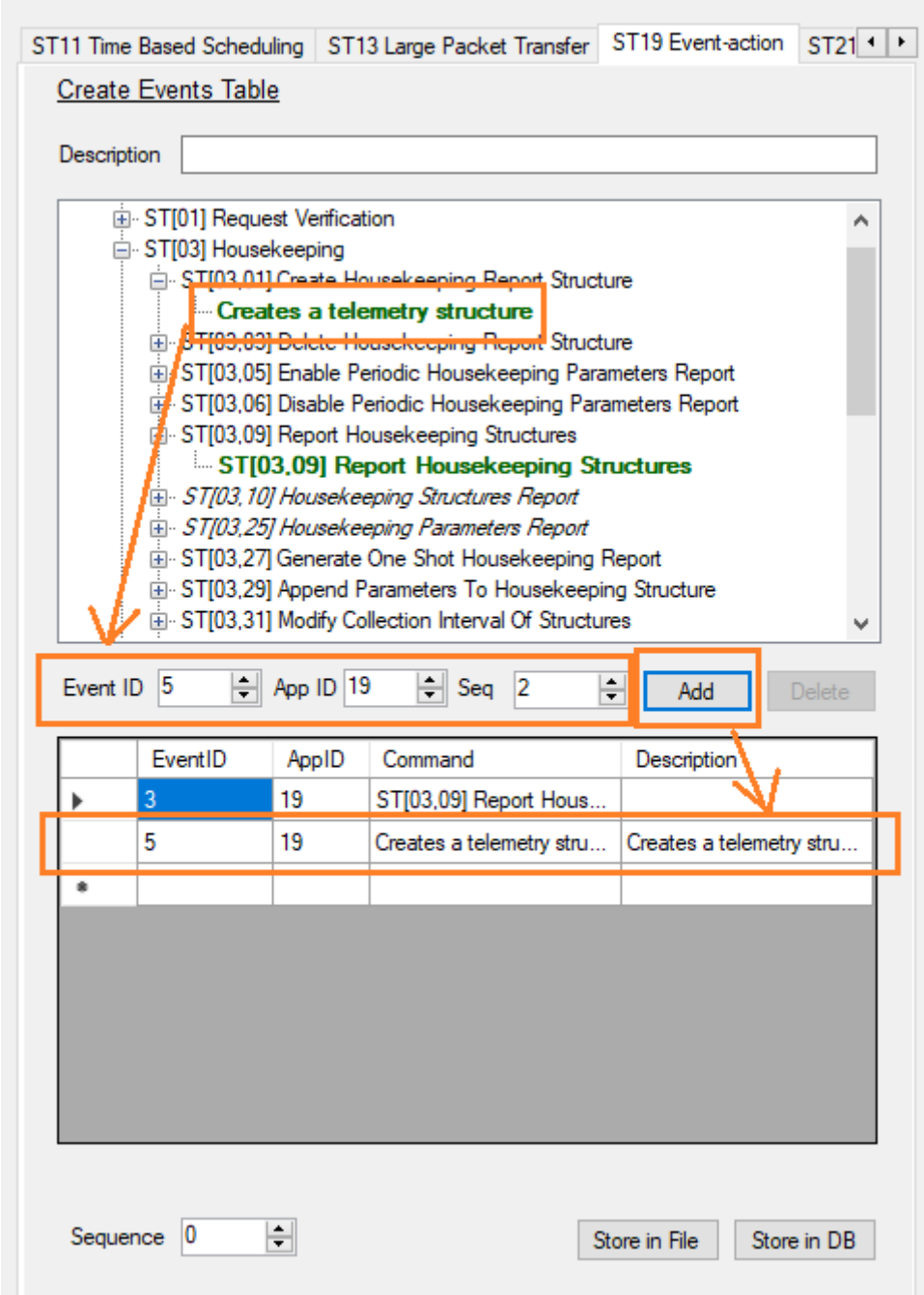
Once the **Request** Messages are store into the database they could be use, see the Sending a Request Message from Database chapter

3.5.4 EVENT-ACTION SPECIAL GUI

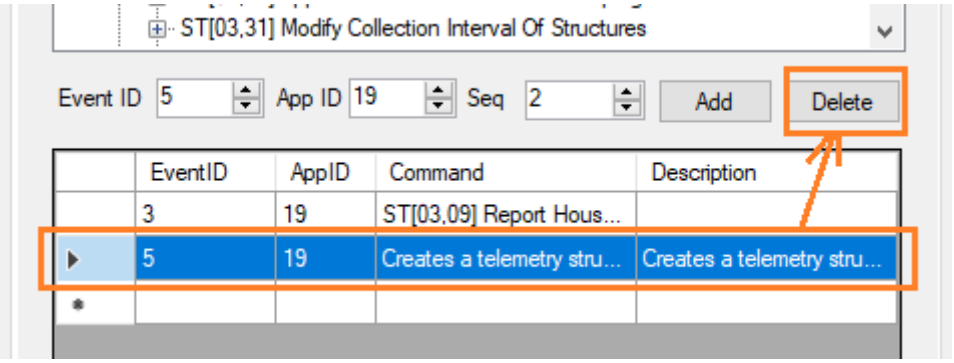
The ST[19] Event-action service has a special GUI allowing the user to create a **Request** Message to upload new **Request** Messages actions.

A PUS service's **Request** Messages database tree is displayed, select the **Request** Message you wish to add into the sequence table from the tree, then select the Event associated to the Request message selected in the *Event ID* updown selector, the Application ID for the Event in the *App ID* updown selector and the sequence ID identification for this **Request** Message from *Sequence* box and press the *Add* button to add it into the table

.

