

TALOS+CIRCUS

How to work with CIRCUS?

Table of contents

I. [CIRCUS structure](#)

1. [TALOS](#)
 - a. [TALOS folder](#)
 - b. [TALOS config file](#)
2. [Arranging LabView code](#)
 - a. [PC specific projects](#)
 - b. [uServices](#)
 - c. [Custom classes](#)
3. [CIRCUS errors file](#)
4. [CIRCUS config file](#)

II. [Using CIRCUS](#)

1. [Launching CIRCUS](#)
 - a. [As distributed system](#)
 - b. [As stand-alone Guardian](#)
2. [Guardian overview](#)
3. [TALOS palette and useful functions](#)

III. [Making a new uService](#)

1. [Creating from the template](#)
 - a. [Hardware class](#)
 - b. [Detector class](#)
2. [Example](#)
 - a. [Step by step simple chat uService](#)
 - b. [Starting uService](#)
 - c. [Using uService](#)

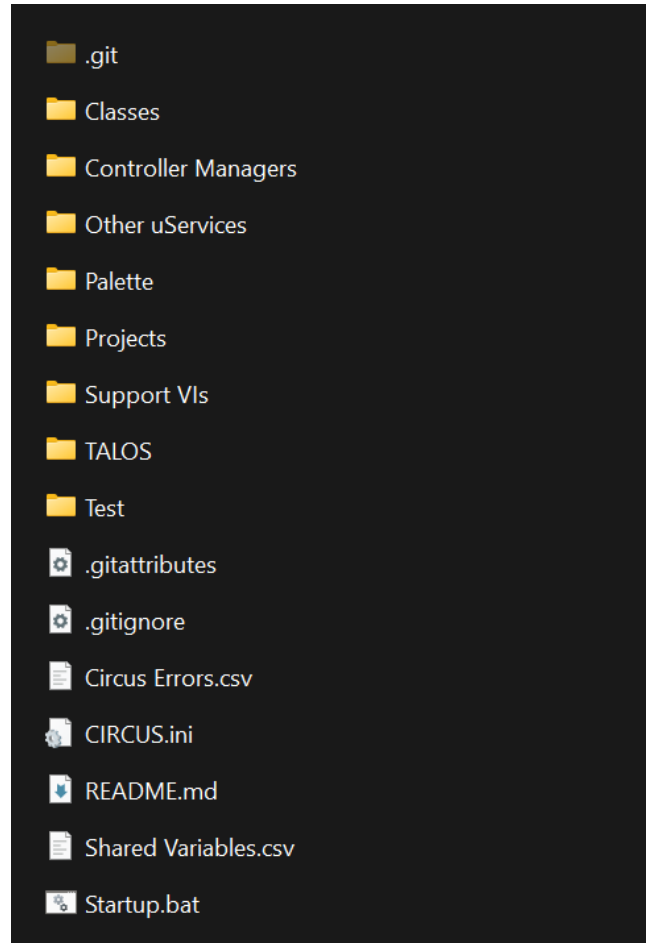
I. CIRCUS structure

1. [TALOS](#)
 - a. [TALOS folder](#)
 - b. [TALOS config file](#)
2. [Arranging LabView code](#)
 - a. [PC specific projects](#)
 - b. [uServices](#)
 - c. [Custom classes](#)
3. [CIRCUS errors file](#)
4. [CIRCUS config file](#)

CIRCUS - Computer Interface for Reliably Controlling, in an Unsupervised manner, Scientific experiments

Project based on TALOS that allows for introducing new uServices to perform specific tasks

It provides the skeleton for the Guardian with uServices system



Built in TALOS

CIRCUS – built in TALOS
**This is the folder that holds
the compiled TALOS library**

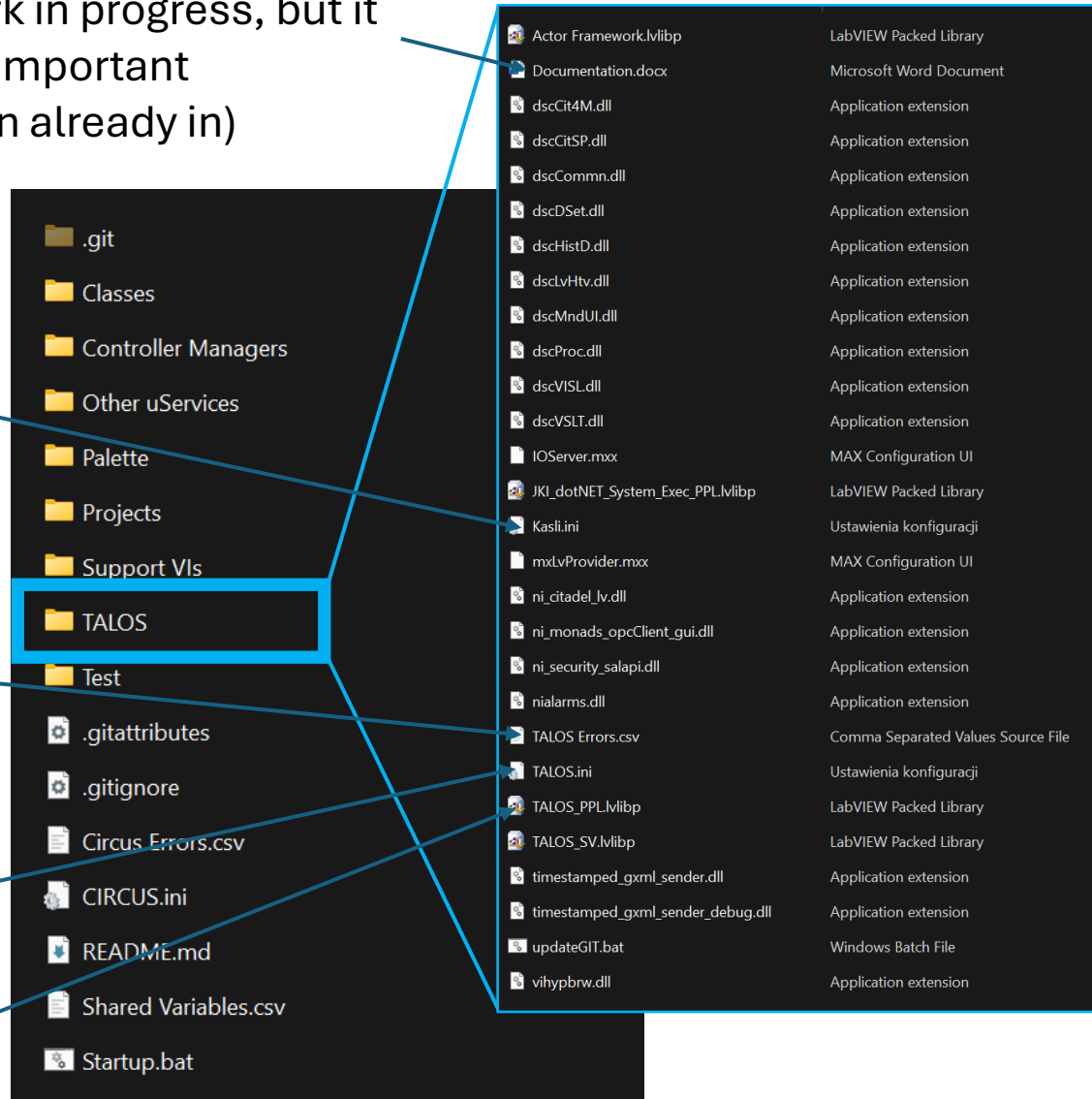
Documentation of TALOS
(this is work in progress, but it
has some important
information already in)

Configuration file for the Kasli
controllers (allowing TALOS to be
integrated with Sinara control-
ecosystem)

Pre-defined custom TALOS errors

Configuration file for TALOS

TALOS library file



CIRCUS – built in TALOS

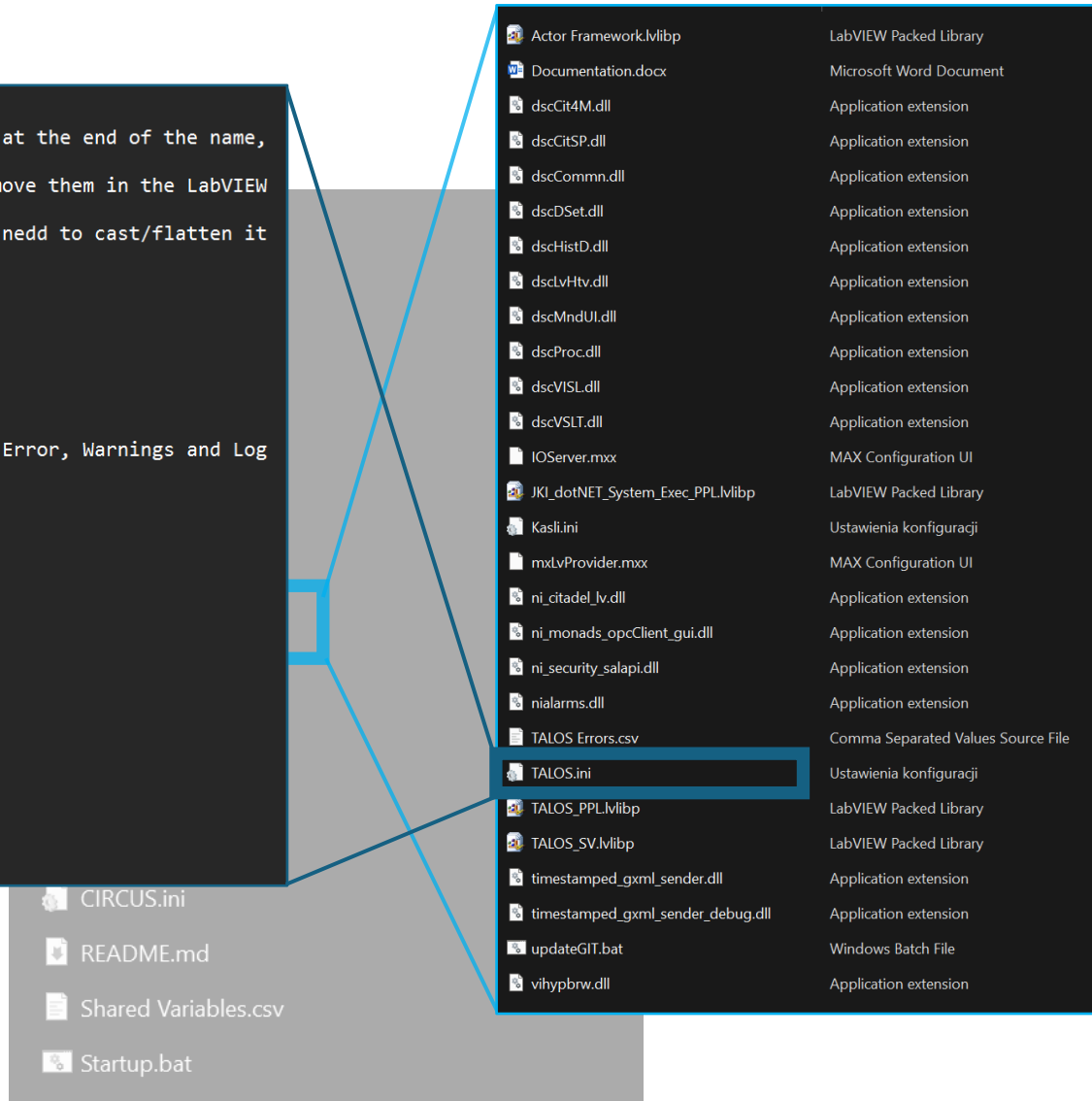
TALOS configuration file

```
;This is the Configuration Parameters File.
;Please, if the parameter has a unit (like seconds), append it at the end of the name,
with an underscore (like name_s).
;Please, for the paths, wrap them around double quotes, and remove them in the LabVIEW
code.
;Internally, all the parameters are handled as strings, so you need to cast/flatten it
into the appropriate data type.

[TALOS]
uServices_watchdog_triggering_time_s = 60.0
Guardian_watchdog_triggering_time_s = 30.0
Guardian_check_tau_s = 5.0
uServices_check_tau_s = 5.0
ABORT_check_tau_s = 30.0
; Verbosity level: 1 = Error only, 2 = Error and Warnings, 3 = Error, Warnings and Log
Verbosity = 3
; Relative path to Root path
Log_File_Path = "C:\TALOS Log Files"
SV_Timeout_ms = 1000
Timeout_of_RUN_s = 30
uServices_replies_check_tau_s = 1
TCP_Writer_Connection_Check_s = 2
Rotate_files_days = 7
Rotate_files_MB = 50
Highlanders_List = Error Manager
Personal_Log =
GUI_Skins =
Project_INI = CIRCUS.ini

[ERRORS]
Errors_to_ignore = 5359,5362
```

These parameters can be used to change configure behaviour of TALOS for your needs.



CIRCUS – built in TALOS

TALOS configuration file

```
;This is the Configuration Parameters File.
;Please, if the parameter has a unit (like seconds), append it at the end of the name,
with an underscore (like name_s).
;Please, for the paths, wrap them around double quotes, and remove them in the LabVIEW
code.
;Internally, all the parameters are handled as strings, so you need to cast/flatten it
into the appropriate data type.

[TALOS]
uServices_watchdog_triggering_time_s = 60.0
Guardian_watchdog_triggering_time_s = 30.0
Guardian_check_tau_s = 5.0
uServices_check_tau_s = 5.0
ABORT_check_tau_s = 30.0
; Verbosity level: 1 = Error only, 2 = Error and Warnings, 3 = Error, Warnings and Log
Verbosity = 3
; Relative path to Root path
Log_File_Path = "C:\TALOS Log Files"
SV_Timeout_ms = 1000
Timeout_of_RUN_s = 30
uServices_replies_check_tau_s = 1
TCP_Writer_Connection_Check_s = 2
Rotate_files_days = 7
Rotate_files_MB = 50
Highlanders_List = Error Manager
Personal_Log =
GUI_Skins =
Project_INI = CIRCUS.ini

[ERRORS]
Errors_to_ignore = 5359,5362
```

These parameters can be used to change configure behaviour of TALOS for your needs.

- watchdog_triggering_time_s -> how long system waits before it considers Guardian/uService offline
- check_tau_s -> how often Guardian/uService sends the message that it is still online
- ABORT_check_tau_s -> how often the ABORT flag is updated (if system sees ABORT it goes into idle state and no messages from outside can control uServices)
- Log_File_Path -> where TALOS will create all log files (error logs, uService specific logs, system logs, etc.)
- Highlanders_List -> uServices that need to always be running (if uService stops, the system will try to restart it)
- Personal_Log -> uServices that save log files with what messages it receives and sends together with consumer cases executed
- Project_INI -> name of the configuration file for CIRCUS (this file should be in the main directory of the project)
- Errors_to_ignore -> list of error codes that will not be visible on the system (the errors will still be logged in the error log)

CIRCUS.ini
README.md
Shared Variables.csv
Startup.bat

Arranging LabView Code

CIRCUS – PC specific projects

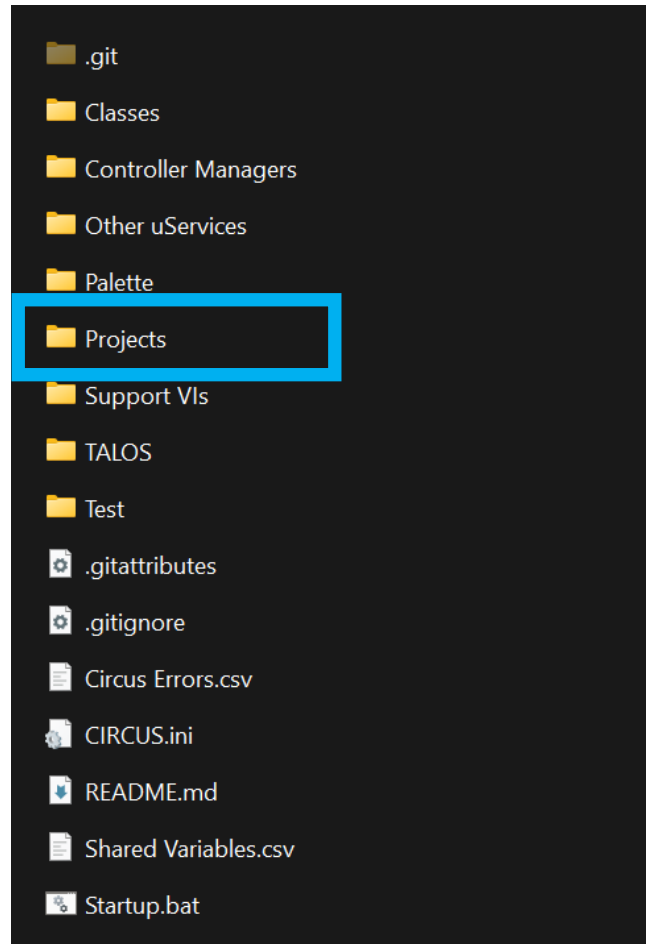
This is the folder that LabView projects for each PC in your system

CIRCUS based on TALOS is a distributed system that can be run on multiple PCs connected via a network with TCP protocol.

