

# OSP FMU:s - Minimal example

<https://github.com/thisDotLucas/DATSimulatorFMU>

In this document I describe the workflow using the [cppfmu](#) library to create FMU:s (Functional mockup unit) using c++ and how to use them as components in the OSP (Open simulation plattform) environment. In this example I will recreate the example we previously did using SystemC.

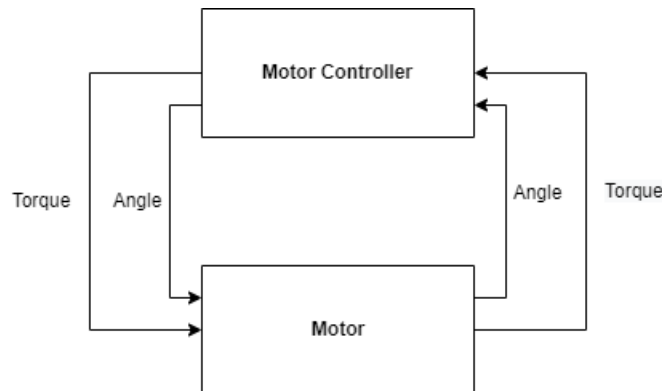


Figure 1: Diagram of the components from the SystemC example, the motor continuously reports the angle and torque values to the motor controller and the controller can write new values to the motor.

## 1 Creating FMU:s

The source code for the motor and motor controller FMU:s can be found in the Github url at the top of the document. It is a fork of the [cpp-fmus](#) repository which is a good template for using the cppfmu library written by the author of the cppfmu library with some examples and the cmake configurations to make the build process seamless. To get started on creating

an FMU start of by creating a directory with the name of your fmu, under it create a sources directory where you will write your C++ code and a file [modelDescription.xml](#) which describes the model, in our simple example we have to describe the four variables and define which of them are output variables. For the C++ part create a file fmu.cpp where you define a class that inherits the abstract class SlaveInstance from the cppfmu library. It contains a set of virtual methods, which you may implement or not apart from the function DoStep which is mandatory to implement, as the name suggest it defines the behaviour for each time step in the simulation. For more details about how to implement the FMU:s see the Github page for the [cppfmu](#) library.

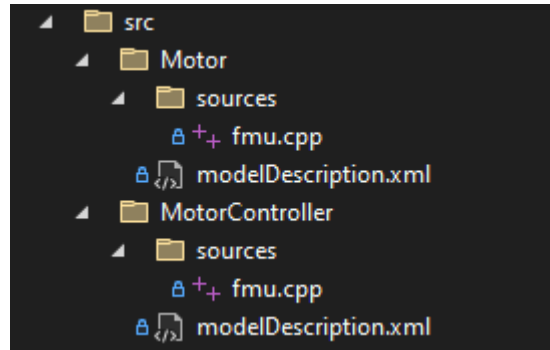


Figure 2: The previously described file hierarchy, note that the base hierarchy must be defined like this for Cmake to build the FMU correctly, you can add more files freely but these must exist. Also in the root CmakeLists.txt add the name of your FMU into the list of FMU:s.

## 2 Setting up OSP

With the FMU:s we can set up the OSP environment, I will describe how it can be done for this set of FMU:s to create the model in figure 1. In the [OSP workshop tutorial](#) Github repository you can find a set of tutorials which I recommend if you want a better understanding of the process of setting up and running a simulation. It also contains the software we need to validate our models and run the simulation, you can find instructions how to download them in the prerequisite documents in the Github repository.

To start of we need to create a directory for our model, in that directory we place our FMU:s. Now we need to create a corresponding OSP model description xml file for each of the FMU:s





 Motor.fmu	23/11/2022 10:37	FMU File
 Motor_OspModelDescription.xml	22/11/2022 10:59	XML Document
 MotorController.fmu	23/11/2022 10:37	FMU File
 MotorController_OspModelDescription.x...	22/11/2022 11:06	XML Document

Figure 3: The content of the OSP directory so far.