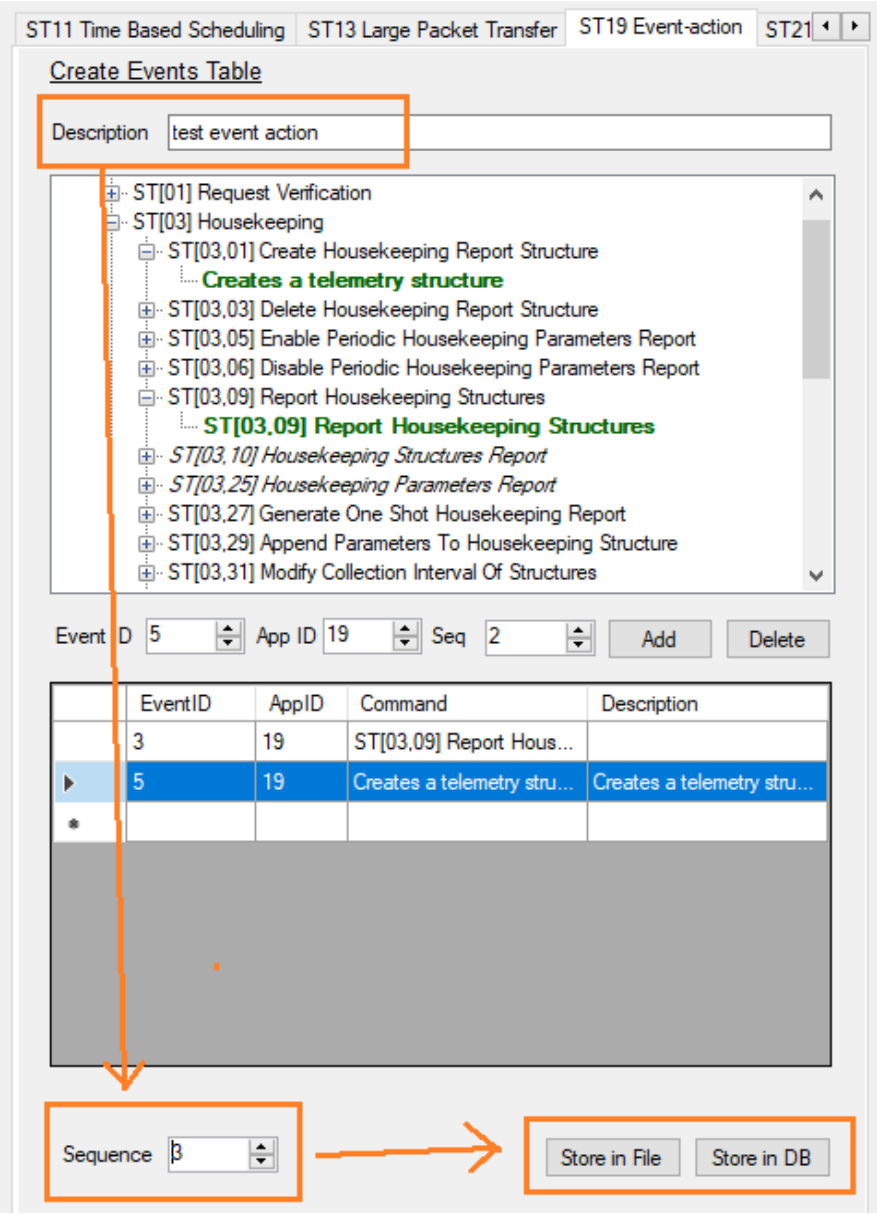


To delete a table entry select the row to delete and rpess the *delete* button

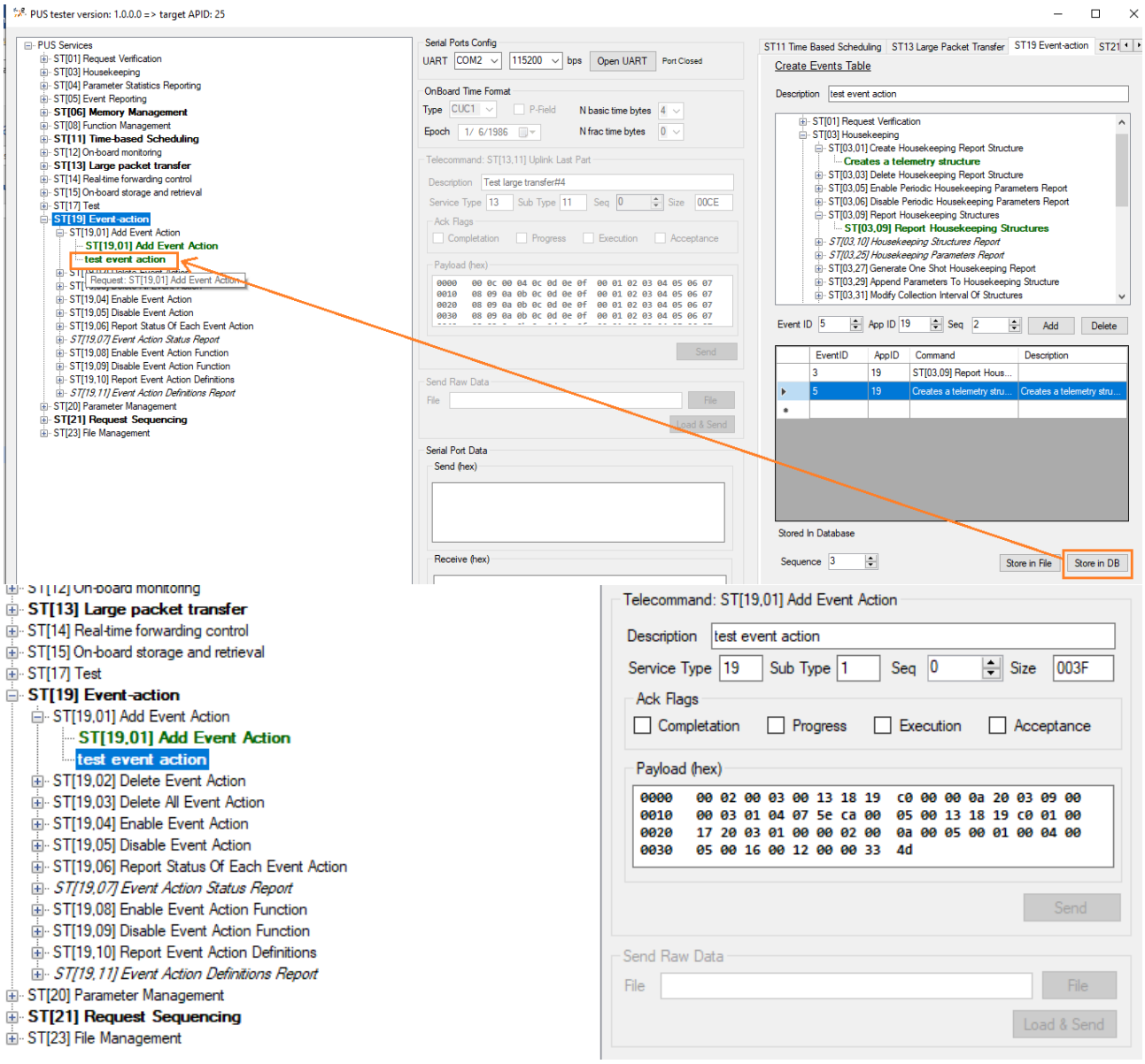


Once the table is filled with all the events **Request** messages you wish to insert into the events table, select the **Request** Message Sequence ID from *Sequence* box if you want to save the **Request** Message into a file and change it and fill the *Description* box with the new **Request** Message name, otherwise the default subservice name will be used

To create a new **Request** Message in the database under ST[19,01], press the *Store in DB* button, alternatively you are able to create a binary file with the **Request** Message pressing the *Store in File* button, this is usefull either to sent the **Request** Message as binary request without saving it into the database or if the **Request** Message is large enough to required the use of the Large Packet transfer service, which could use this binary file as input to split the **Request** Message