Each PC should have it's own project with all uServices that run on that PC inside the project.

This allows PCs to have different software on it (special device drivers).

It also allows for parallel development of uServices for different PCs. Projects are independent of each-other (unless you need to use the same uService).



The project name should reflect the name of the PC (as set up in Windows settings). This is used to start specific project whenever starting CIRCUS in the full system mode.

For the system to start on the machine it needs to have the PC's project opened and there cannot be any errors in that project.

CIRCUS – uServices

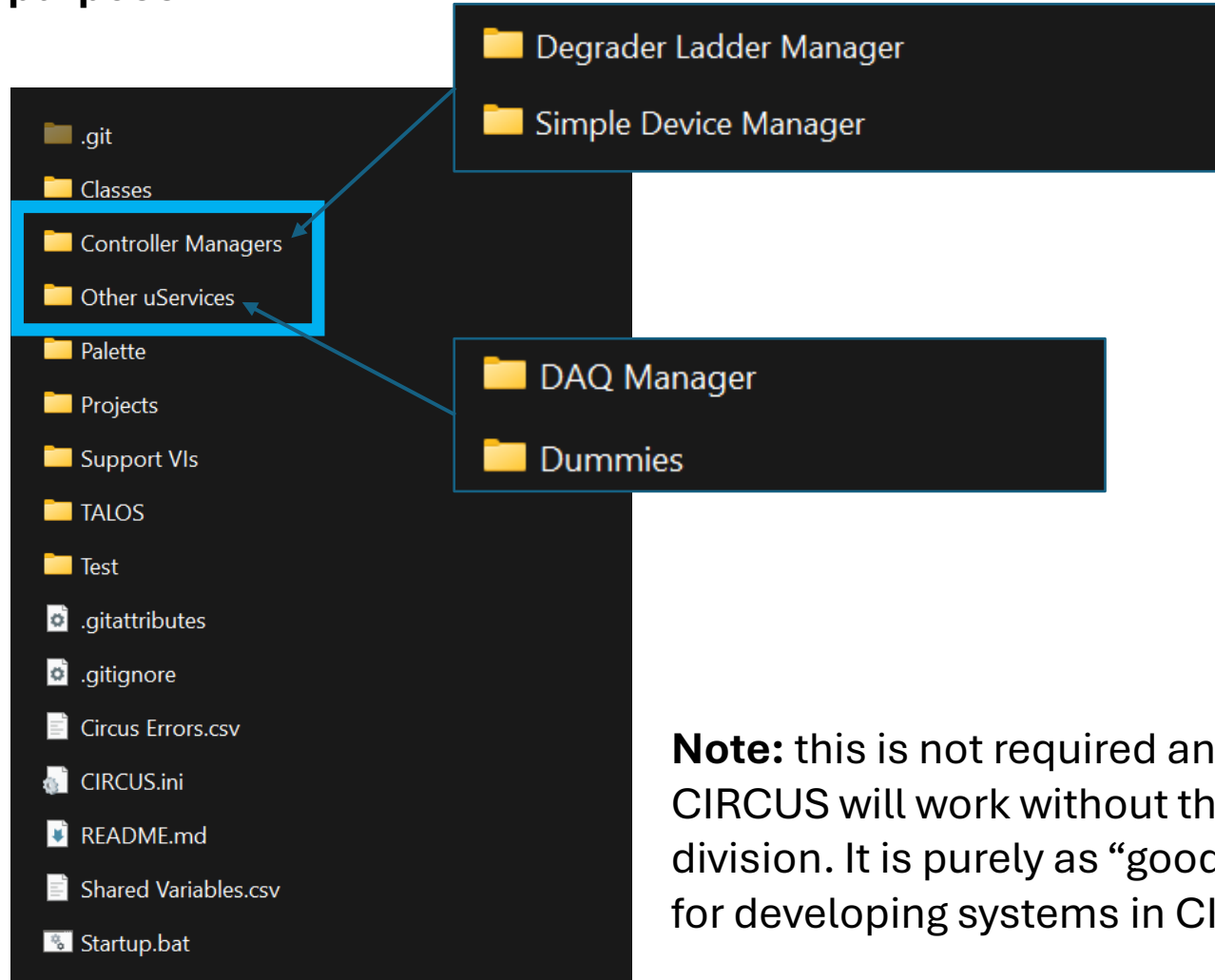
To keep the **CIRCUS** folder clean it is recommended to put **uServices** in subfolders based on their purpose.

uServices usually are called Managers.

uService that controls a controller, for example a piston that moves up and down, should be saved in the **Controller Managers** folder.

uService that controls a detector, for example an oscilloscope, should be saved in the **Detector Managers** folder (this folder not shown on this configuration as it doesn't implement any detectors).

uService that isn't controlling a hardware device can be put in the **Other uServices** folder



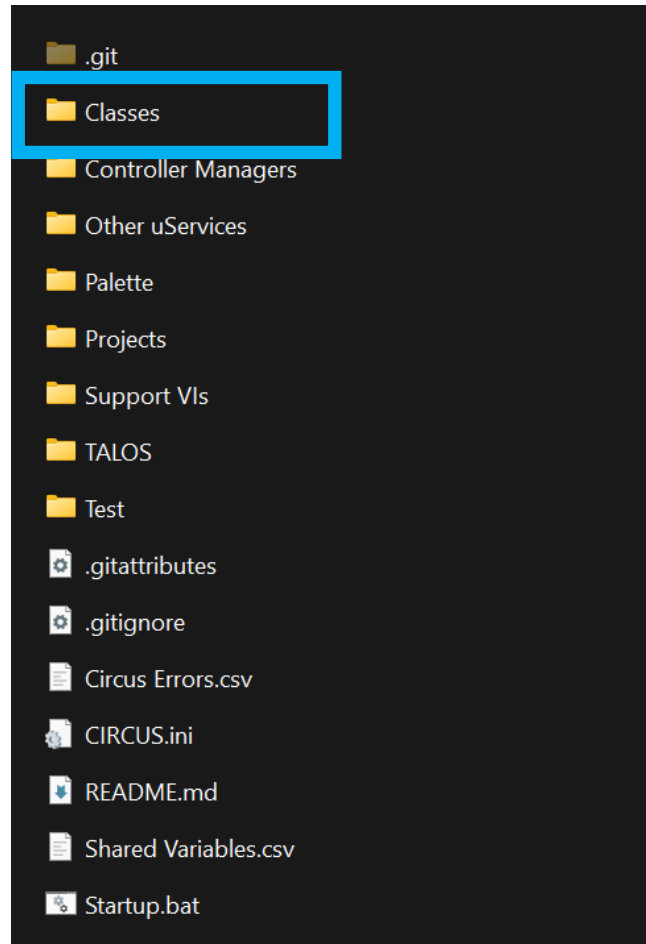
Note: this is not required and the CIRCUS will work without this division. It is purely as “good practice” for developing systems in CIRCUS.

CIRCUS – user specific custom classes

This folder is intended for custom classes that are used to implement specific interfaces from TALOS or just custom classes used in your system.

TALOS provides couple of interfaces like Analysis Framework, DAQ Communicator, FPGA Interface, etc. This allows to define custom behaviour.

For example, each system will be using a specific DAQ infrastructure. This allows the developer to define functions for communicating with the DAQ (sending data or checking the status) which will be automatically used by TALOS.



Note: this is not required and the CIRCUS will work without this division. It is purely as “good practice” for developing systems in CIRCUS.

Error codes in CIRCUS

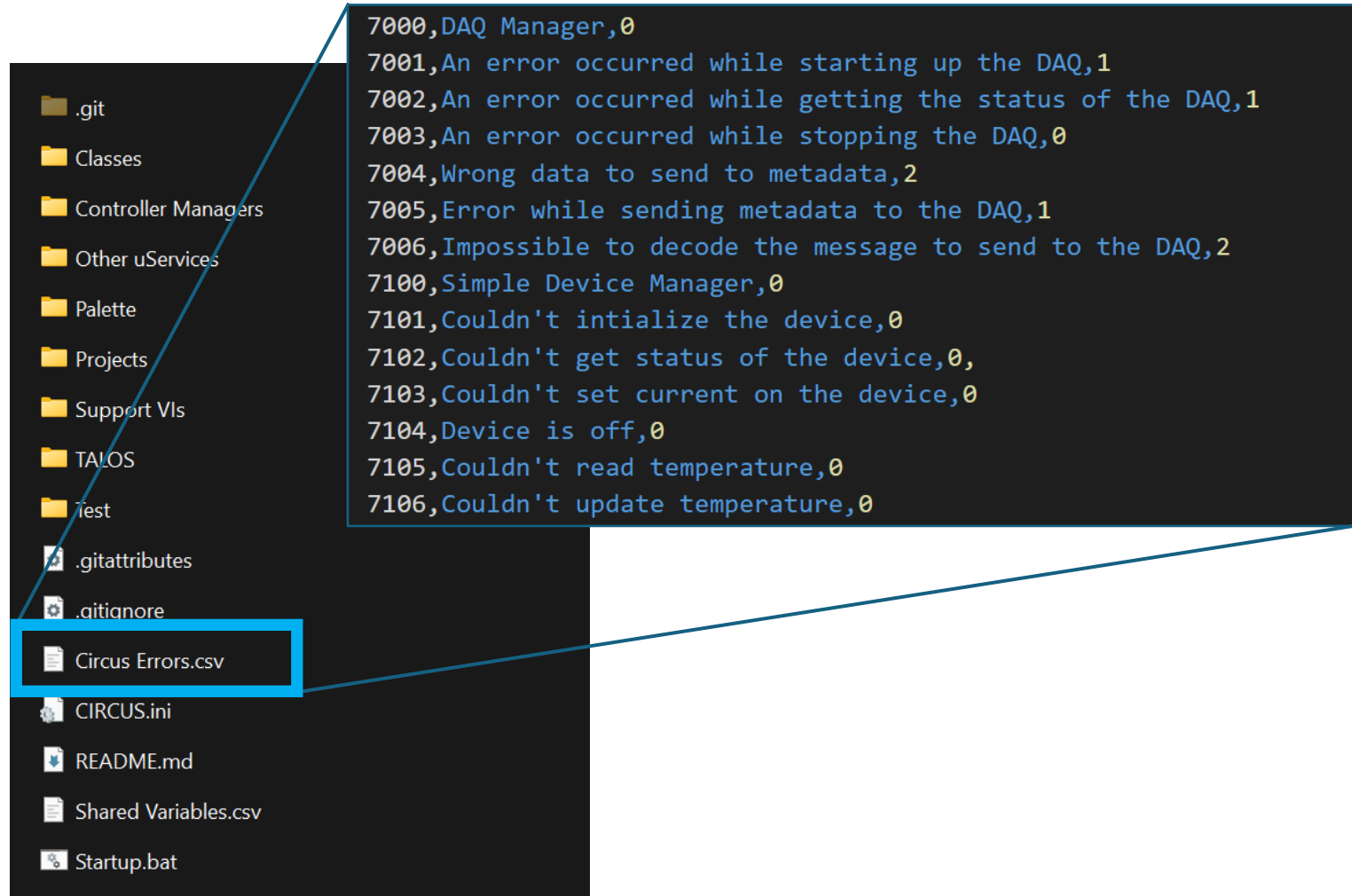
CIRCUS – defining custom errors

Proper error handling is one of the main requirements of the control system. This is done by TALOS (to some extent).

Errors that are generated by a uService:

- Are translated into understandable error messages
- Don't create an error loop (if something creates an error make sure it isn't also called infinitely)

Circus Errors.csv is a file that holds definitions of custom errors inside Circus



CIRCUS – defining custom errors

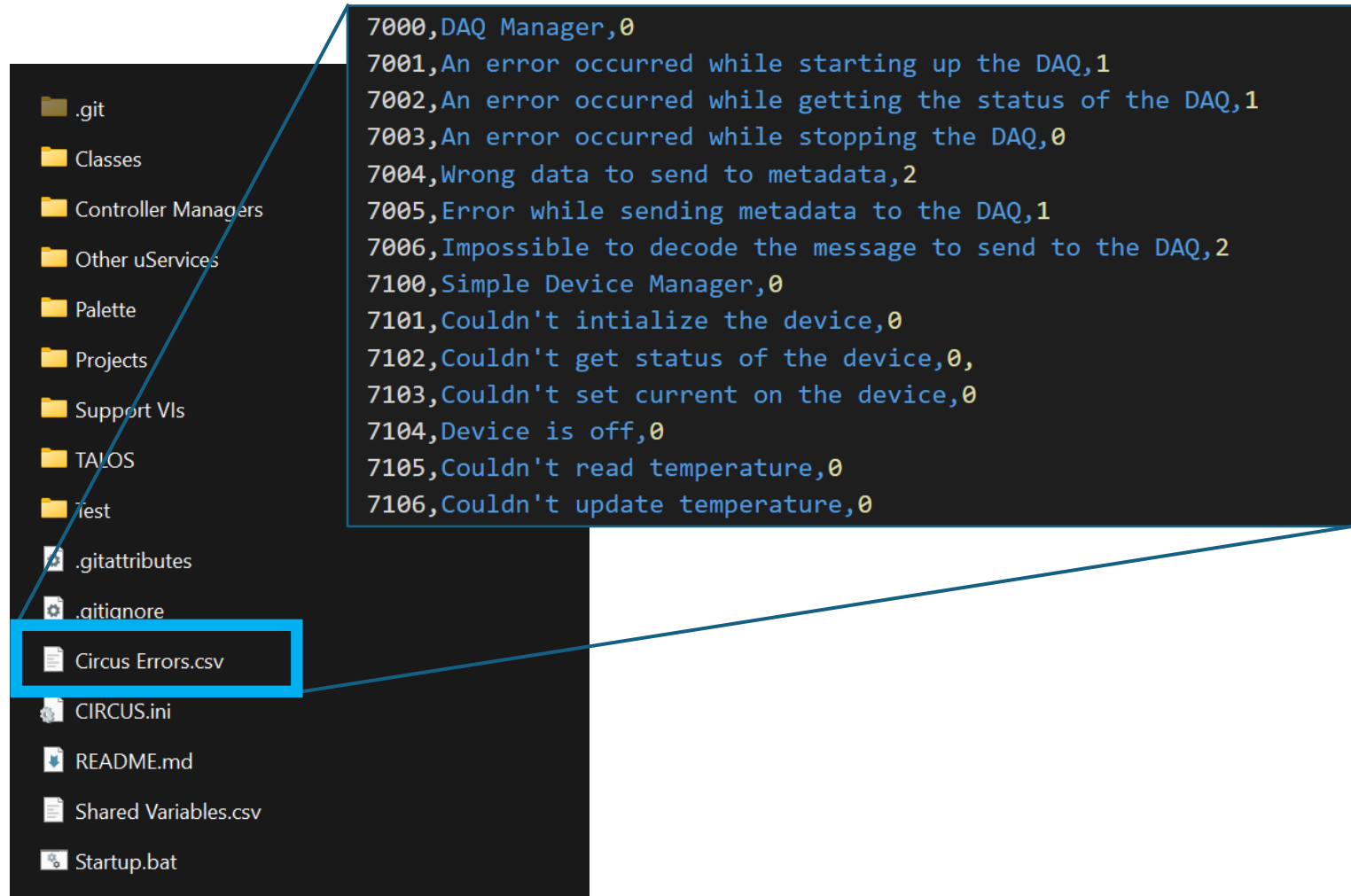
Proper error handling is one of the main requirements of the control system. This is done by TALOS (to some extent).

Errors are arranged in increasing order starting from 7000 for uServices.

Assumed convention is to dedicate 50 error codes for each uService where the starting error is not used and defines the name of the uService (7000, 7050, 7100, etc.).