In the [OSP model description](#) xml file we define variable groups, in our simple example we create a group for each variable.

```
<?xml version="1.0" encoding="utf-8" ?>
<OspModelDescription xmlns="https://open-simulation-platform.com/
  OspModelDescription/1.0.0" version="1.0">
  <VariableGroups>
    <Generic name="MotorAngleIn">
      <Variable ref="angleIn" />
    </Generic>
    <Generic name="MotorAngleOut">
      <Variable ref="angleOut" />
    </Generic>
    <Generic name="MotorTorqueIn">
      <Variable ref="torqueIn" />
    </Generic>
    <Generic name="MotorTorqueOut">
      <Variable ref="torqueOut" />
    </Generic>
  </VariableGroups>
</OspModelDescription>
```

This is the content of the Motor\_OspModelDescription.xml file. The motor controller is essentially identical as it has the same amount of in- and output variables for the angle and torque. To make sure everything is properly setup so far you can use the validator from the [OSP workshop tutorial](#) Github repository.

Now we need to define the system, this is done through the [OSP system](#)

[structure](#) xml file. Start of by creating a file named `OspSystemStructure.xml` in your OSP directory.

```
<?xml version="1.0" encoding="utf-8" ?>
<OspSystemStructure xmlns="http://opensimulationplatform.com/MSMI/
  OSPSystemStructure" version="0.1">
  <StartTime>0.0</StartTime>
  <BaseStepSize>0.01</BaseStepSize>
  <Algorithm>fixedStep</Algorithm>
  <Simulators>
    <Simulator name="Motor" source="Motor.fmu" stepSize="0.1" />
    <Simulator name="MotorController" source="MotorController.fmu"
      stepSize="0.1" />
  </Simulators>
  <Connections>
    <VariableConnection>
      <Variable simulator="Motor" name="angleOut" />
      <Variable simulator="MotorController" name="angleIn" />
    </VariableConnection>
    <VariableConnection>
      <Variable simulator="Motor" name="torqueOut" />
      <Variable simulator="MotorController" name="torqueIn" />
    </VariableConnection>
    <VariableConnection>
      <Variable simulator="MotorController" name="angleOut" />
      <Variable simulator="Motor" name="angleIn" />
    </VariableConnection>
    <VariableConnection>
      <Variable simulator="MotorController" name="torqueOut" />
      <Variable simulator="Motor" name="torqueIn" />
    </VariableConnection>
  </Connections>
</OspSystemStructure>
```

The most relevant part for this example is the defining of the simulators and the connections between the variables. For more information about the rest of the xml elements and further functionality complete the [OSP workshop tutorial](#) or check the [documentation](#).






 Motor.fmu	23/11/2022 10:37	FMU File
 Motor_OspModelDescription.xml	22/11/2022 10:59	XML Document
 MotorController.fmu	23/11/2022 10:37	FMU File
 MotorController_OspModelDescription.x...	22/11/2022 11:06	XML Document
 OspSystemStructure.xml	22/11/2022 11:24	XML Document

Figure 4: The content of the OSP directory with the added OspSystemStructure.xml file.

With this we are able to run a simulation using the cosim application from the [OSP workshop tutorial](#) Github repository. We simply load the configuration by specifying the path to our OSP directory in the GUI interface and we can fiddle with the different parameters to our liking, this is described more in depth in the [tutorial](#) and in the [documentation](#) for the cosim application as well as how to create plots using the variables.

So far we have no way of interacting with the variables. We would like some way to set the out values sent from the motor controller. For this OSP uses [scenarios](#), to create a scenario you should create a directory called scenarios inside of your OSP directory. In there you can create scenario files in a JSON format where you can set variables to a given value at a given time in the simulation.

```
{
  "description": "Test_scenario",
  "defaults":
  {
    "model": "MotorController",
    "action": "override"
  },
  "events": [
    {
      "time": 0.0,
      "variable": "angleOut",
      "value": 10.0
    },
    {
      "time": 0.0,
      "variable": "torqueOut",
```

```

        "value": 20.0
    },
    {
        "time": 200.0,
        "variable": "angleOut",
        "value": 20.0
    },
    {
        "time": 300.0,
        "variable": "torqueOut",
        "value": 50.0
    }
]
}

```

In defaults we can set default values for some of the attributes, so if the attribute is not explicitly defined inside an event it will use the value defined in the defaults object for that attribute. In our example the scenario will only affect the output variables from the motor controller so it's set as default and being implicitly used inside all of the events. The time, value and variable are all explicitly defined in each event. The variable key defines which variable is affected by the event, the time key defines at what time in the simulation the event occurs and the value key defines the value to be written to the variable.

Now in the cosim simulation application it will detect the scenarios in the scenarios directory and you can run the simulation where the motor controller writes to the motor during the simulation. In the Github repository at the top of the document you can find a ZIP folder of my OSP directory.