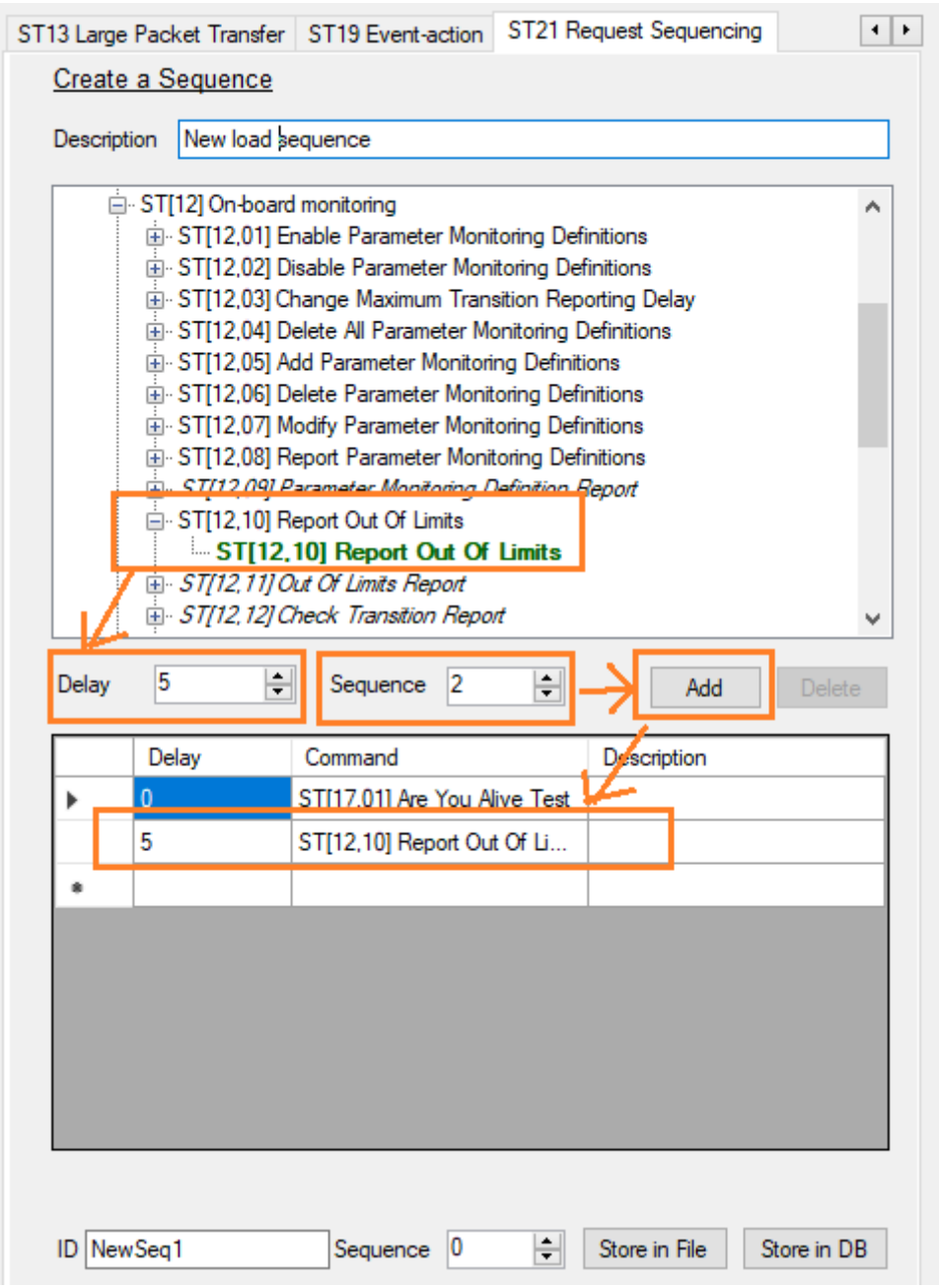
Once the **Request** Message is store into the database it could be use, see the Sending a Request Message from Database chapter



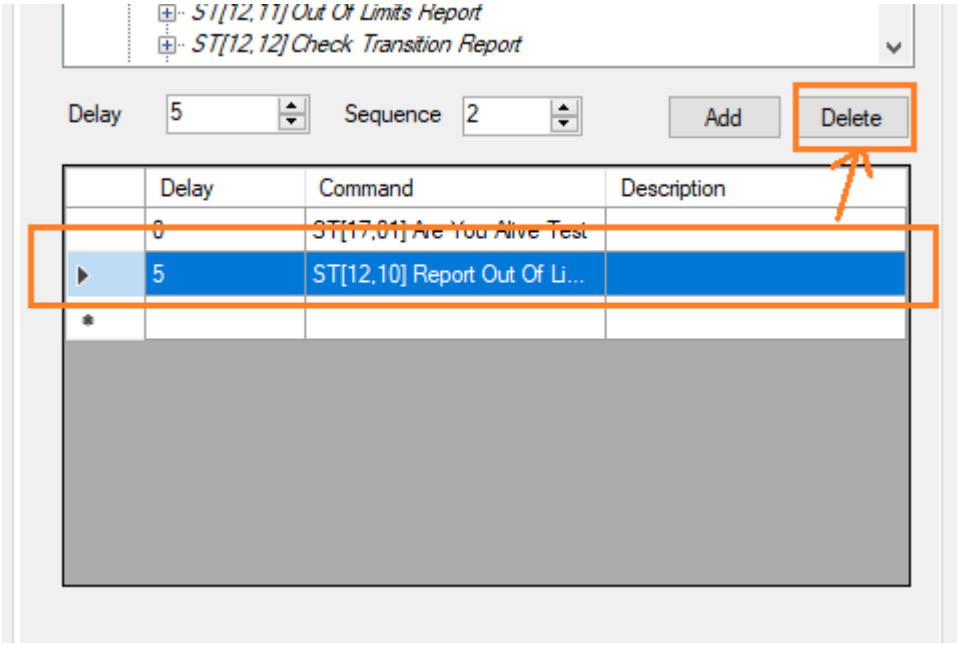
3.5.5 REQUEST SEQUENCING SPECIAL GUI

The ST[21] Request Sequencing service has a special GUI allowing the user to create a **Request** Message to upload new **Request** Messages sequences.
A PUS service's **Request** Messages database tree is displayed, select the **Request** Message you wish to add into the sequence table from the tree, then select the delay between releases from *Delay* box, the sequence ID identification for this **Request** Message from *Sequence* box and press the Add button to add it into the table

.