Errors are saved in three columns:
Error code | Description | Error criticality

Error criticality is used by the system when using Tamer and Monkey for running series of experiments (Schedules) and defines the response of the system to each error (skip, stop, etc.)



CIRCUS config file

CIRCUS – configuring the system

Configuration file with parameters used inside CIRCUS.
This is where you can declare the configuration of PCs in your system and parameters used by defined uServices

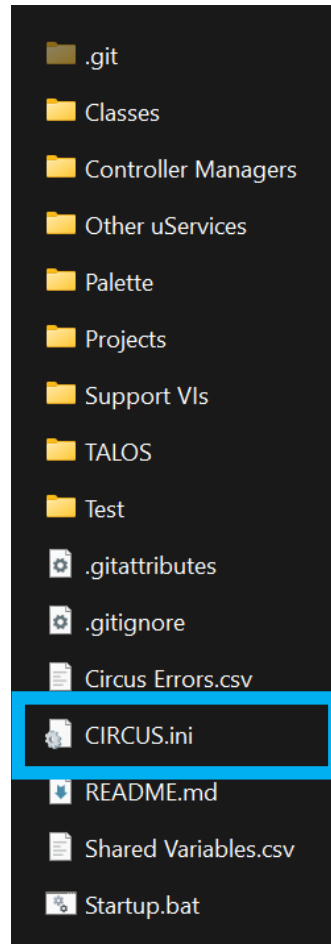
File uses the same format as the TALOS config file

All parameters are stored as strings and can be later converted into specified types

Parameters from this file can be accessed in the LabView code via the CIRCUS parameters.vi

Parameters are divided into sections, usually based on the uService that uses the parameters.

Parameter names should be unique (no duplication, even between sections). If parameter has a unit, append it at the end of the name.



```
;This is the CIRCUS Configuration Parameters file.
;If the parameter has a unit (like seconds), append it at the end of the name, with an
underscore (like name_s).
;For the paths, wrap them around double quotes, and remove them in the LabVIEW code (use
the VI in Support VIs).
;Internally, all the parameters are handled as strings, so you need to cast/flatten it into
the appropriate data type.

[General]
PC_List = my_pc_name1, my_pc_name2
Crit_PC_S =
;In the following the IPs of the PCs can be specified, in the format PC_Name, IP, PC_Name,
IP, etc.
IPs = my_pc_name1, 127.0.0.1, my_pc_name2, 127.0.0.1
;
; Matrix: for every pc, the uServices to launch at startup
local = Error Manager, Tamer, Scheduler
my_pc_name1 = Error Manager, Tamer, Scheduler, DAQ Manager
my_pc_name2 = Kasli Server
aegis-2 = Error Manager, Tamer, Scheduler
;
Data_Path = "C:\CIRCUS Data"
Highlanders_List = Scheduler, Tamer
Personal_Log = DAQ Manager, Tamer

[ALPACA_Optimiser]
ALPACA_Path = "C:\ALPACA\python-analyses"
optimiser-with-training-beta = True
optimiser-history-file-path = "C:\TALOS Log Files\OPTIMISER\training-history.csv"

[AEgIS DAQ]
DAQ_Bomb_Fuse_s = 45
GXML_DLL_Debug = False
DAQ_Data_Address =
GXML_Native_LV = False
SCP_Timeout_s = -1
;If the SCP Timeout is set to -1, the timeout is not set at all

[Simple device]
Simple-device-temp-check_s = 0.1
```

CIRCUS – configuring the system

Configuration file with parameters used inside CIRCUS.
This is where you can declare the configuration of PCs in your system and parameters used by defined uServices

General parameters are used for configuring the CIRCUS system:

- PC_List -> names of the PCs that are currently used as the distributed system
- Crit_PCs -> names of the PCs that are excluded from standard updates from git of the system
- IPs-> map of the IPs for all available PCs
- <PC_NAME> -> list of uServices that will be started automatically on the specified PC
- Data_Path -> path where TALOS will save data from detectors as a backup
- Highlanders_List -> uServices that need to always be running (if uService stops, the system will try to restart it)
- Personal_Log -> uServices that save log files with what messages it receives and sends together with consumer cases executed

```
;This is the CIRCUS Configuration Parameters file.
;If the parameter has a unit (like seconds), append it at the end of the name, with an
underscore (like name_s).
;For the paths, wrap them around double quotes, and remove them in the LabVIEW code (use
the VI in Support VIs).
;Internally, all the parameters are handled as strings, so you need to cast/flatten it into
the appropriate data type.

[General]
PC_List = my_pc_name1, my_pc_name2
Crit_PCs =
;In the following the IPs of the PCs can be specified, in the format PC_Name, IP, PC_Name,
IP, etc.
IPs = my_pc_name1, 127.0.0.1, my_pc_name2, 127.0.0.1
;
; Matrix: for every pc, the uServices to launch at startup
local = Error Manager, Tamer, Scheduler
my_pc_name1 = Error Manager, Tamer, Scheduler, DAQ Manager
my_pc_name2 = Kasli Server
aegis-2 = Error Manager, Tamer, Scheduler
;
Data_Path = "C:\CIRCUS Data"
Highlanders_List = Scheduler, Tamer
Personal_Log = DAQ Manager, Tamer

[ALPACA_Optimiser]
ALPACA_Path = "C:\ALPACA\python-analyses"
optimiser-with-training-beta = True
optimiser-history-file-path = "C:\TALOS Log Files\OPTIMISER\training-history.csv"

[AEgIS DAQ]
DAQ_Bomb_Fuse_s = 45
GXML_DLL_Debug = False
DAQ_Data_Address =
GXML_Native_LV = False
SCP_Timeout_s = -1
;If the SCP Timeout is set to -1, the timeout is not set at all

[Simple device]
Simple-device-temp-check_s = 0.1
```

II. Using CIRCUS

1. [Launching CIRCUS](#)
 - a. [As distributed system](#)
 - b. [As stand-alone Guardian](#)
2. [Guardian overview](#)
3. [TALOS palette and useful functions](#)

Launching CIRCUS (Guardian)

There are two options for starting the CIRCUS:

- As a distributed system
- As a standalone Guardian (usually used for testing)

Launching CIRCUS (Guardian)

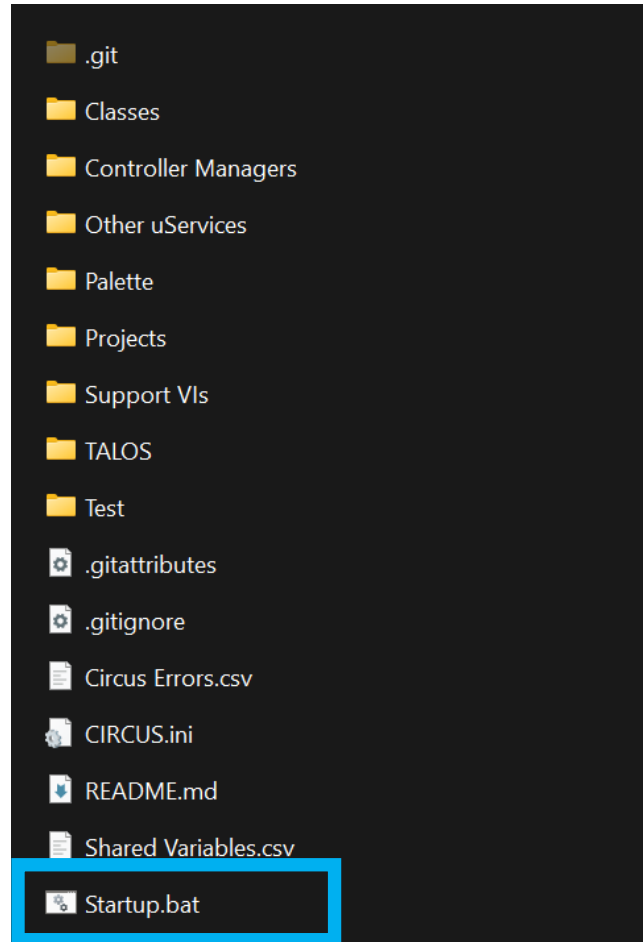
As a distributed system

Starting CIRCUS as a distributed system requires:

- Project with this PC's name (without errors)
- This PC's name in the PC_List of the config file

To start simple double-click the Startup.bat file

It is recommended to open the LabView project of the PC beforehand, however, Startup.bat can take care of it



Note: system started as part of the distributed system expects all PCs from the PC_List to be online. If one of the PCs won't start before Watchdog time, the system will go into Safe Mode (via the ABORT).

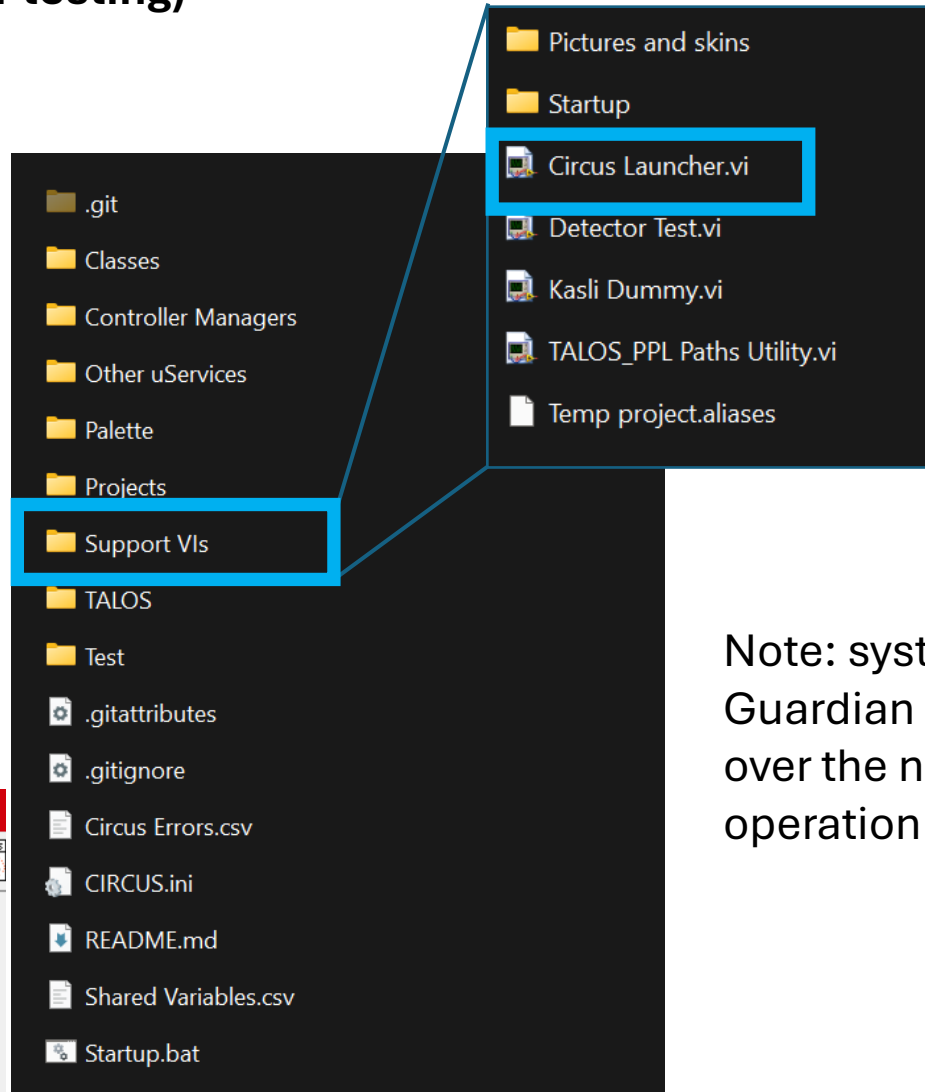
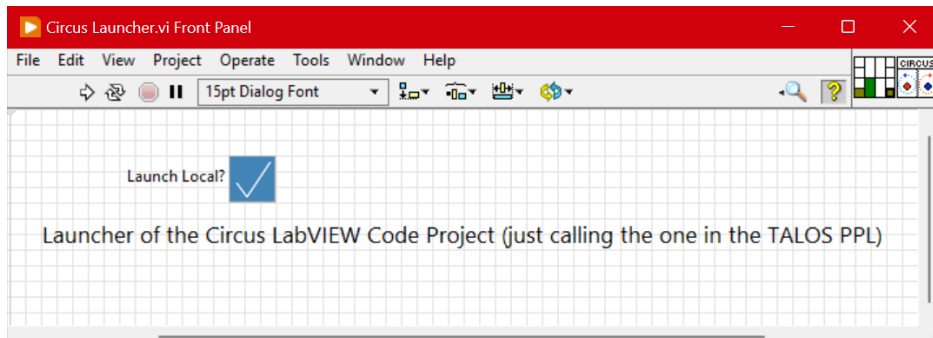
Launching CIRCUS (Guardian)

As a standalone Guardian (usually used for testing)

Starting CIRCUS as a standalone Guardian requires:

- A LabView project opened (the name doesn't matter in this case, however, there cannot be errors in the project)

To start navigate to the *Support Vis* directory and open the *Circus Launcher.vi*. Run the VI with the *Launch Local?* ticked on.



Note: system started as a standalone Guardian doesn't support messages over the network. You can still test operation of different uServices

Guardian Overview

The interface is divided into several sections:

- Guardians List:** A table showing the status of Guardians. It includes columns for ID, Name, and Status. The first row shows a Guardian with ID 09:57:16.444 and Name 07/11/2024.
- uServices Online:** A table showing the status of uServices. It includes columns for Name, ID, and Status. The first row shows a uService with Name DAQ Sender of local and ID 09:57:18.417.
- local Subpanel:** A central panel with a red and white striped tent background. It contains a list of uServices running on the current Guardian.
- Errors:** A panel showing a list of errors. It includes a "Dismiss All Errors" button.
- ABORT indicator:** A panel with a red "STOP" button and a "RUN" button. It also includes a "Service to show" dropdown menu.
- uService view selector:** A panel with a "uService" input field and an "overwrite name" input field. It includes "Start uService" and "Stop uService" buttons.
- Buttons:** A series of buttons for system control: "Start online checks", "Stop online checks", "Send Pat to all GDs", "List the uServices available for launching", "Reset ABORT", "Reload Config files", and "Update from GIT".
- Kasli Log:** A panel at the bottom right showing a log of events.

Annotations and their corresponding elements:

- List of PCs in the system (currently running as standalone Guardian):** Points to the "Guardians List" table.
- List of uServices running on the current Guardian:** Points to the "uServices Online" table.
- Button to stop the system (press ctrl to lift up the cover):** Points to the "STOP" button in the "ABORT indicator" panel.
- List of errors:** Points to the "Errors" panel.
- ABORT indicator:** Points to the "ABORT" button in the "ABORT indicator" panel.
- uService view selector:** Points to the "uService" input field in the "uService view selector" panel.
- Clear list of errors:** Points to the "Dismiss All Errors" button.
- Reset ABORT flag and stop the Safe Mode:** Points to the "Reset ABORT" button.
- Load config files of CIRCUS and TALOS:** Points to the "Reload Config files" button.

TALOS palette and useful functions

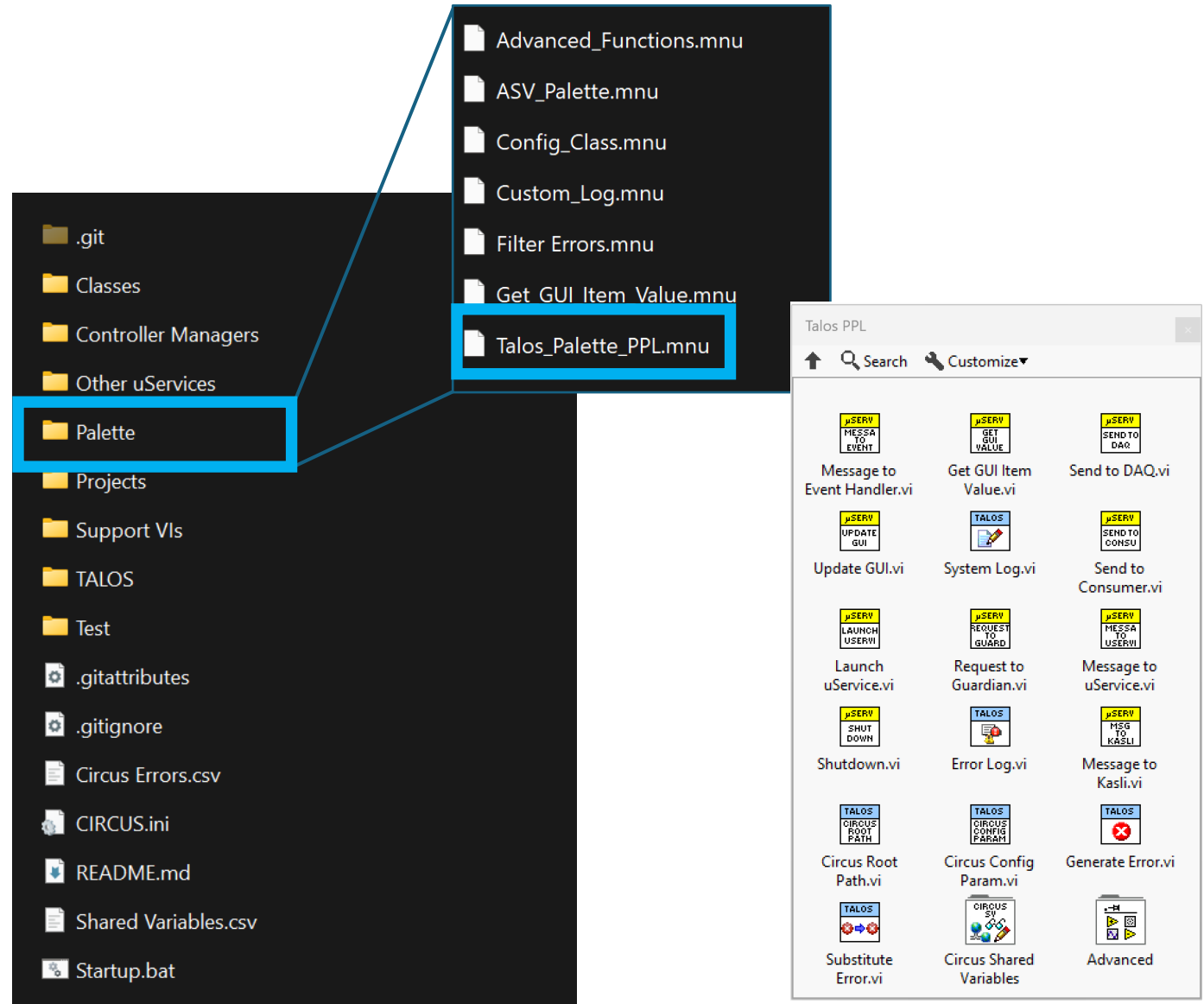
TALOS palette

TALOS comes in with a set of functions as a palette that can be added to the installation of LabView

TALOS comes in with a set of functions as a palette that can be added to the installation of LabView









To add palette to LabView:






1. Open LabView
2. Tools > Advanced > Edit Palette Set...
3. Right click on the *Functions* palette
4. Insert > Subpalette
5. Link to an existing palette file (.mnu)
6. Select the Talos_Palette_PPL.mnu file



TALOS palette

Some functions that are available

 Send to Consumer.vi	Add the case to the consumer loop.	Father of all uServices\Father of all uServices\Children Methods\Send to Consumer.vi
 Message to uService.vi	Send a message to the uS (any uS that is online in the system).	Father of all uServices\Father of all uServices\Children Methods\Message to uService.vi
 Infinite Message Manager.vi	Infinite Message is a continuous case enqueued in the consumer ever specified period (from the CIRCUS parameters).	Father of all uServices\Father of all uServices\Children Methods\Infinite Message Manager.vi
 Update GUI.vi	Send data to the control on the uS GUI element (type of the data must much the type of the control).	Father of all uServices\Father of all uServices\Children Methods\Update GUI.vi
 Get GUI Item Ref.vi	Get Reference of the control from the current uS GUI.	Father of all uServices\Father of all uServices\Children Methods\Get GUI Item Ref.vi
 Get GUI Item Value.vi	Get data from the control of the uS GUI (can return standard types like double, string, integer, or variants).	Father of all uServices\Father of all uServices\Children Methods\Get GUI Item Value.vi
 Update GUI - Append ...	Update string control on the uS GUI by appending data to the current value.	Father of all uServices\Father of all uServices\Children Methods\Update GUI - Append String.vi
 Update GUI - Waveform ...	Update data by adding a next point to the Waveform Graph on the uS GUI.	Father of all uServices\Father of all uServices\Children Methods\Update GUI - Waveform Chart.vi

 Circus Config Param.vi	Get parameter from the CIRCUS config file.	Support VI\Params\Circus Config Param.vi
 Substitute Error.vi	Replace the error with the specified error from the CIRCUS error file.	Support VI\Error Generation\Substitute Error.vi
 Generate Error.vi	Create error with the specified error code from the CIRCUS error file.	Support VI\Error Generation\Generate Error.vi
 String to Bool.vi	Convert string to Boolean.	Support VI\Misc\String to Bool.vi
 Temporised Pop-up ...	Open a window with a message for a specified duration.	Support VI\Misc\Temporised Pop-up Window.vi

If the palette doesn't work, you can access all the VIs from the palette directly from the library (open the library and navigate to the specified paths in the library).

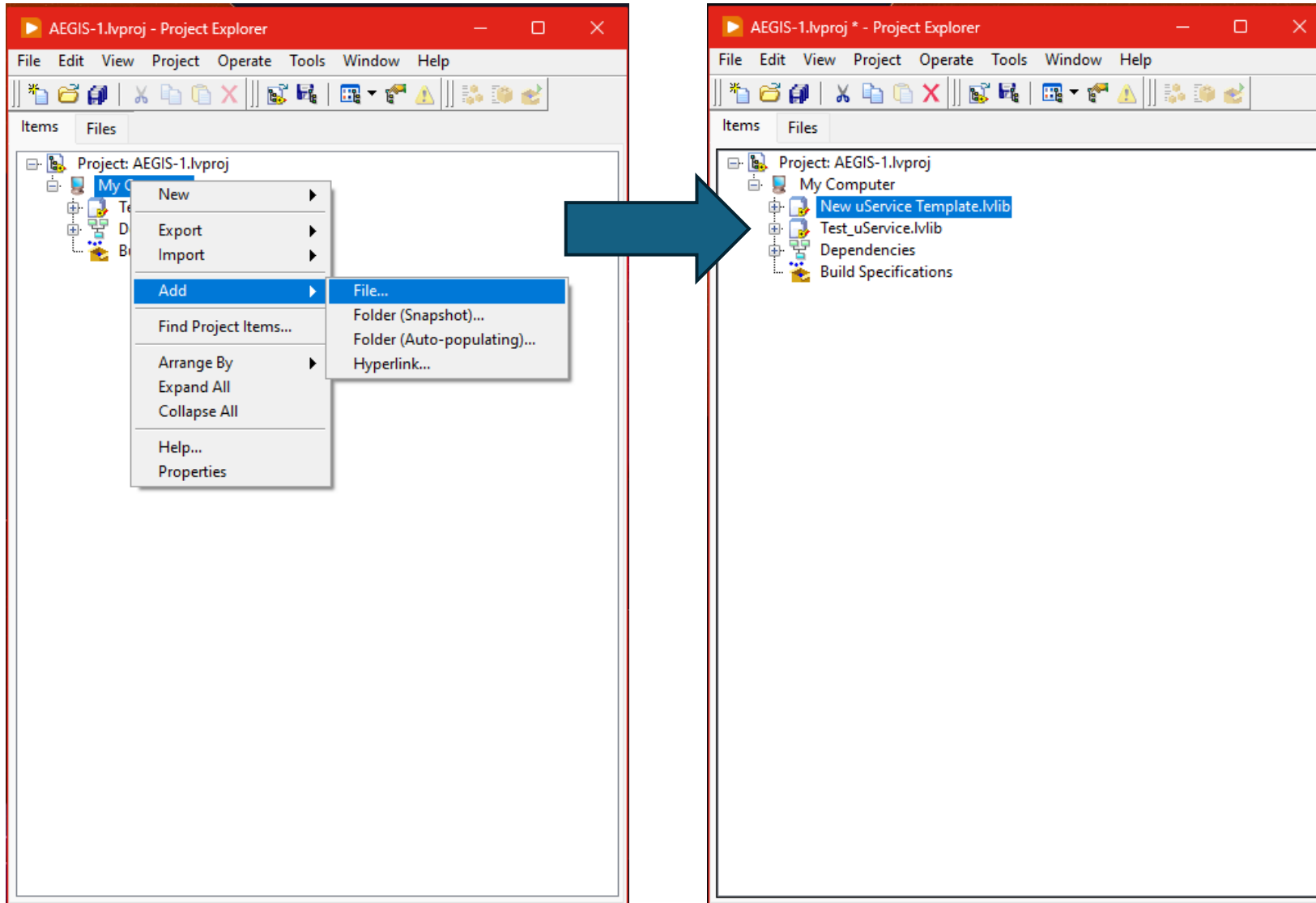
III. Making a new uService

1. [Creating from the template](#)
 - a. [Hardware class](#)
 - b. [Detector class](#)
2. [Example](#)
 - a. [Step by step simple chat uService](#)
 - b. [Starting uService](#)
 - c. [Using uService](#)

Making a uService

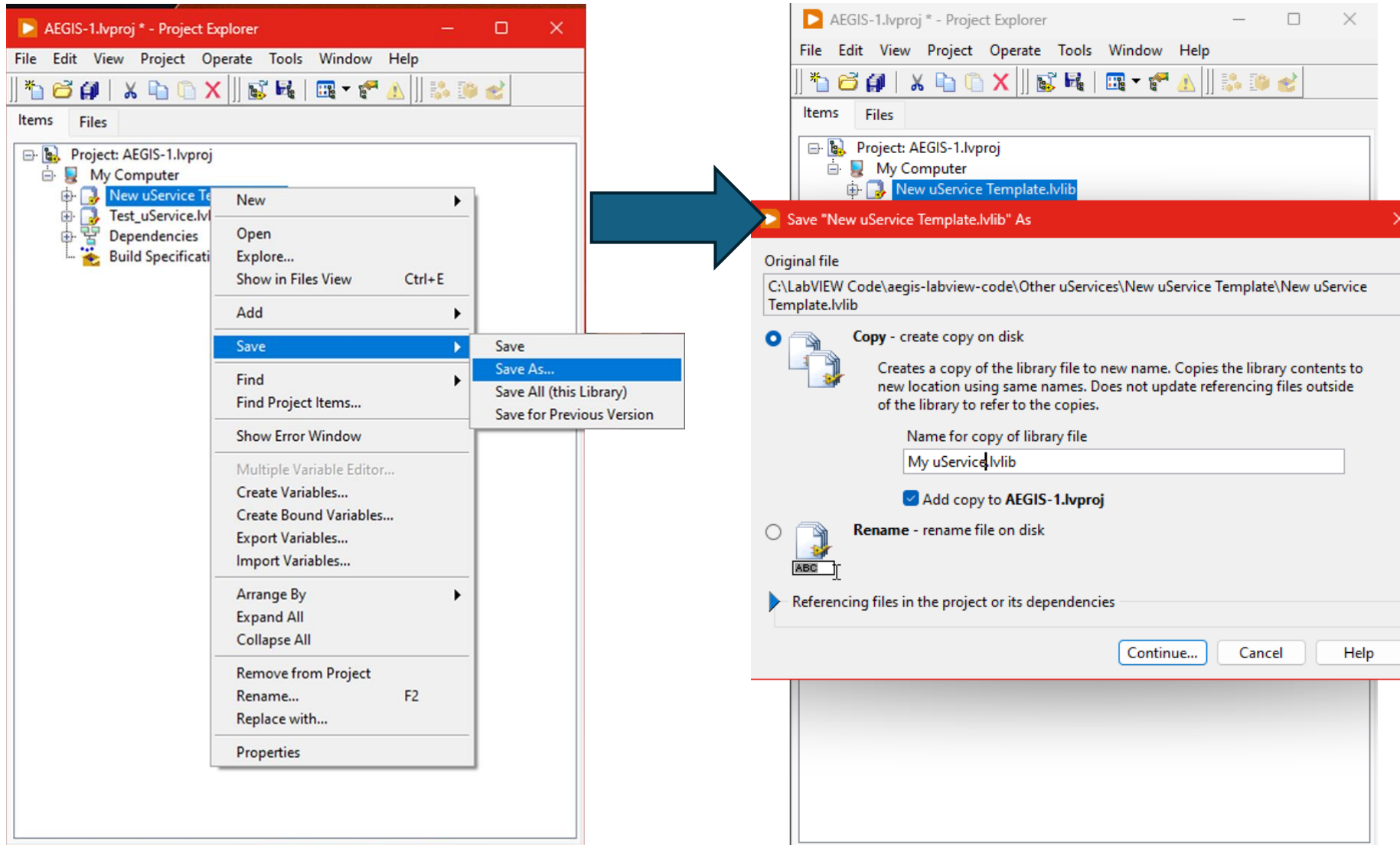
Based on the template provided

Add the New uService Template to the project



1. Right-click on the *My Computer* in the LV's Project Explorer and select *Add->File...*
2. Inside the directory of the repository (aegis-labview-code) select *Other uServices>New uService Template>New uService Template.lvlib*
 - You want to add the entire library, not only the class