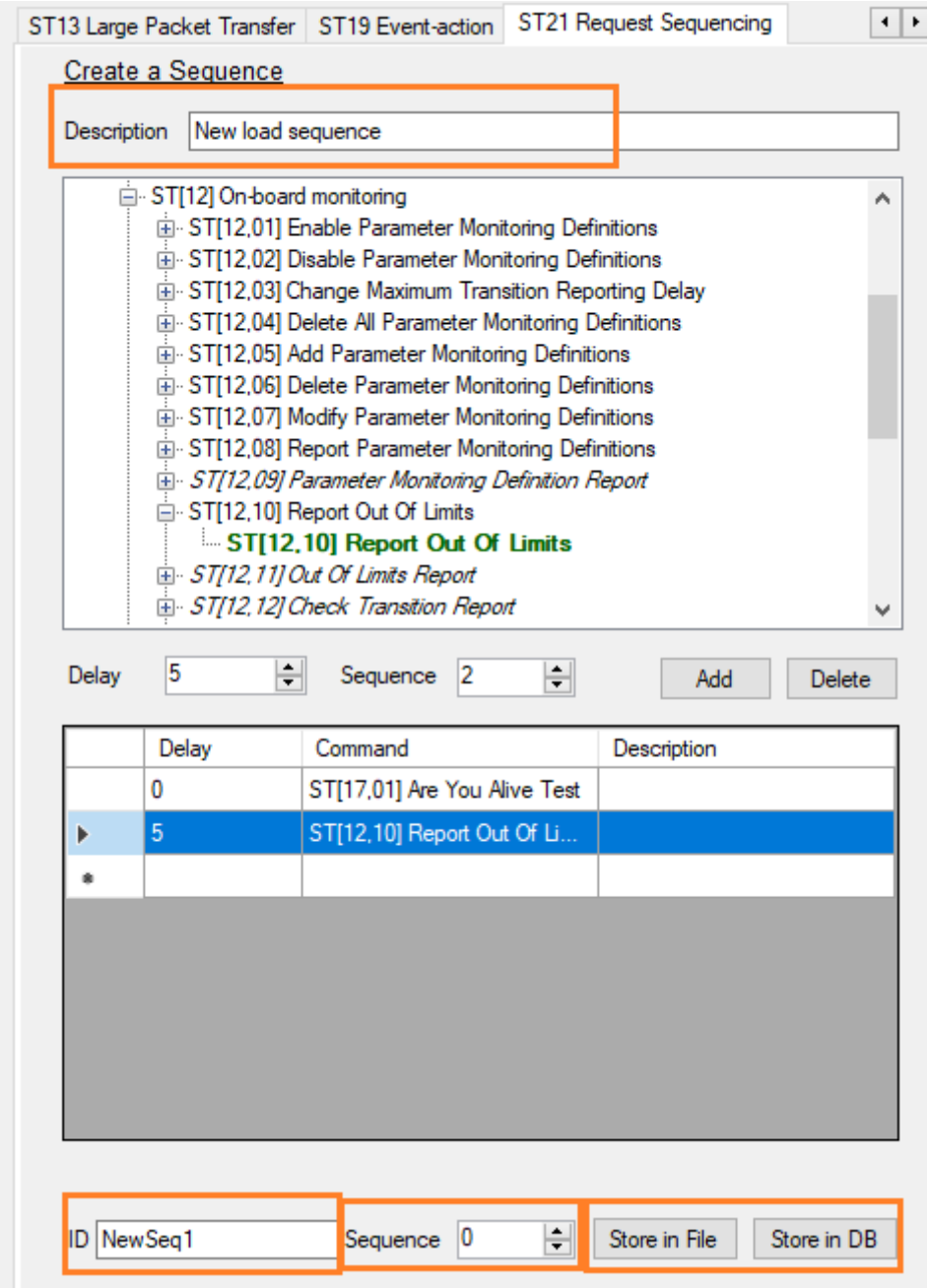


To delete a table entry select the row to delete and press the *delete* button

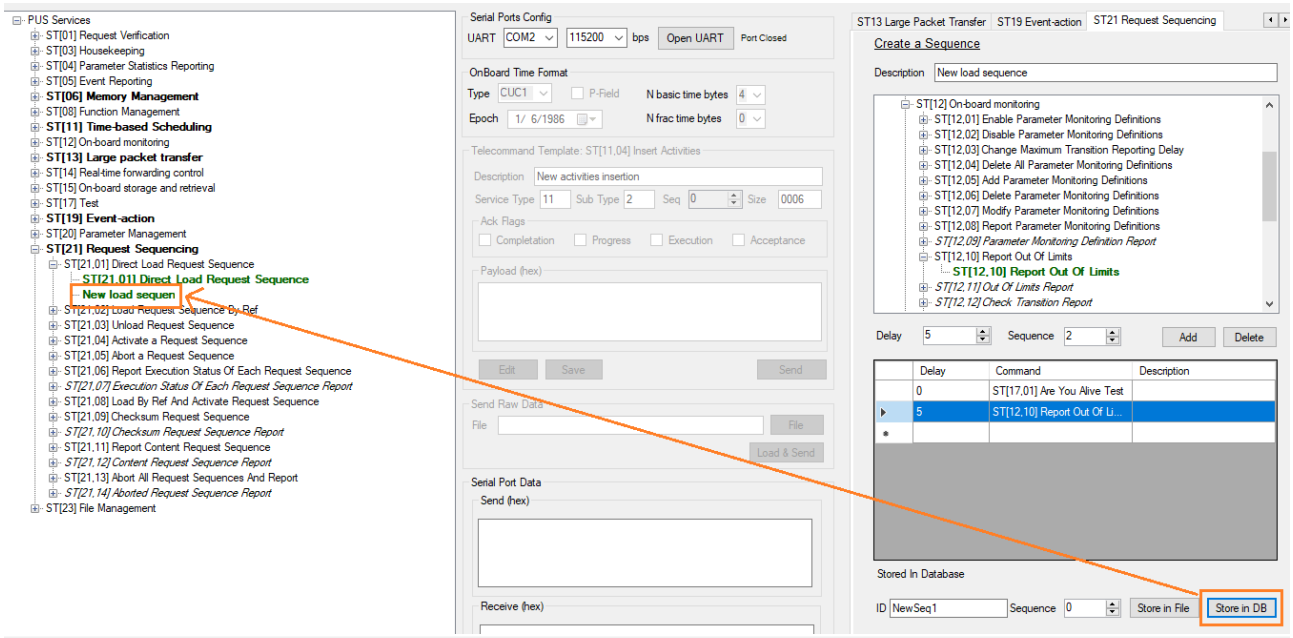


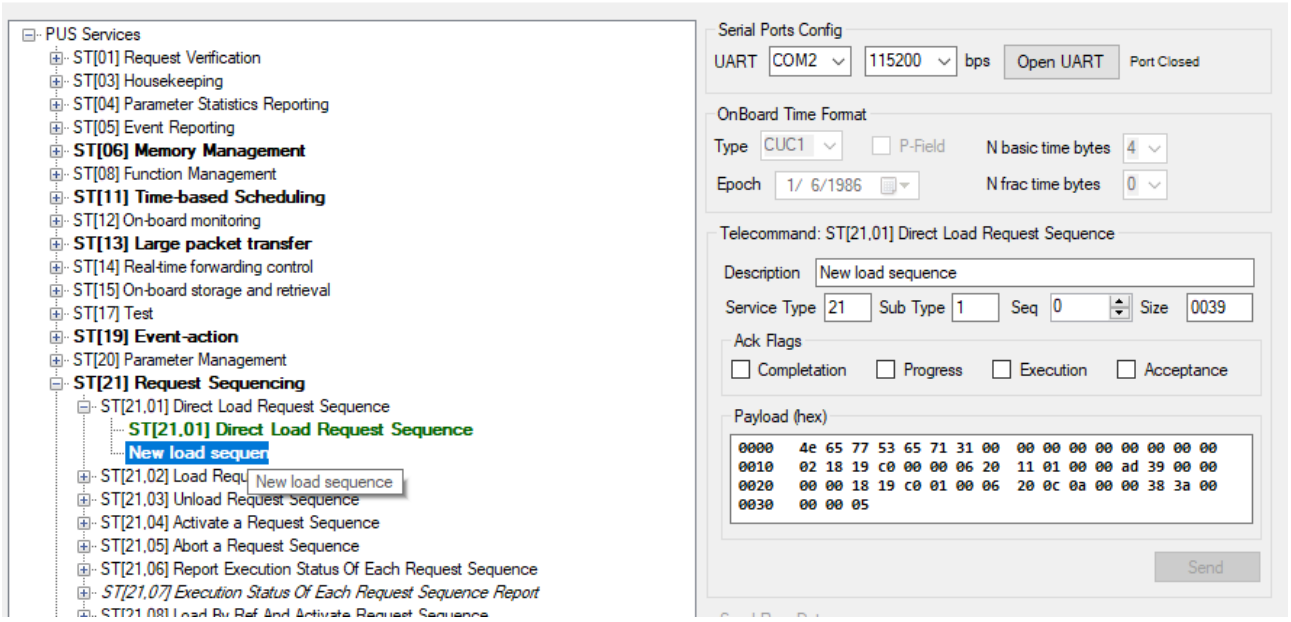
Once the table is filled with all the sequencing **Request** messages you wish to insert into a new sequence, fill the sequence name in the *ID* box, select the **Request** Message Sequence ID from *Sequence* box if you want to save the **Request** Message into a file and change it and fill the *Description* box with the new **Request** Message name, otherwise the default subservice name will be used

To create a new **Request** Message in the database under ST[21,01], press the *Store in DB* button, alternatively you are able to create a binary file with the **Request** Message pressing the *Store in File* button, this is usefull either to sent the **Request** Message as binary request without saving it into the database or if the **Request** Message is large enough to required the use of the Large Packet transfer service, which could use this binary file as input to split the **Request** Message



Once the **Request** Message is store into the database it could be use, see the Sending a Request Message from Database chapter





3.6 LOGS

Three log files are generated by the PUS Tester and stored into the directories set in the configuration file, see Configuration chapter.

The default directories are

- *InstallationPath\Logs* for Main Log
- *InstallationPath \Logs\SerialLogs* for UART Rx/Tx logs

3.6.1 MAIN LOG

The main log is saved into a file with the date time stamp in the file name using next format: ***YYYYMMDDhhmmss Log File.txt***

The main log registers the main events and actions the user does in the PUS Tester application, such as create, editing, deleting **Requests**, sending **Request** and receiving **Reports**
Each entry includes the local date time in the windows format used by the user and the event description. For **Requests** and **Reports**, also includes the payloads in hexadecimal ASCII format, for pure binary raw format see the UART RX and TX logs

Log Entry Example:

```
12/12/2023 14:01:34 Starting PUS Tester App
                PUS Tester APID: 3
12/12/2023 14:01:51 Opening UART: COM2, 115200, 8-N-1
12/12/2023 14:04:09 Rx:
0000   08 19 c0 00 00 0e 20 05  04 00 00 00 00 52 a3 55
0010   07 00 06 d9 3a

12/12/2023 14:04:16 Tx: ST[13,09] Uplink First Part
0000   18 03 c0 00 00 19 20 0d  09 00 00 00 01 00 00 01
0010   02 03 04 05 06 07 08 09  0a 0b 0c 0d 0e 0f 86 62
0020