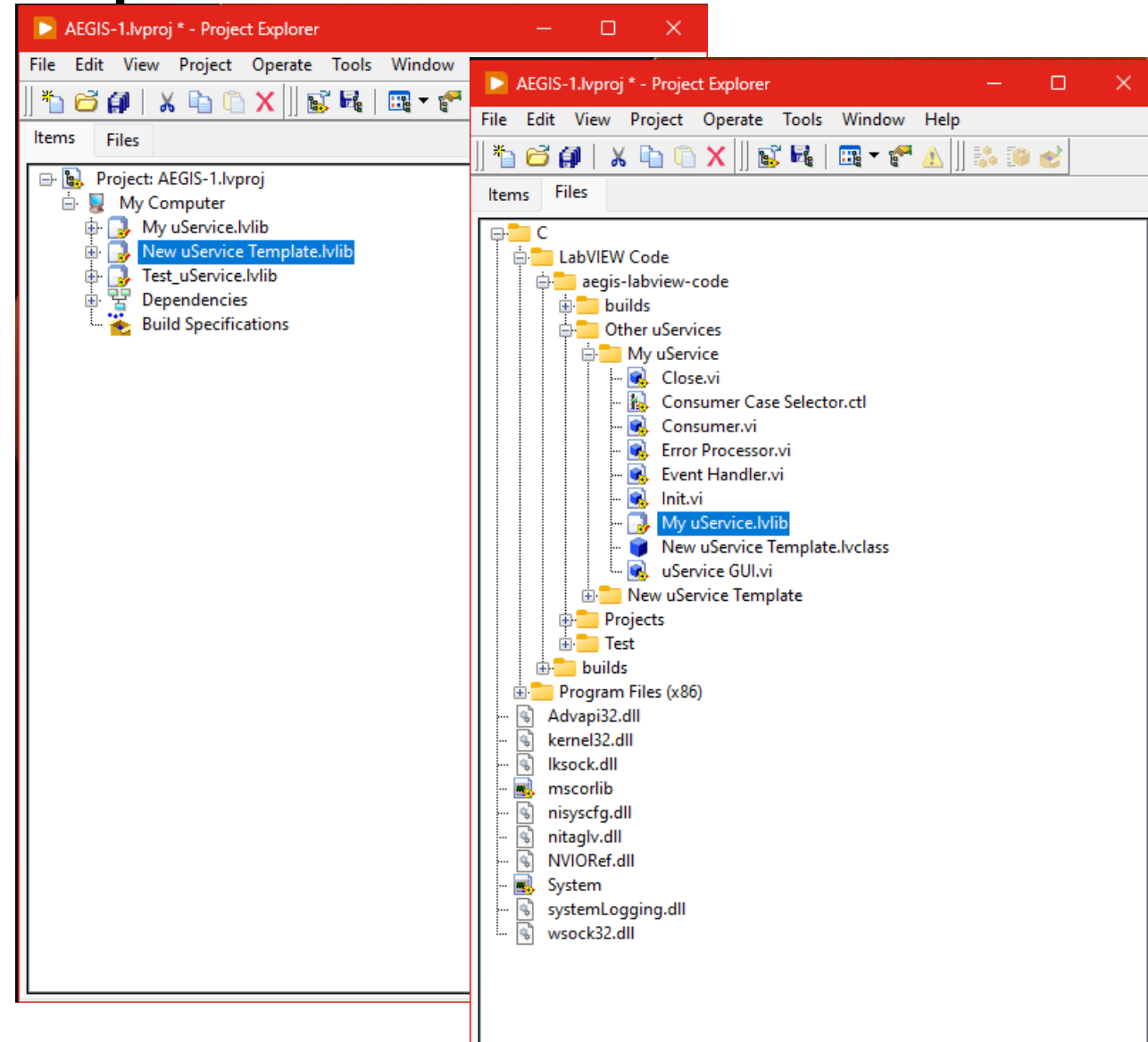
Create a copy of the template



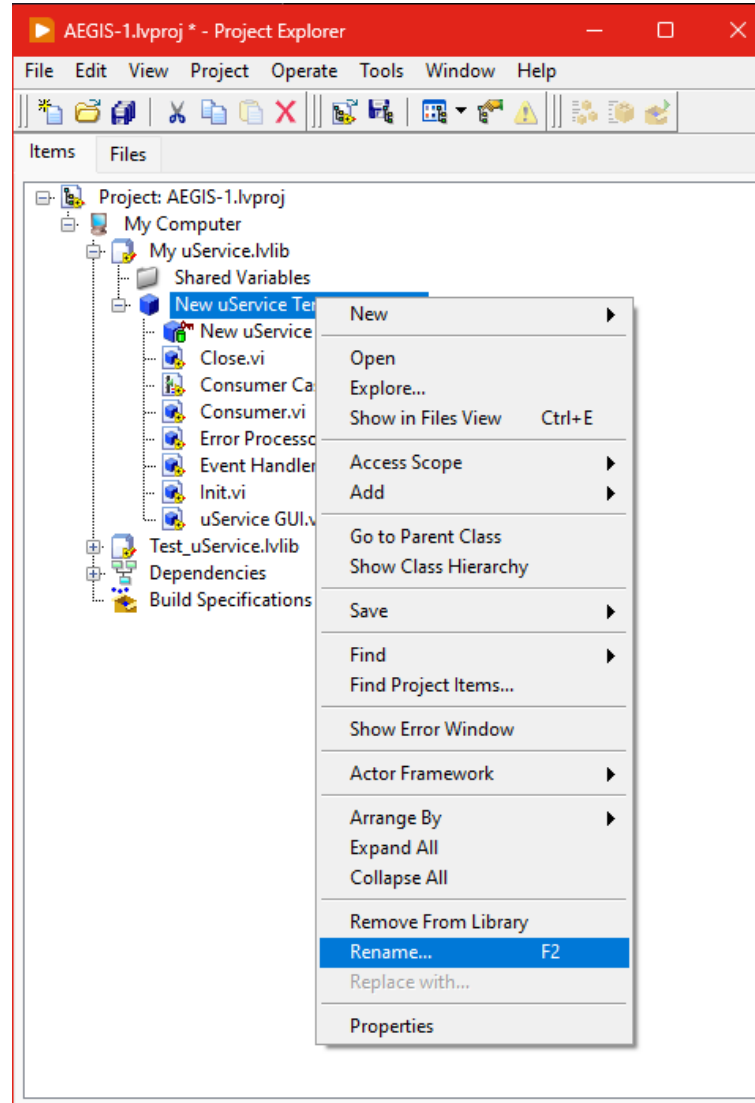
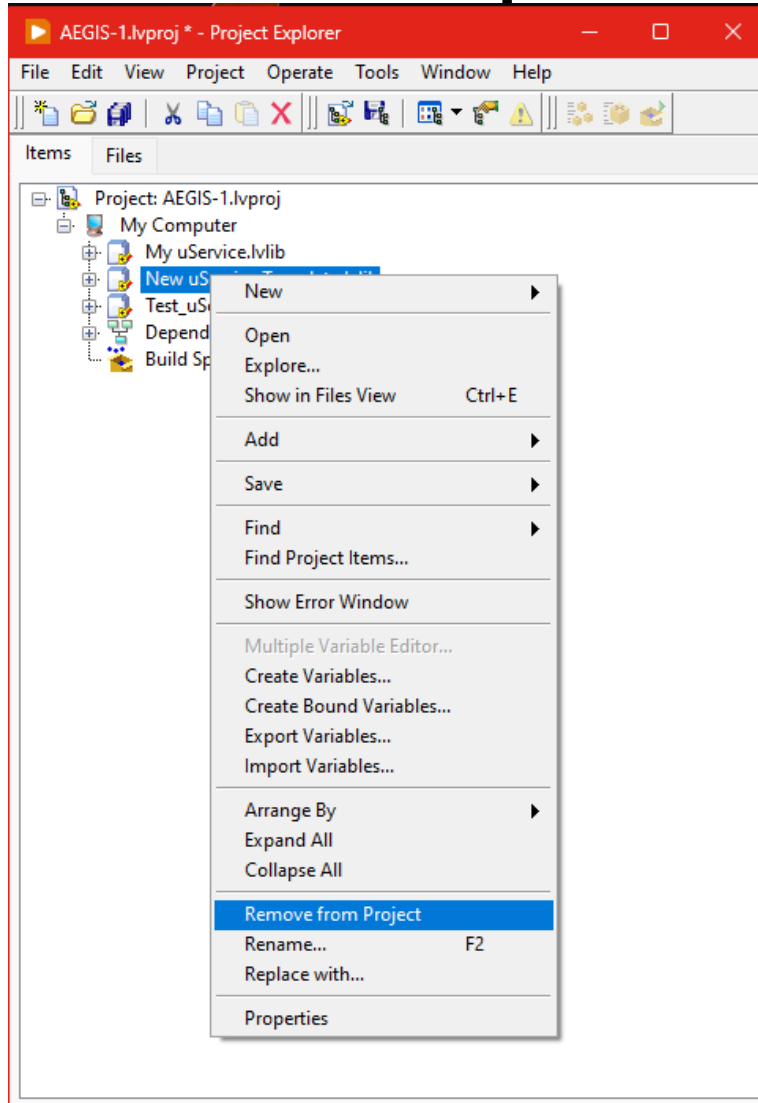
3. Right-click on *New uService Template.lvlib* -> *Save* -> *Save as...*
4. Baptise it as you wish (*My uService.lvlib* in the example on the slide)
 - Leave the flag *Add copy to <project name>.lvproj* ticked. This will add the new library to the project you are working in.
5. Press continue

Create a copy of the template

6. Browse for the folder where you want to save your new uService. Some general rules:
 - *Controller Managers* folder -> uServices that control controllers (actuators, electro gun, Rotators, Temperature, etc.)
 - *Detector Managers* folder -> uServices that control detectors (oscilloscopes, cameras etc.)
 - *Other uServices* -> anything else (logging, messaging system, etc.)
 - The name of the folder for your library should be the same as the library (LabView will add *Folder* at the end; you can remove it to make it clean)
7. Press *Save*, it will create a new folder with the specified name and copy all the VIs inside

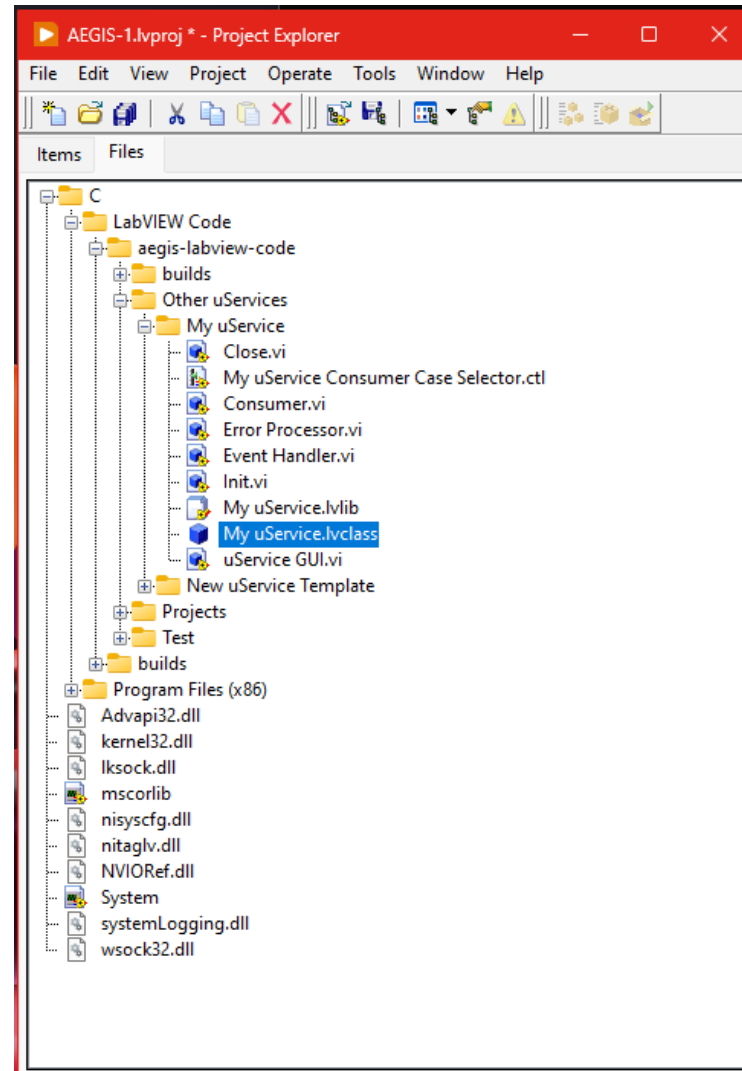
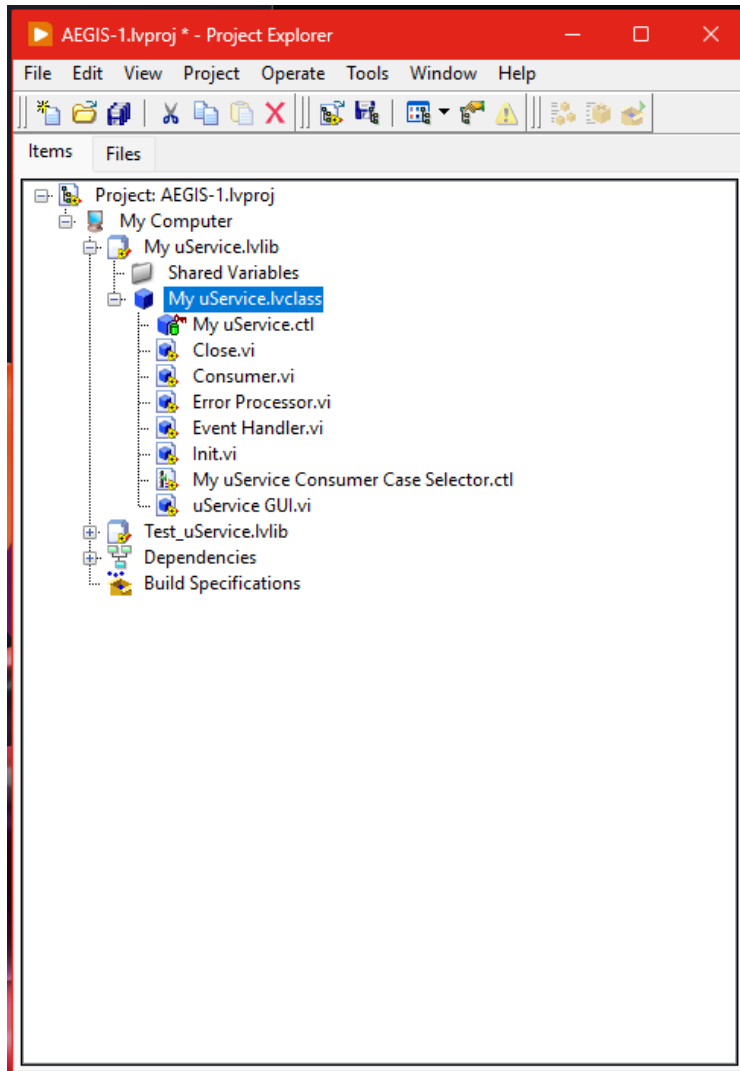


Clean up



8. Remove the template
 1. Right-click on the *New uService Template.lvlib*
 2. Select *Remove from Project*
 3. Press *OK*
9. Rename the uService class
 1. Open the freshly created library
 2. Right-click on the *New uService Template.lvclass*
 3. Select *Rename*
 4. Give it the same name as the library
 5. Rename also the *Consumer Case Selector* by appending uService name in the front
 6. You can also edit the class's icon. It is recommended to at least set the *Text icon* (This can be accessed by right-clicking the class and selecting *Properties*)

Finish

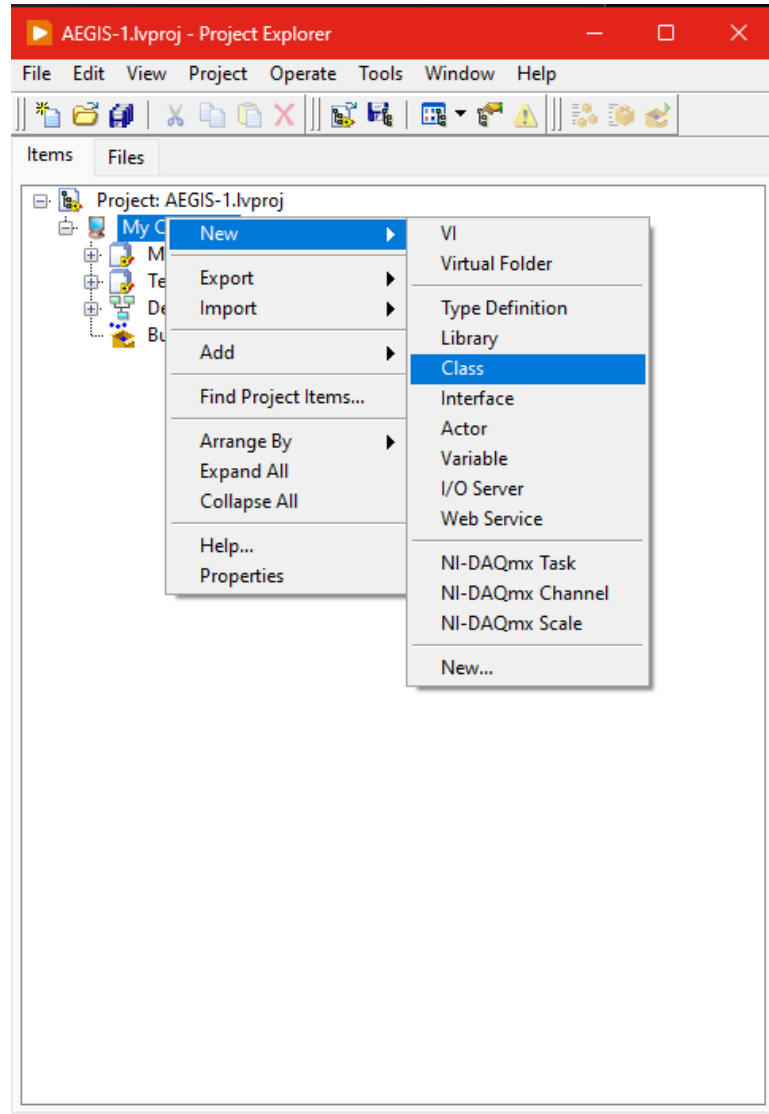


Your project should look like this.

Hardware and Detector class

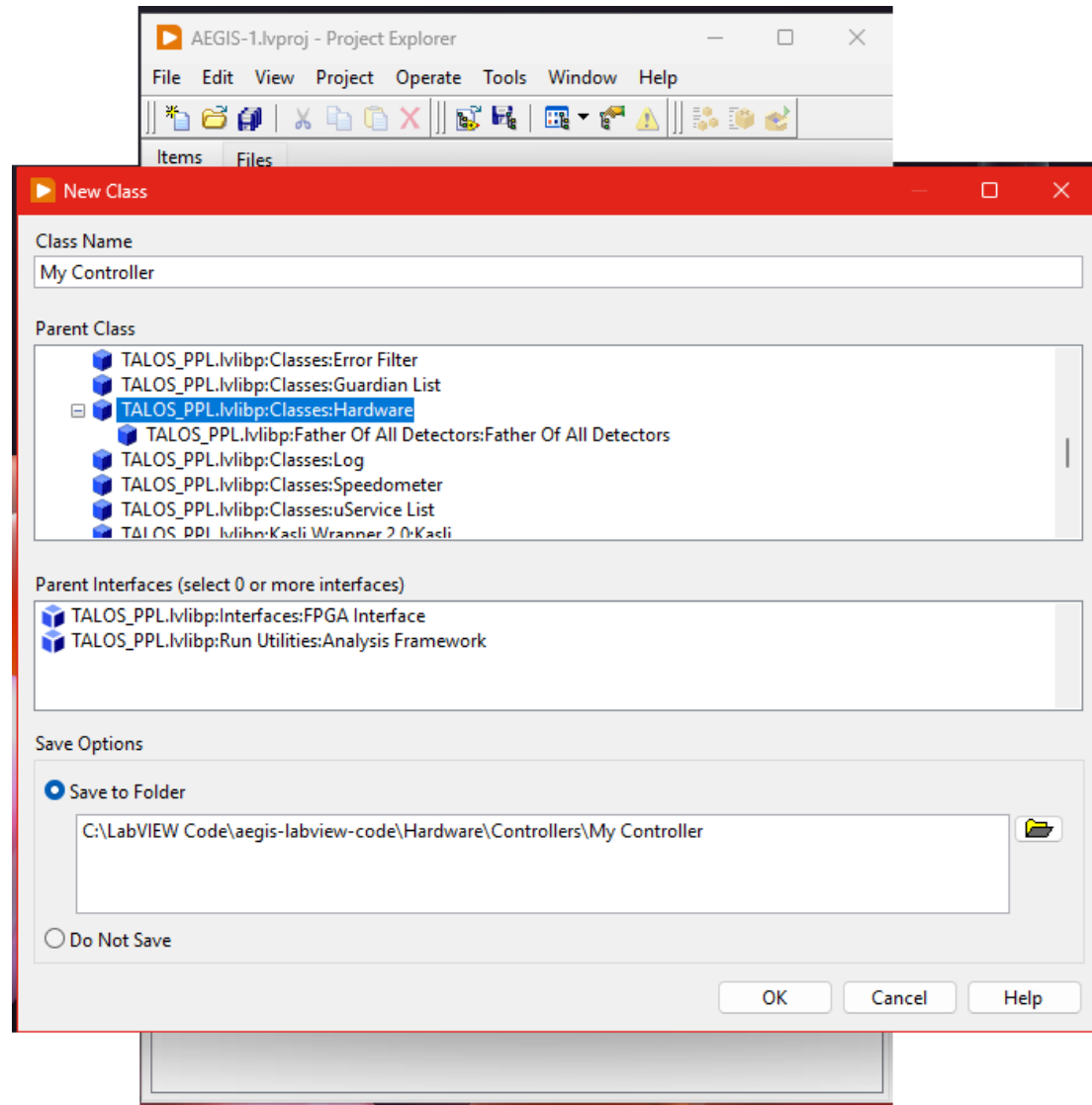
If you want to create a uService for a controller or detector with some new hardware, you will need to create a hardware class (Detector class is a child of the Hardware class)

Hardware/Detector class



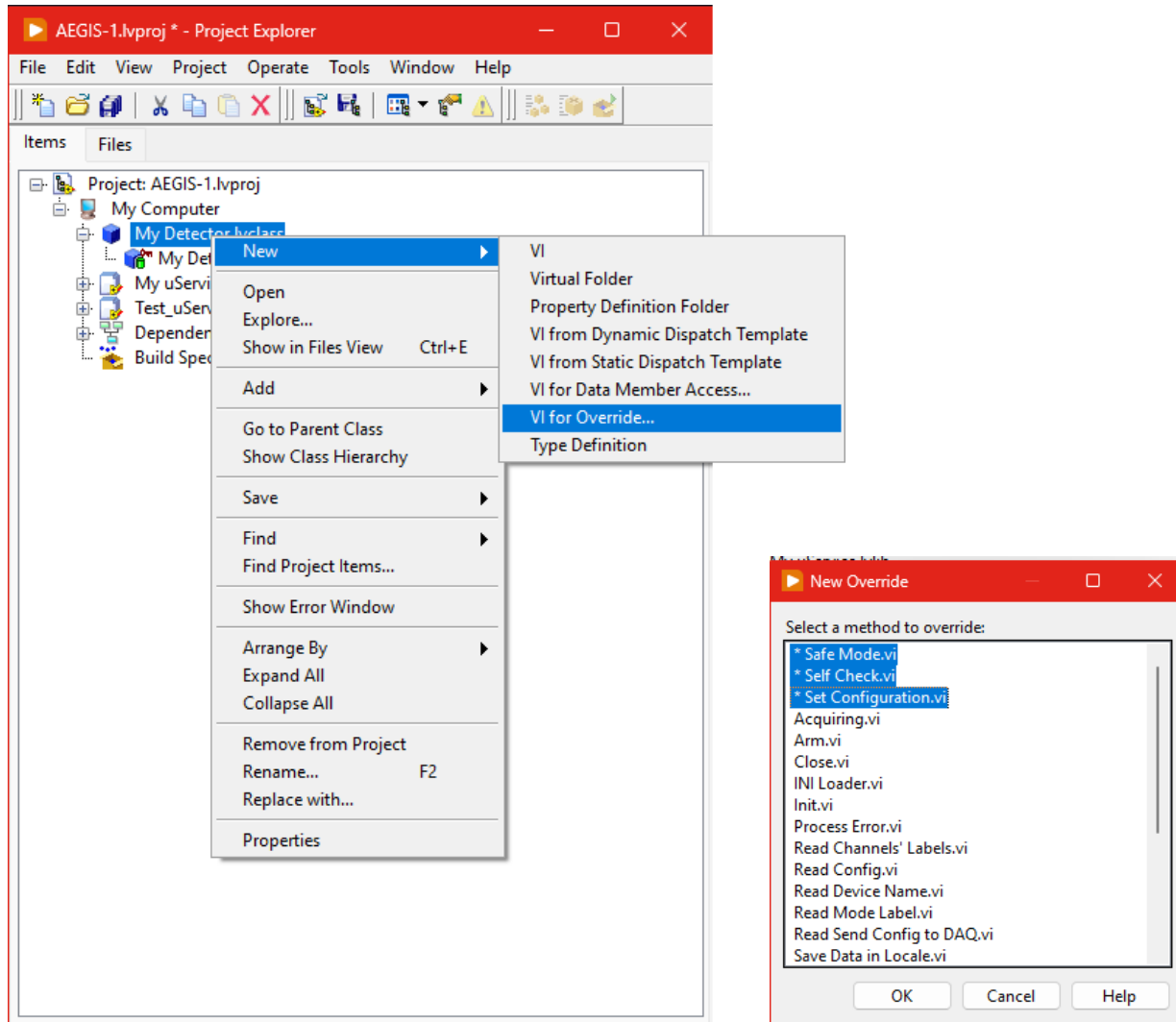
1. Right-click on the *My Computer* in the LV's Project Explorer and select *Add->Class*

Hardware class



2. Give a new name to the class, usually it will be the name of the controller
 - This class is a general class for operating a specific controller. The same controller can be used by different uServices
3. From the *Parent Class* scroll down and select *TALOS_PPL.lvlib:Classes:Hardware* (if you are implementing a hardware class for a detector, select the entry below *TALOS_PPL.lvlib:Father of All Detectors:Father of All Detectors*)
4. Tick the *Save to Folder* and then press the browse button to the right. Navigate to the *Hardware>Controllers* (or *Detectors*) folder in the project, type the name of your class and press save
5. Press Ok and your new class is created
6. Fill in the methods for the class by creating new VI's (preferably from the *Dynamic Dispatch Template*)
 - This methods should be general ones for the device: init, close, set parameter, move (depending on the device of course)

Detector class



7. Detector class is used inside Detector Manager and it needs some methods to be overwritten
 1. Rick-click on the detector class created
 2. Select *New->VI for Override...*
 3. Select all Vis with the * and press OK
 4. Implement the functionality according to your desires

Detector Manager uService

Detector Manager is a special uService class that can be used to define new uService for a detector quickly.

You can create it with the same scheme as a new class, but selecting *TALOS_PPL.lvlibp:Detector Manager:Detector Manager* as the parent instead.

Then there are 2 Vis that should be overwritten:

- **Init.vi** – here a new instance of the detector class should be created and saved inside the private data cluster of the class
- **SubPanel.vi** – this will be the GUI inserted into the main the uService

Example uService development

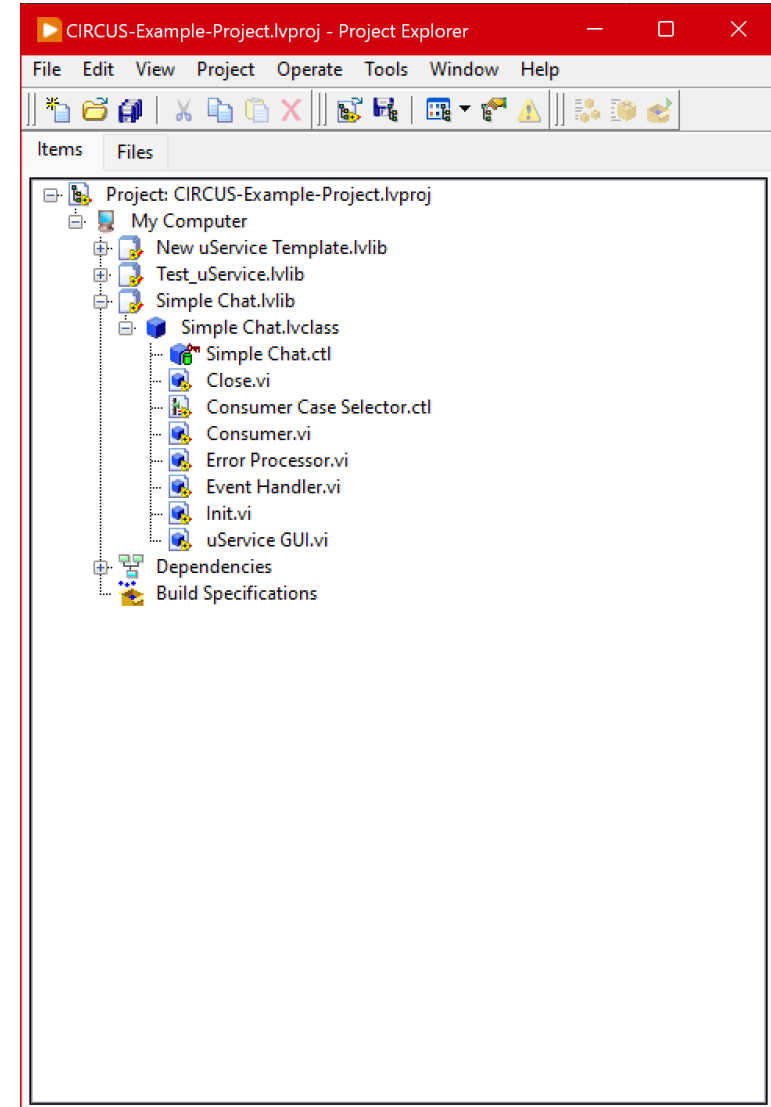
Making a simple chat uService

Simple chat

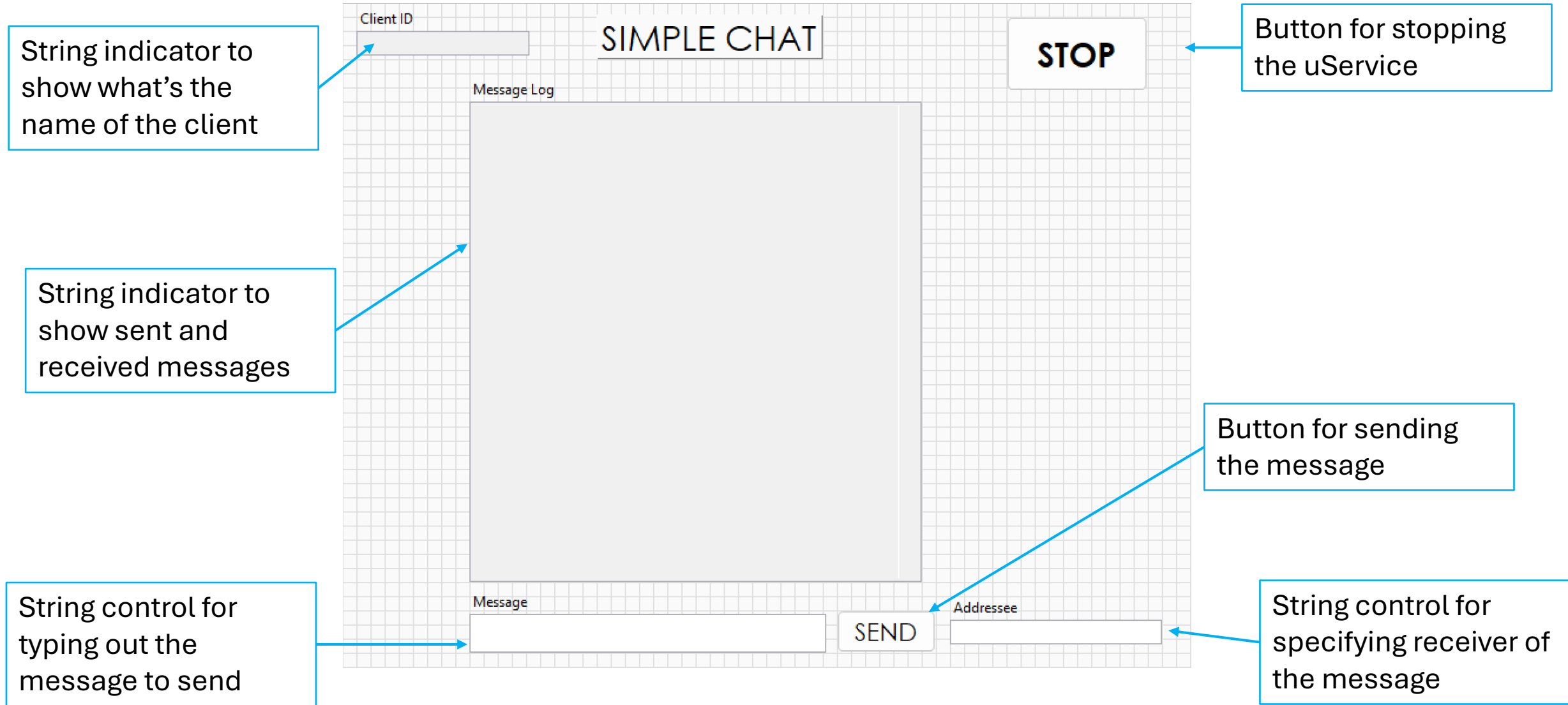
The idea is to create a simple uService that can be used as a chat.

The uService should support sending and receiving a message between other chat participants.