12/12/2023 14:04:26 Rx:
0000   08 19 c0 01 00 0f 20 0d  10 00 00 00 00 52 a3 55
0010   19 00 01 01 df df

12/12/2023 14:04:30 Tx: ST[13,09] Uplink First Part
0000   18 03 c0 01 00 19 20 0d  09 00 01 00 01 00 00 01
0010   02 03 04 05 06 07 08 09  0a 0b 0c 0d 0e 0f a6 b7
0020

12/12/2023 14:04:33 Tx: ST[13,10] Uplink Intermediate Part
0000   18 03 c0 02 00 19 20 0d  0a 00 02 00 01 00 01 11
0010   12 13 14 15 16 17 18 19  1a 1b 1c 1d 1e 1f bf df
0020

12/12/2023 14:04:35 Tx: ST[13,10] Uplink Intermediate Part
0000   18 03 c0 03 00 19 20 0d  0a 00 03 00 01 00 03 31
0010   32 33 34 35 36 37 38 39  3a 3b 3c 3d 3e 3f 13 d5
0020

12/12/2023 14:04:37 Tx: ST[13,10] Uplink Intermediate Part
0000   18 03 c0 04 00 19 20 0d  0a 00 04 00 01 00 02 21
0010   22 23 24 25 26 27 28 29  2a 2b 2c 2d 2e 2f 3f 81
0020

12/12/2023 14:04:40 Tx: ST[13,11] Uplink Last Part
0000   18 03 c0 05 00 19 20 0d  0b 00 05 00 01 00 04 41
0010   42 43 44 45 46 47 48 49  4a 4b 4c 4d 4e 4f f7 cc
```

```
0020

12/12/2023 14:05:15 Tx: ST[13,09] Uplink First Part
0000   18 03 c0 06 00 19 20 0d  09 00 06 00 01 00 00 01
0010   02 03 04 05 06 07 08 09  0a 0b 0c 0d 0e 0f 44 9c
0020

12/12/2023 14:05:18 Tx: ST[13,10] Uplink Intermediate Part
0000   18 03 c0 07 00 19 20 0d  0a 00 07 00 01 00 03 31
0010   32 33 34 35 36 37 38 39  3a 3b 3c 3d 3e 3f 90 81
0020
```

3.6.2 UART RX/TX LOGS

The UART RX and TX logs are saved into two files with the date time stamp in the file names using next format:

YYYYMMDDhhmmss Log422RxFile.bin
YYYYMMDDhhmmss Log422TxFile.bin

The UART logs will created two binary files for each running contains the raw data sent and received by the UART during software usage, no additional data is included other than the UART RAW data

4 PUS TESTER SOFTWARE CONFIGURATION

The PUS Tester software uses an xml file to configure all the next startup variables

Nombre	Tipo	Ámbito	Valor
ApplicationID	ushort	Aplicación	25
TC_XML_file	string	Usuario	..\..\config\PUS_services.xml
Max_payload_size	int	Usuario	65536
DefaultCOMUART	string	Usuario	COM2
SerialLogRxFilePath	string	Usuario	..\..\Logs\SerialLogs\
SerialLogTxFilePath	string	Usuario	..\..\Logs\SerialLogs\
LogFilePath	string	Usuario	..\..\Logs\
OBTType	int	Usuario	0
PField	bool	Usuario	False
Epoch	System.Dat...	Usuario	1/6/1986
BasicTimeSize	int	Usuario	4
FracTimeSize	int	Usuario	0
CRCReports	bool	Usuario	True
Param_XML_file	string	Usuario	..\..\config\parametersdef.xml
ST8FunctionSize	int	Usuario	15
ST8FunctionArgSize	int	Usuario	15
ST21StringSize	int	Usuario	15
ST15StoreslDSize	int	Usuario	15

The configuration file *App.config* in the installation folder contains the default configuration for the PUS Tester, it can be open with a text editor to change the configuration:

```
    <applicationSettings>
        <PUS_tester.Properties.Settings>
            <setting name="ApplicationID" serializeAs="String">
                <value>25</value>
            </setting>
        </PUS_tester.Properties.Settings>
    </applicationSettings>
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    <startup>
    <applicationSettings>
    <userSettings>
        <PUS_tester.Properties.Settings>
            <setting name="TC_XML_file" serializeAs="String">
                <value>..\..\config\PUS_services.xml</value>
            </setting>
            <setting name="Max_payload_size" serializeAs="String">
                <value>65536</value>
            </setting>
            <setting name="DefaultCOMUART" serializeAs="String">
                <value>COM2</value>
            </setting>
            <setting name="SerialLogRxFilePath" serializeAs="String">
                <value>..\..\Logs\SerialLogs\</value>
            </setting>
            <setting name="SerialLogTxFilePath" serializeAs="String">
                <value>..\..\Logs\SerialLogs\</value>
            </setting>
            <setting name="LogFilePath" serializeAs="String">
                <value>..\..\Logs\</value>
            </setting>
            <setting name="OBTType" serializeAs="String">
                <value>0</value>
            </setting>
            <setting name="PField" serializeAs="String">
                <value>False</value>
            </setting>
            <setting name="Epoch" serializeAs="String">
                <value>1986-06-01</value>
            </setting>
            <setting name="BasicTimeSize" serializeAs="String">
                <value>4</value>
            </setting>
            <setting name="FracTimeSize" serializeAs="String">
                <value>0</value>
            </setting>
```

```
<setting name="ApplicationID" serializeAs="String">
    <value>25</value>
</setting>
```

The ApplicationID stores the unsigned 16bit integer with the destination Aplicación Process ID, all the request in the database will be sent to this AppID

```
<setting name="TC_XML_file" serializeAs="String">
    <value>.\config\PUS_services.xml</value>
</setting>
```

The TC_XML_file stores the file path to the PUS Service requests and Reports data base. The data base is an XML file. The path location is \Users\Public\Documents plus this setting

```
<setting name="Max_payload_size" serializeAs="String">
    <value>65536</value>
</setting>
```

The Max_payload_size variable defines the UART serial reception buffer

```
<setting name="DefaultCOMUART" serializeAs="String">
    <value>COM2</value>
</setting>
```

The DefaultCOMUART defines the default, at startup, serial port selected in the selection panel.

```
<setting name="SerialLogRxFilePath" serializeAs="String">
    <value>.\Logs\SerialLogs\</value>
</setting>
```

The SerialLogRxFilePath stores the path where the RX logs will be stored. The path location is \Users\Public\Documents plus this setting

```
<setting name="SerialLogTxFilePath" serializeAs="String">
    <value>.\Logs\SerialLogs\</value>
</setting>
```

The SerialLogTxFilePath stores the path where the TX logs will be stored. The path location is \Users\Public\Documents plus this setting

```
<setting name="LogFilePath" serializeAs="String">
    <value>.\Logs\</value>
</setting>
```

The LogFilePath stores the path where the general logs will be stored. The path location is \Users\Public\Documents plus this setting

```
<setting name="OBTType" serializeAs="String">
    <value>0</value>
</setting>
```

The OBTType defines the startup OBT configuration, currently only CUC1=0 and CUC2=1 are supported.

```
<setting name="PField" serializeAs="String">
    <value>False</value>
</setting>
```
The PField defines if the PField is present in the Time fields.

```
<setting name="Epoch" serializeAs="String">
    <value>1986-06-01</value>
</setting>
```
The Epoch defines the time date epoch when using CUC2.

```
<setting name="BasicTimeSize" serializeAs="String">
    <value>4</value>
</setting>
```
The BasicTimeSize defines the size in bytes of the integer time field. Current version of gr-pus only support 4 bytes.

```
<setting name="FracTimeSize" serializeAs="String">
    <value>0</value>
</setting>
```
The FracTimeSize defines the size in bytes of the fractional time field. Current version of gr-pus doesn't support fractional type, the leave this variable in 0 bytes.

```
<setting name="CRCReports" serializeAs="String">
     <value>True</value>
</setting>
```
The CRCReports defines if the Messages Reports include CRC or not.

```
<setting name="Param_XML_file" serializeAs="String">
     <value>.\config\parametersdef.xml</value>
</setting>
```
The Param_XML_file stores the file path to the parameters definition data base. The data base is an XML file. The path location is \Users\Public\Documents plus this setting

```
<setting name="ST8FunctionSize" serializeAs="String">
     <value>15</value>
</setting>
```
The ST8FunctionSize defines the size of the fixed string used in ST08 as Function name.

```
<setting name="ST8FunctionArgSize" serializeAs="String">
     <value>15</value>
</setting>
```
The ST8FunctionArgSize defines the size of the fixed string used in ST08 as Function Arguments.

```
<setting name="ST21StringSize" serializeAs="String">
```

```
    <value>15</value>
</setting>
```
The ST21StringSize defines the size of the fixed string used in ST21 as Sequences ID names.

```
 <setting name="ST15StoresIDSize" serializeAs="String">
     <value>15</value>
 </setting>
```
The ST15StoresIDSize defines the size of the fixed string used in ST15 as packet stores ID names.

5 PUS TESTER SOFTWARE DATABASE

The PUS Tester software ***Request*** and ***Reports*** messages database is in an xml file.

Warning: be aware that changing the xml by hand could cause a PUS Tester software malfunction, please avoid changing the xml file unleast a new ad-hoc PUS service is added or you wish to make a Request read only and always keep a backup of this file.

This file has three diferent node types

```
<service />
      <subservice />
            <packet />
```

service nodes are in the top hierarchy grouping each PUS service, these nodes have next properties

- `name:` mandatory, this properties is the node name as is list in the PUS services tree
- `type:` mandatory only if special GUI is used, this property indicates if the service node has special GUI. Set type=”255” for special GUI otherwise not include this property
- `service:` mandatory only if special GUI is used, this poperty indicate the service type number. It is used by special GUI to identify in which service the new Requests are stored
- `specialGUI:` mandatory only if special GUI is used, indicate the specialGUI tab index in the window Form

PUS_services.xml

```
<xml>
  <service name="ST[01] Request Verification">
  <service name="ST[03] Housekeeping">
  <service name="ST[04] Parameter Statistics Reporting">
  <service name="ST[05] Event Reporting">
  <service name="ST[06] Memory Management" type="255" service="06" specialGUI="0">
  <service name="ST[08] Function Management">
  <service name="ST[11] Time-based Scheduling" type="255" service="11" specialGUI="1">
  <service name="ST[12] On-board monitoring">
  <service name="ST[13] Large packet transfer" type="255" service="13" specialGUI="2">
  <service name="ST[14] Real-time forwarding control">
  <service name="ST[15] On-board storage and retrieval">
  <service name="ST[17] Test">
  <service name="ST[19] Event-action" type="255" service="19" specialGUI="3">
  <service name="ST[20] Parameter Management">
  <service name="ST[21] Request Sequencing" type="255" service="21" specialGUI="4">
  <service name="ST[23] File Management">
</xml>
```

subservice nodes follows the *service* nodes grouping each subservice **Request/Report**, these nodes have next properties