We are going to start with a uService from the template and we will call it “Simple Chat”



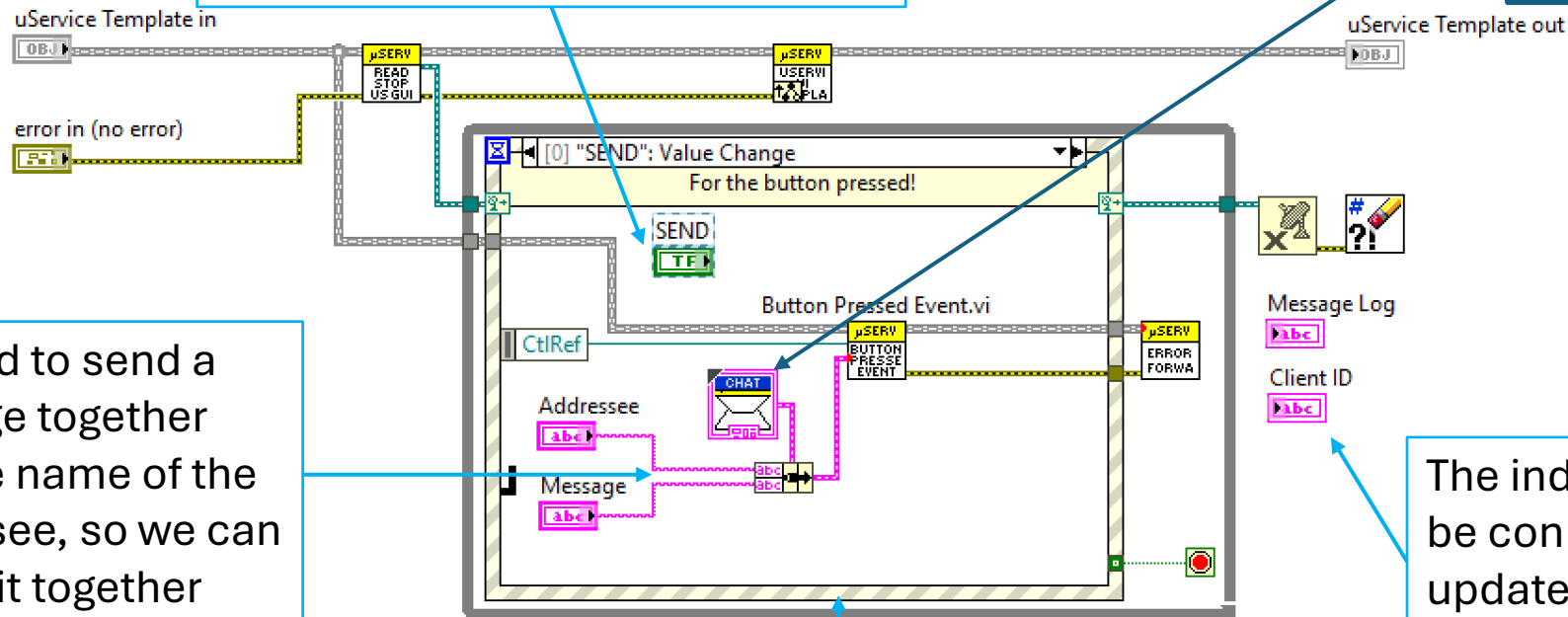
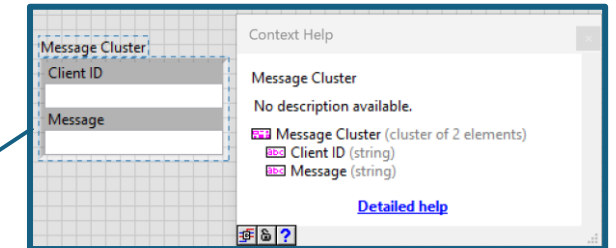
Simple chat – GUI (from panel)



Simple chat – GUI (block diagram)

Create a custom type definition that we can use throughout the uService (save it inside the class)

Button for sending the message.
Having the button inside the
case always to use latching
mechanism



We need to send a message together with the name of the addressee, so we can bundle it together

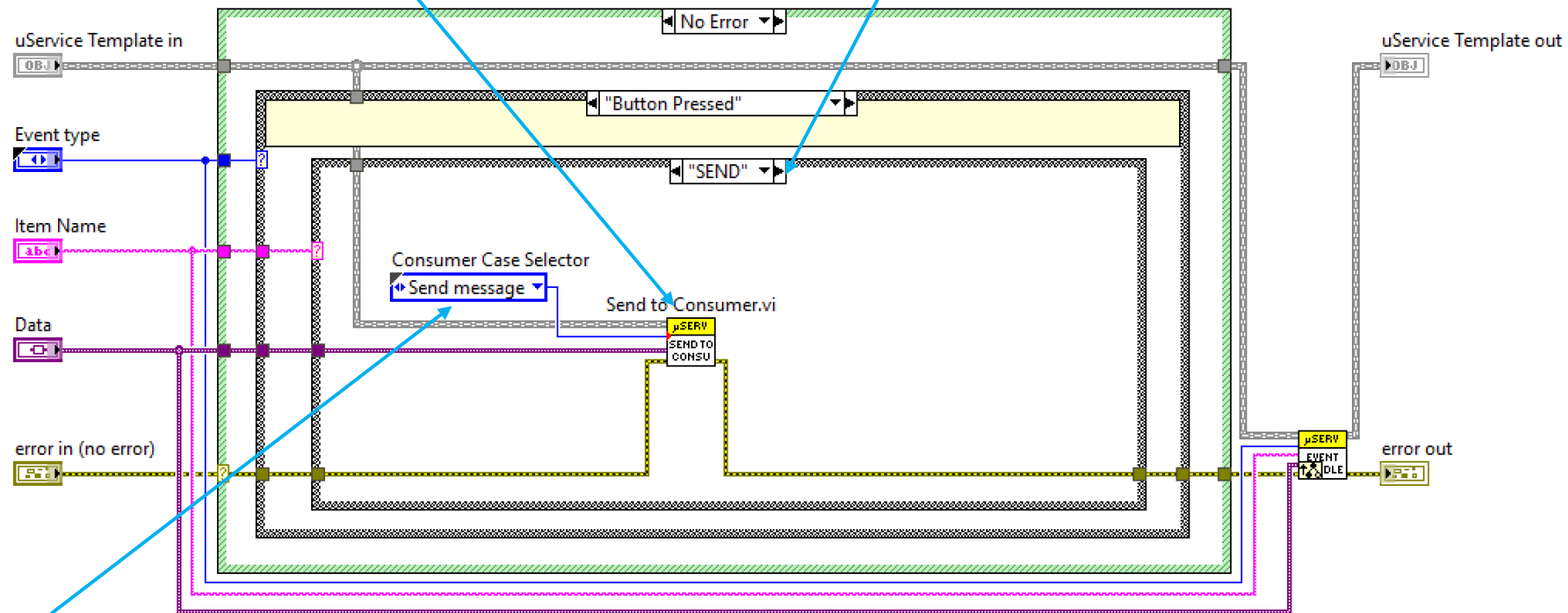
The indicators don't need to be connected. They will be updated throughout the uService running

In this uService we only have one simple button, so only one custom case needed

Simple chat – Add button case in the Event Handler.vi

In this event we want to call a specific case in the consumer

We have a single button calling the *Button Pressed Event* in the GUI.vi. This means we need to create a case for that button with the case name the same as the button label



Options for the Consumer Cases are declared with the Consumer Case Selector Enumerator

We add a new consumer case by editing the items in the type definition

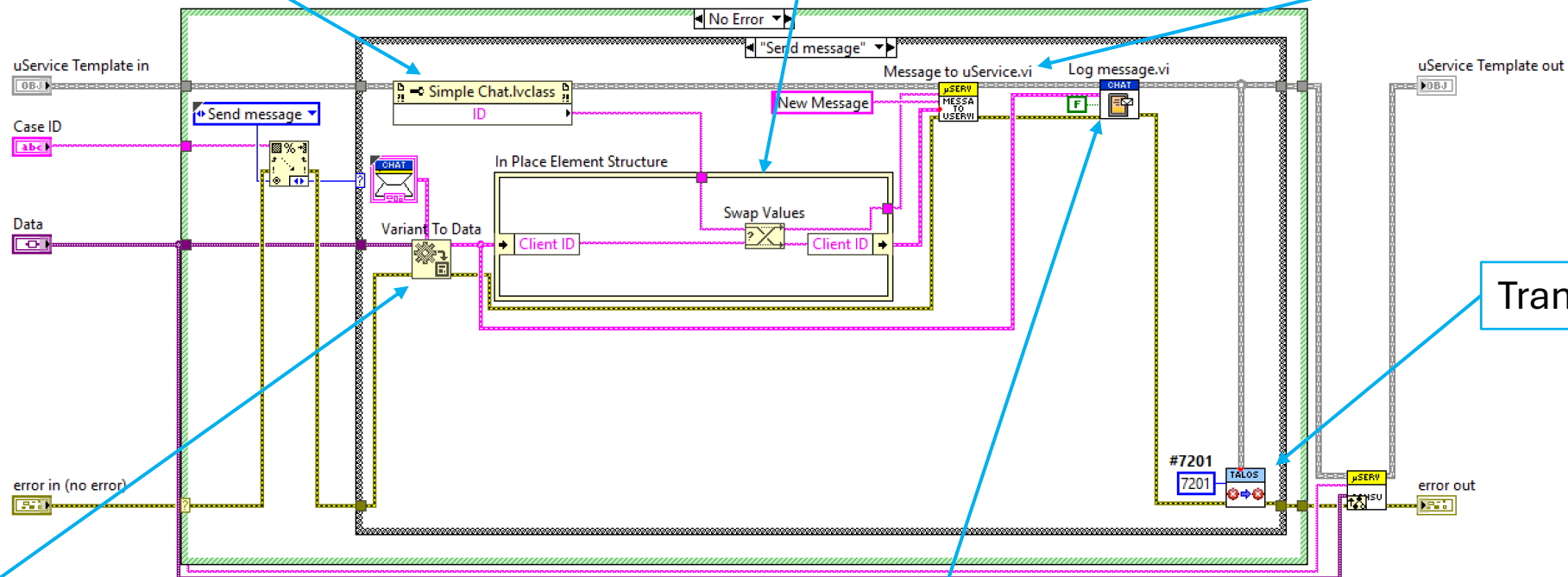
Simple chat – Define the case in the Consumer.vi

We add a new case for sending the message

We are going to use the uService ID as the Client ID

We need to convert the message so we send information about who sent the message

Send a message “New Message” to the uService specified from the message cluster



Translate error

We need to convert variant to the message type (by using the type def)

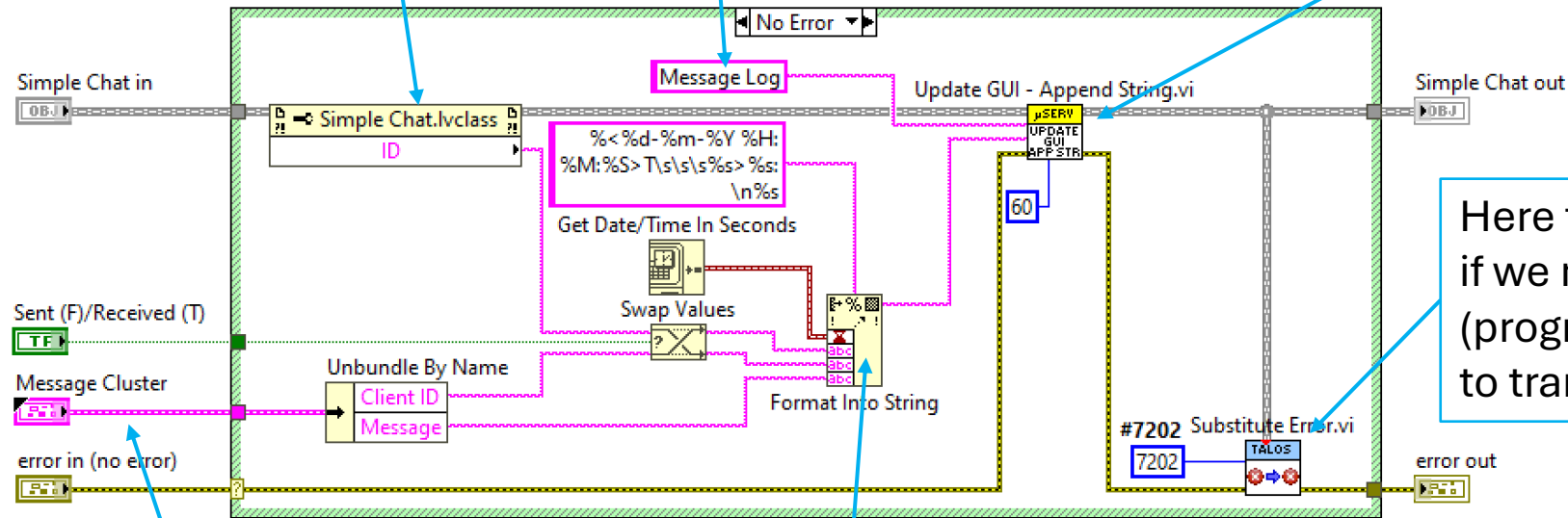
Print message on the GUI. For this we can create a SubVI as we will use it also in received message. This can also be done as a consumer case

Simple chat – SubVI for printing message on the GUI

We are using the
uService ID as the
Client ID of this
uService

Name of the indicator that is used for printing the messages

Update the GUI with the Append String VI to save the previous messages



Here the error can happen only if we made a mistake (programmer's error). It is safer to translate it anyway

We are going to use the same message type def as the input for the VI

Format message for printing

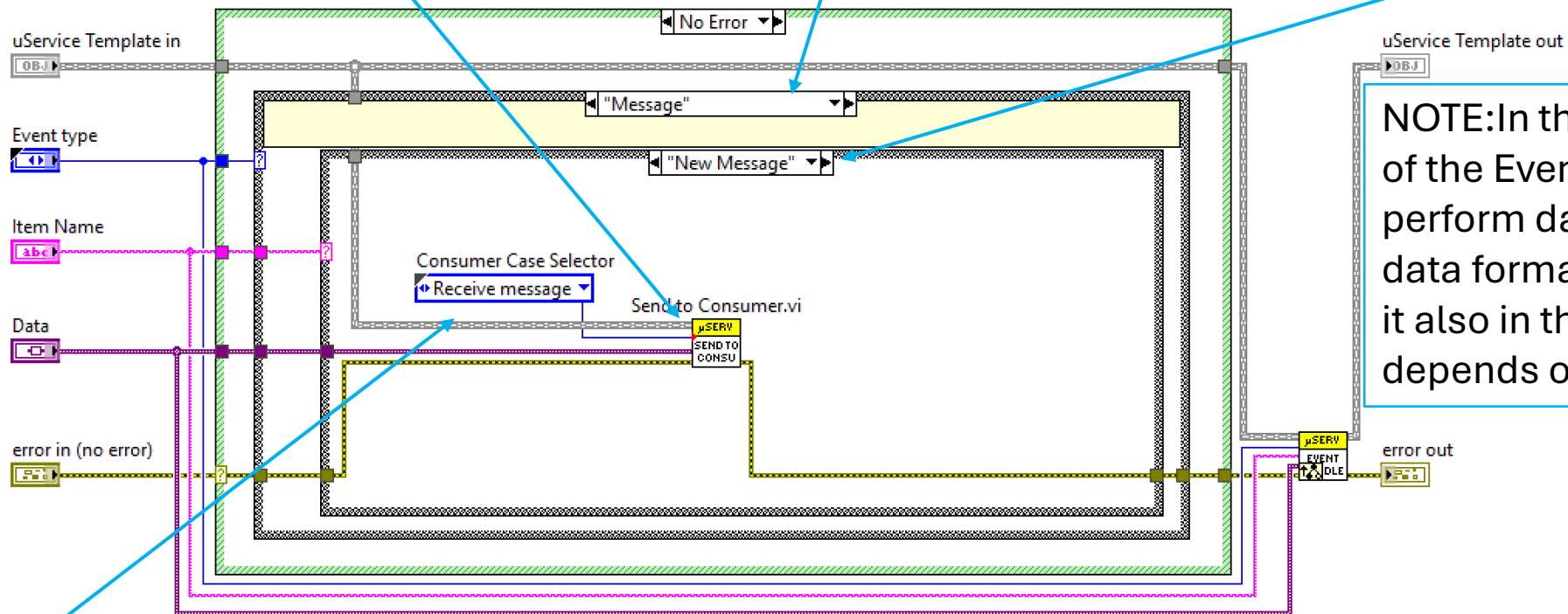
We add a new case by editing the items in the type definition

Simple chat – Add message case in the Event Handler.vi

In this event we want to call a specific case in the consumer

In the message case of the Event Handler we can add cases for all messages that we expect the uService to receive.