`name:` mandatory, this properties is the node name as is list in the PUS services tree

`type:` no mandatory, this property indicates if the subservice has a Request editing template form. Set type=”2” to use a templae GUI otherwise not include this property

`service:` mandatory only for type=”2”, this poperty indicate the service type number. It is used by Request editing template togheter with the subservice property to identify in which subservice the new Requests are stored

`subservice:` mandatory only for type=”2”, this poperty indicate the service sub type number. It is used by Request editing template togheter with the service property to identify in which subservice the new Requests are stored

`form:` mandatory only for type=”2”, indicate the template form name to use when editing/creating a new Request. The templates are implemented in the software code, if new templates are required then a software code modification will be required

PUS_services.xml

```
<xml>
  <service name="ST[01] Request Verification">
  <service name="ST[03] Housekeeping">
  <service name="ST[04] Parameter Statistics Reporting">
  <service name="ST[05] Event Reporting">
  <service name="ST[06] Memory Management" type="255" service="06" specialGUI="0">
    <subservice name="ST[06,02] Load Raw Memory Data Areas" type="2" service="6" subservice="2" form="FormRequestST06 02">
    <subservice name="ST[06,05] Dump Raw Memory Data" type="2" service="6" subservice="5" form="FormRequestST06 05">
    <subservice name="ST[06,06] Dump Raw Memory Data Report">
    <subservice name="ST[06,09] Check Raw Memory Data" type="2" service="6" subservice="9" form="FormRequestST06 09">
    <subservice name="ST[06,10] Check Raw Memory Data Report">
  </service>
  <service name="ST[08] Function Management">
  <service name="ST[11] Time-based Scheduling" type="255" service="11" specialGUI="1">
  <service name="ST[12] On-board monitoring">
  <service name="ST[13] Large packet transfer" type="255" service="13" specialGUI="2">
  <service name="ST[14] Real-time forwarding control">
  <service name="ST[15] On-board storage and retrieval">
  <service name="ST[17] Test">
    <subservice name="ST[17,01] Are You Alive Test">
    <subservice name="ST[17,02] Are You Alive Test Report">
    <subservice name="ST[17,03] OnBoard Connection Test">
    <subservice name="ST[17,04] OnBoard Connection Test Report">
  </service>
  <service name="ST[19] Event-action" type="255" service="19" specialGUI="3">
  <service name="ST[20] Parameter Management">
  <service name="ST[21] Request Sequencing" type="255" service="21" specialGUI="4">
  <service name="ST[23] File Management">
</xml>
```

packet nodes follows the *subservice* nodes and are the individual **Request/Report** messages, these nodes have next properties

`id:` mandatory and unique, this properties identify each part in an unequivocal way

`name:` mandatory, this properties is the node name matching the subservice name, if description property is not present, this property is used as node description

`description:` no mandatory, this property is a user friendly identification of the request. If present, this property will be used as name in the PUS services tree

`type:` mandatory, this property indicates if the packet is a **Request** type=”1” or a **Report** type=”0”.

`service:` mandatory, this poperty indicate the service type number of the **Request/Report**.

`subservice:` mandatory, this poperty indicate the service subtype number of the **Request/Report.**

`ack:` mandatory, this poperty indicate the default acknolodge flags configuration for the **Request** as [A.D.1]. These flags could be changed before send the **Request**, see Sending a Request Message From Database chapter

`data:` mandatory, this poperty indicate the payload data of the Request as a hex ASCII string e.i. “aa55012354” represent a payload of 0xaa 0x55 0x01 0x23 0x54

`form:` no mandatory, for a **Request**, type=”1” indicate the template form name to use when editing the Request. The templates are implemented in the software code, if new templates are required then a software code modification will be required. If this property is not included in a packet, then these **Request** packet is not editable from the PUS Tester main interface, then, be sure to remove this property from read only **Requests**

`template:` no mandatory, for a **Report**, type=”0” indicate the template to decode the Report in a user friendly interface. The templates are dynamically generated from xml files, additional user ad-hoc templates could be incorporated without software code changing, see next chapter

```
PUS_services.xml
<xml>
  <service name="ST[01] Request Verification">
  <service name="ST[03] Housekeeping">
    <subservice name="ST[03,01] Create Housekeeping Report Structure" type="2" service="3" subservice="1" form="FormRequestST03_01">
      <packet id="006" name="ST[03,01] Create Housekeeping Report Structure" description="Creates a telemetry structure" type="1" service="3" subservice="1" ack="15"
    </subservice>
    <subservice name="ST[03,03] Delete Housekeeping Report Structure" type="2" service="3" subservice="3" form="FormRequestST03_03">
      <packet id="007" name="ST[03,03] Delete Housekeeping Report Structure" type="1" service="3" subservice="3" ack="15" data="020104" form="FormRequestST03_03">
      </packet>
    </subservice>
```

5.1 REPORT TEMPLATES

Report xml templates are used to dinamically generated a user friendly GUI displaying the received **Reports**, see Viewing Report Messages chapter.
These templates have diferent node types

```
<template />
        <item />
                <subitem />
                <option />
```

template node is the top hierarchy node, having next properties

`name:` mandatory, this properties is the template name as is displayed in the GUI Form

`l:` mandatory, this property indicates the GUI Form length of the template

`h:` mandatory, this property indicates the GUI Form height of the template

```
report_tm[12_12].xml
<xml>
    <template name="ST[12,12] Check Transition Report" l="576" h="770">
        <item type="Repeater" x="120" y="30" data_type="uint16" data_size="2" data_mask="0xffff" data_shift="0" >
    </template>
</xml>
```

item/subitem nodes have same properties, but the *item* nodes follow the *template* node and the *subitem* nodes follows *item, subtiem* or *option* nodes, those nodes have next properties

`type:` mandatory, this properties indicate the type of control to add into the GUI, valid controls are:

- `"Label"` add a Label control with a fixed text
- `"CheckBox"` add a CheckBox control for bool variables from input **Report** data
- `"TextBox"` add a TextBox control to display variables from input **Report** data as text
- `"Repeater"` no control is added, but all subnodes are readed N times adding the subnodos controls N times. N is read from input **Report** data
- `"ListRepeater"` as the Repeater but N is obtained from a data_list, these data_list I s alist of Parameters ID which will be parsed during each step. This control is for display Housekeeping reports only (ST[03,25] Housekeeping Parameters Report), the data_list is the list of parameters excepted in a Housekeeping report
- `"ComboBox"` add a ComboBox control to display variables from input **Report** data as options in the ComboBox

`text:` mandatory for TextBox and Label types, this properties indicate the text to display in the Label or in the TextBox as default

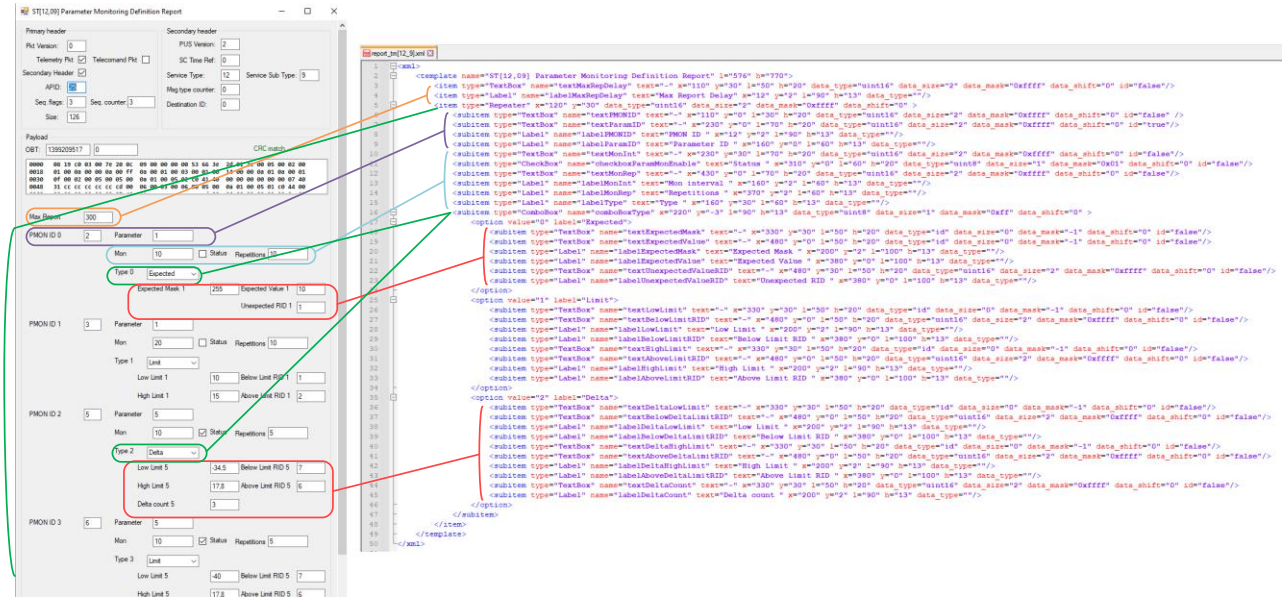
`x:` mandatory, this property indicates the GUI Form control x position in the template as absolute value

`y:` mandatory, this property indicates the GUI Form control y position in the template as relative value from previous control

`l:` mandatory, this property indicates the GUI Form control length of the template

`h:` mandatory, this property indicates the GUI Form control height of the template
`data_type:` mandatory for all except Labels, this property indicates variable type to parse from input **Report** data and to be displayed in the control added. Valid values are uint8, int8 (or sint8), uint16, int16 (or sint16), uint32, int32 (or sint32), uint64, int64 (or sint64), bool, float, double, str, request, id. Data_type str represent a string variable, request represent a Request message variable, id represent the variable is a Parameter value, then it will depends upon parameter configuration, see next chapter
`data_size:` mandatory for all except Labels, this property indicates variable size to parse from input **Report** data and to be displayed in the control added. For data_type uint8, int8 (or sint8), and bool data_size="1", for uint16, int16 (or sint16) data_size="2", for uint32, int32 (or sint32) and float data_size="4", for uint64, int64 (or sint64) and double data_size="8", for str data_size="0" if expected a variable string or the string size for fixed strings, for request data_size="0" (the size is obtained from the Request message size field), for id data_size="0" (the size is obtained from the parameter configuration file)
`data_mask:` mandatory for all except Labels, this property indicates the masking to apply to the value parse from input **Report** data, in general no masking is required so left this value to FFs, for str or id left this value in -1
`data_offset:` mandatory for all except Labels, the data from input **Report** data is parsed in order, if need to skip bytes, set the data offset to the number of byte to skip before parsing the value from input **Report** data. In general this value is set to 0
`id:` mandatory for all except Labels, if ii is true indicates the variable read from input **Report** data is a parameter ID, then it is used as key to identify the parameter for follow controls with id as data_type

option nodes are used for *item/subitmes* nodes with type="ComboBox", those nodes list the ComboBox options to be included in the ComboBox list. The selected option from the ComboBox is obtained from the **Report** input data according to the ComboBox *item/subitem* node. An *option* node could has child *subitems* nodes in cases where the GUI need to be changed according to the selected option
option nodes have next properties
`value:` mandatory, this properties indicate the ComboBox index of this option
`label:` mandatory, this properties indicate the ComboBox text to add to the ComboBox index



6 PUS TESTER SOFTWARE PARAMETERS CONFIGURATION

The PUS Tester software uses an xml file to configure the parameters types, these parameters are used in several ***Request*** and ***Reports*** messages and PUS services and shall match the parameters in the gr-pus implementation. Knowing the parameters IDs and type are needed to decode the ***Reports*** and generate the ***Request*** messages. If an improper configuration is set, then the PUS tester might cause exceptions

PUS_services.xml | parametersdef.xml

```xml
<xml>
    <param id="1" name="Parametro 1" type="uint8" data_size="1"/>
    <param id="5" name="Parametro 5" type="double" data_size="8"/>
    <param id="11" name="Parametro 11" type="bool" data_size="1" />
    <param id="34" name="Parametro 34" type="int16" data_size="2"/>
    <param id="15" name="Parametro 15" type="double" data_size="8"/>
    <param id="16" name="Parametro 16" type="uint8" data_size="1"/>
    <param id="31" name="Parametro 31" type="uint8" data_size="1"/>
    <param id="32" name="Parametro 32" type="uint16" data_size="2"/>
    <param id="128" name="Parametro 128" type="uint16" data_size="2"/>
</xml>
```

To add a parameter configuration into the xml file, simple add a new entry <param /> with the parameter data. If you wish to delete a parameter simple delete the row

```
<param id="32" name="Parametro 32" type="uint16" data_size="2"/>
```

The `id` field is the parameter identification as in gr-pus
The `name` is a user friendly description not used by the PUS Tester software but usefull to identificate the parameter when the xml is read
The `type` is the parameter type, either uint8, int8 (or sint8), uint16, int16 (or sint16), uint32, int32 (or sint32), uint64, int64 (or sint64), bool, float or double
The `data_size` is the parameter size in octets as follows uint8=1, int8/sint8=1, uint16=2, int16/sint16=2, uint32=4, int32/sint32=4, uint64=8, int64/sint64=8, bool=1, float=4 or double=8

Note:

The gr-pus uses a json file instead of xml files, but a python converter is included in the gr-pus apps folder as json2xml allowing to convert the json file used to configure the gr-pus into a xml usefull for the PUS Tester