The case name should be the same as the message sent to the uService



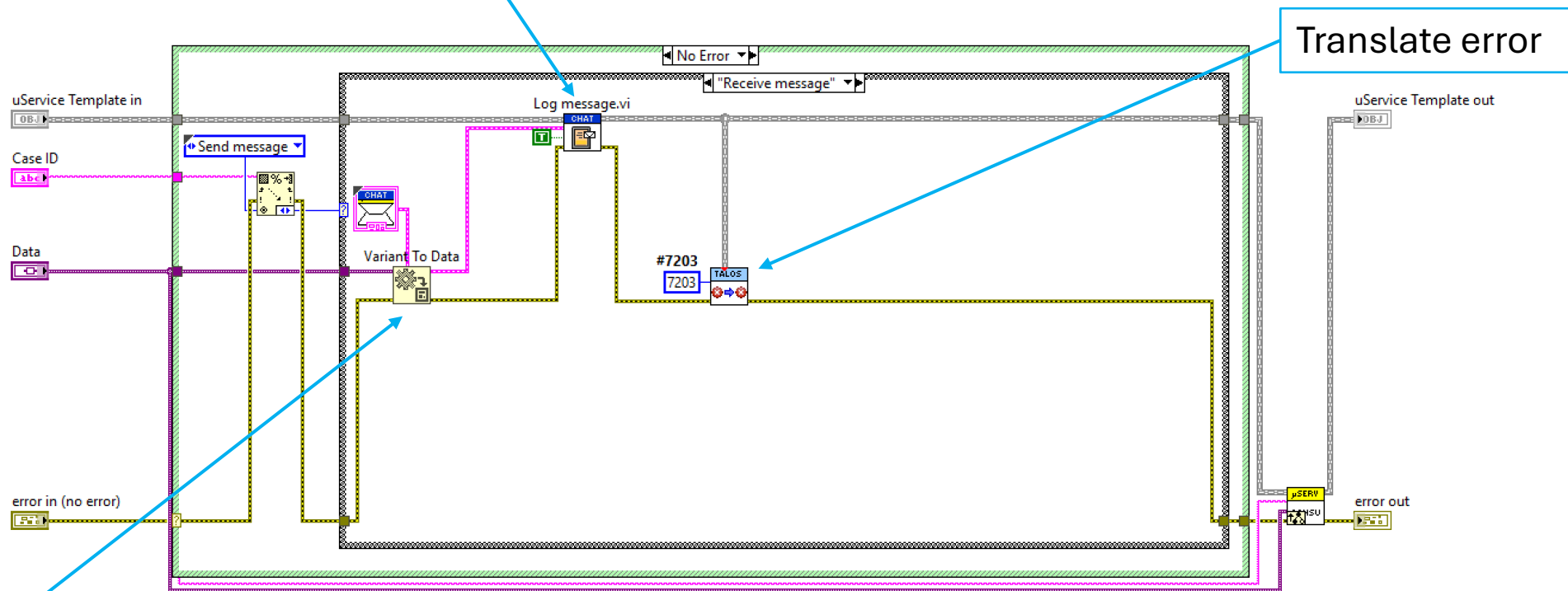
NOTE: In the message cases of the Event Handler we can perform data checking or data formatting. We can do it also in the Consumer. It depends on our needs.

Options for the Consumer Cases are declared with the Consumer Case Selector Enumerator

We add a new consumer case by editing the items in the type definition

Simple chat – Define the case in the Consumer.vi

Receive message case has only to print message on the GUI. Here we can use our SubVI again

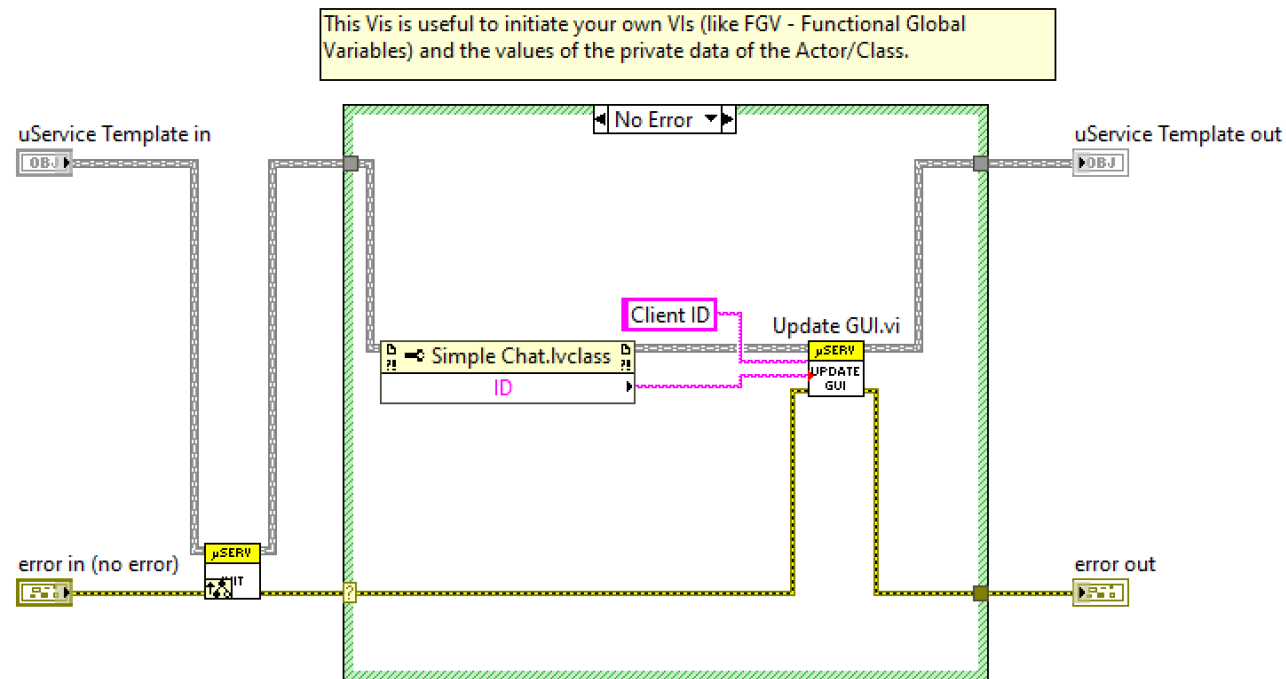


Translate error

We need to convert variant to the message type (by using the type def)

Simple chat – Init.vi

In the Init.vi we only want to upda the indicator with the client ID



Simple chat – update error list

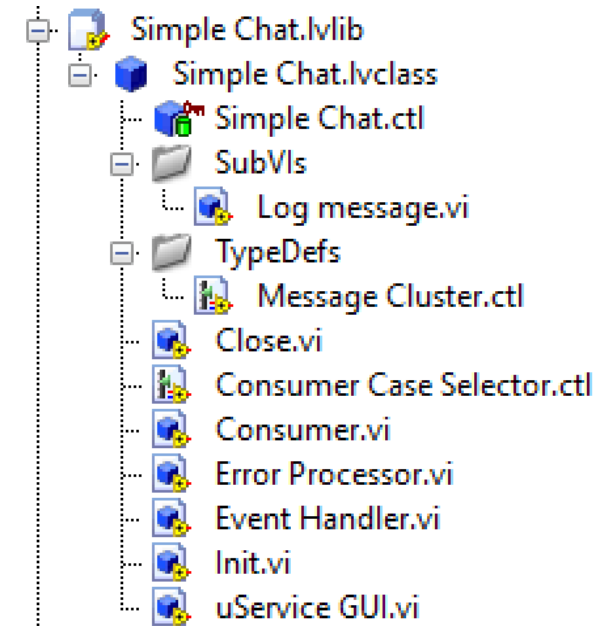
Error list should be updated throughout the development of the code

We decided to start the Simple chat at the error code 7200

We defined three custom error codes for this uService. These codes are saved in the CIRCUS error file.

```
7200,Simple Chat,0
7201,Error sending a message,0
7202,Error printing the message,0
7203,Error decoding the message,0
```

Simple chat – uService library



Simple chat – start the uService

The screenshot displays the CIRCUS+TALOS interface. At the top, there are two panels: 'Guardians List' (Size: 1) and 'uServices Online' (Size: 6). The 'Guardians List' shows a single entry for 'local' with a status of '22:58:23.655 07/11/2024'. The 'uServices Online' panel lists several services including 'DAQ Sender of local', 'Error Manager', and 'GUI of local', each with a status of '22:58:25.650 07/11/2024'. A central banner features a red and white striped tent with a sign that says 'CIRCUS'. To the right of the banner is an 'Errors' panel. Below the banner is a large 'local Subpanel' area. On the left side of the 'local Subpanel', there is a section for 'uService' with a dropdown menu set to 'Simple Chat', an 'overwrite name' field, and two buttons: 'Start uService' and 'Stop uService'. A blue arrow points from a text box to the 'Start uService' button. To the right of the 'uService' section are several buttons: 'Start online checks', 'Stop online checks', 'Send Pat to all GDs', 'List the uServices available for launching', 'Reset ABORT', 'Reload Config files', and 'Update from GIT'. A 'Dismiss All Errors' button is also present. On the far right, there is a 'Message received' panel and a 'Kasli Log' panel. At the bottom right, there is a 'STOP' button with the text 'PRESS CTRL TO ARM' above it.

local Subpanel

uService
Simple Chat

overwrite name

Start uService

Stop uService

Start online checks

Stop online checks

Send Pat to all GDs

List the uServices available for launching

Reset ABORT

Reload Config files

Update from GIT

Dismiss All Errors

Message received

Kasli Log

STOP

PRESS CTRL TO ARM

We can start a single instance of the uService by typing the uService name and pressing *Start uService* button

Simple chat – send message to itself

The screenshot displays the CIRCUS+TALOS interface with the following components:

- Guardians List (Size: 1):** Shows a single guardian named 'local' with a status of '23:03:39.081' and a date of '07/11/2024'.
- uServices Online (Size: 7):** Lists several services including 'DAQ Sender of local', 'Error Manager', and 'GUI of local', each with a status of '23:03:38.972' or '23:03:38.373' and a date of '07/11/2024'.
- Central Chat Window:** Titled 'SIMPLE CHAT', it contains a 'Message Log' showing two identical messages: '07-11-2024 23:03:29 Simple Chat> Simple Chat: Hello there'. Below the log is a 'Message' input field with 'Hello there' and an 'Addressee' dropdown menu set to 'Simple Chat'. A 'SEND' button is located between the input field and the dropdown.
- Client ID:** A label 'Client ID' with a dropdown menu showing 'Simple Chat'.
- STOP Button:** A large red button labeled 'STOP' with the text 'PRESS CTRL TO ARM' above it.
- Errors:** A section on the right with a list of error messages.
- Error Details:** A section on the right for detailed error information.
- Kasli Log:** A section on the right for logging.

Annotations provide additional context:

- The Client ID is the same as the uService name as we didn't overwrite the name during the startup** (points to the Client ID dropdown).
- We can view our uService by selecting it from the Guardian** (points to the Guardians List).
- Message is shown twice, because we show sent and received messages** (points to the Message Log).
- We can send a message to itself** (points to the Addressee dropdown).

Simple chat – start another instance of the uService

The screenshot displays the CIRCUS+TALOS interface. At the top, there are two panels: 'Guardians List' (Size: 1) and 'uServices Online' (Size: 8). The 'Guardians List' shows a single entry for 'local' with a status of '23:05:54.294 07/11/2024'. The 'uServices Online' panel shows three entries: 'DAQ Sender of local', 'Error Manager', and 'GUI of local', all with a status of '23:05:52.870 07/11/2024'. In the center, there is a large illustration of a red and white striped circus tent with a sign that says 'CIRCUS'. To the right of the tent, there is a 'STOP' button and a 'PRESS CTRL TO ARM' label. Below the tent, there is a 'local Subpanel' label. On the left side of the 'local Subpanel', there is a 'uService' section with a dropdown menu set to 'Simple Chat', an 'overwrite name' field containing 'Simpleton1', and two buttons: 'Start uService' and 'Stop uService'. A blue arrow points from the 'Start uService' button to a text box. In the center of the 'local Subpanel', there are several buttons: 'Start online checks', 'Stop online checks', 'Send Pat to all GDs', 'List the uServices available for launching', 'Reset ABORT', 'Reload Config files', and 'Update from GIT'. There is also a 'Dismiss All Errors' button. On the right side of the 'local Subpanel', there is a 'Message received' section with a large grey area. At the bottom right, there is an 'Error Details' section and a 'Kasli Log' section. A text box at the bottom center contains the text: 'We can start a second instance of the uService with a different name by typing it in the *overwrite name space*'.

Size: 1 Guardians List

local	23:05:54.294 07/11/2024
	00:00:00.000 DD/MM/YYYY
	00:00:00.000 DD/MM/YYYY

Size: 8 ☐ All? uServices Online

DAQ Sender of local	23:05:52.870 07/11/2024
Error Manager	23:05:52.870 07/11/2024
GUI of local	23:05:52.870 07/11/2024

Errors

ABORT

RUN local

uService to show local Subpanel

local Subpanel

uService
Simple Chat

overwrite name
Simpleton1

Start uService

Stop uService

Start online checks

Stop online checks

Send Pat to all GDs

List the uServices available for launching

Reset ABORT

Reload Config files

Update from GIT

Dismiss All Errors

Message received

Error Details

Kasli Log

We can start a second instance of the uService with a different name by typing it in the *overwrite name space*

Simple chat – send message between uServices

The screenshot displays the CIRCUS+TALOS interface with several panels:

- Guardians List:** Shows a single entry for 'local' with a status of 23:10:44.661 on 07/11/2024.
- uServices Online:** Shows three entries: 'DAQ Sender of local', 'Error Manager', and 'GUI of local', all with a status of 23:10:43.527 on 07/11/2024.
- Errors:** An empty list box.
- Control Panel:** Includes 'ABORT' and 'RUN' buttons, a 'PRESS CTRL TO ARM' button with a 'STOP' label, and a 'uService to show' dropdown menu currently set to 'Simpleton1'.
- Client ID:** A dropdown menu showing 'Simpleton1'.
- Message Log:** A large text area displaying the message: '07-11-2024 23:10:43 Simpleton1>Simple Chat: What's up?'.
- Message Input:** A text field containing 'What's up?' and a 'SEND' button.
- Addressee:** A dropdown menu showing 'Simple Chat'.
- Error Details:** An empty text area.
- Kasli Log:** An empty text area.

Annotations with blue arrows point to specific elements:

- Annotation 1: 'uService ID is the same as the overwritten name giving us a different client ID' points to the 'Simpleton1' client ID.
- Annotation 2: 'Second instance of the uService with overwritten name' points to the 'Simple Chat' addressee.
- Annotation 3: 'Sending message to another uService is shown only as a single message' points to the message log entry.
- Annotation 4: 'We can send a message to another uService' points to the 'SEND' button.

Simple chat – received message by the uService

The screenshot displays the CIRCUS+TALOS Simple Chat interface. At the top, there are two panels: 'Guardians List' (Size: 1) and 'uServices Online' (Size: 8). The 'Guardians List' shows a single entry for 'local' with a status of '23:13:39.927 07/11/2024'. The 'uServices Online' panel shows three entries: 'DAQ Sender of local' (23:13:38.895 07/11/2024), 'Error Manager' (23:13:38.897 07/11/2024), and 'GUI of local' (23:13:38.896 07/11/2024). In the center, there is a cartoon illustration of a circus tent with a sign that says 'CIRCUS'. To the right of the tent is a 'STOP' button. Below the tent, there is a 'Message Log' section showing a list of messages. The first two messages are 'Hello there' from 'Simple Chat' to 'Simple Chat' at 23:03:29. The third message is 'What's up?' from 'Simpleton1' to 'Simple Chat' at 23:10:43. A blue arrow points from a text box to the third message in the log. Below the message log, there is a 'Message' input field containing 'Hello there' and a 'SEND' button. To the right of the 'Message' input field is an 'Addressee' dropdown menu set to 'Simple Chat'. On the far right, there is a 'Kasli Log' section. At the top right, there is a 'PRESS CTRL TO ARM' button with a 'STOP' label. Below it, there is a 'RUN' button with a '0' value and a 'local' dropdown menu. Below the 'RUN' button is a 'uService to show' dropdown menu set to 'Simple Chat'. Below the 'uService to show' dropdown menu is an 'Error Details' section. Below the 'Error Details' section is a 'Kasli Log' section.

Size: 1 Guardians List

local	23:13:39.927 07/11/2024
	00:00:00.000 DD/MM/YYYY
	00:00:00.000 DD/MM/YYYY

Size: 8 All? uServices Online

DAQ Sender of local	23:13:38.895 07/11/2024
Error Manager	23:13:38.897 07/11/2024
GUI of local	23:13:38.896 07/11/2024

Errors

Client ID
Simple Chat

SIMPLE CHAT

STOP

Message Log

- 07-11-2024 23:03:29 Simple Chat>Simple Chat:
Hello there
- 07-11-2024 23:03:29 Simple Chat>Simple Chat:
Hello there
- 07-11-2024 23:10:43 Simpleton1>Simple Chat:
What's up?

Message
Hello there

SEND

Addressee
Simple Chat

On the initial uService we can see the new message from the "Simpleton1" uService

ABORT

PRESS CTRL TO ARM

STOP

RUN

0 local

uService to show
Simple Chat

Error Details

Kasli Log