Exercises for imc FAMOS II – Digital Course

- Block 5 - Doc. Rev.: 1.2- 28.08.2025

Exercise A

Exercise objective:

In this exercise you will learn how to handle subsequences and their optional transfer/return parameters. The sequence from exercise A block 1 serves as an example, in which calculations are applied to various data sets. These calculations are outsourced to subsequences in this exercise. Through the outsourcing, a simplification as well as a greater flexibility of the evaluation sequence is achieved.

Task formulation:

A) Simplify the sequence from Exercise A Block 1 (also included in the sample files) by separating the marker placement section into a subsequence. Transfer parameters should be:
   - The channel where the marker is to be created.
   - X, and Y coordinates of the marker.
   - An additional text of your choice, that will appear additionally in front of the existing marker text.

B) Next, in the same sequence, outsource the creation of the 600 s parts as well as the calculation of their maximum values into a 2nd subsequence:
   - Transfer parameters: Channel, X-value for left intersection, Y-value for right intersection.
   - Return parameter: Cut out section, maximum value of the section.

Result:

You get an evaluation sequence with 2 subsequences, that are called at their respective positions in the main sequence.

Exercise steps:

<u>A) Subsequence for setting the markers:</u>

- A subsequence call is to be implemented according to the exercise description in the following general form (parameter names for illustration only):

```
Seq    SetMarker    Channel,    X-Pos,    Y-Pos,    TextBeforeMax
```

- First load the saved sequence from Block 1 Exercise A and create a subsequence named **SetMarker** (menu-icon).
  <u>Note:</u> It is also possible to work with a separate sequence file of the same name instead.

- Copy one of the blocks from the main sequence that sets the markers in the curve window, e.g. the **Speed** block, and paste it into the subsequence:

```
CwSelectByChannel("line", Speed)
CwNewElement("marker")
CwMarkerSet("line.selected", 1)
CwMarkerSet("x.type", 1)
CwMarkerSet("y.type", 1)
CwMarkerSet("x", Pos(CutSpeed, MaxSpeed))
CwMarkerSet("y", MaxSpeed)
CwMarkerSet("text", TForm(MaxSpeed, "f1.3"))
```

- Replace the **Speed** variable with **PA1**, the 1st parameter from our sequence call that contains the channel:

```
CwSelectByChannel("line", PA1)
```

- For the remaining transfer parameters, create meaningful local variables for better readability, and assign them at the beginning of the subsequence at the very top:

```
Local xpos = PA2
Local ypos = PA3
Local AddText = PA4
```

- Adjust the last 3 sequence lines according to the new variable and also add the free text of the marker:

```
CwMarkerSet("x", xpos)
CwMarkerSet("y", ypos)
CwMarkerSet("text", AddText + TForm(ypos, "f1.3")); Text in front of
Max-Value
```

<u>Note:</u> The assignment to a local variable cannot be used for PA1 in this case, because otherwise the command **CwSelectByChannel()** would look for this local variable in the curve window without success.

- The subsequence is now completed. In the main sequence, replace the 3 blocks that set the markers with corresponding calls to the subsequence:

```
Seq SetMarker Speed, Pos(CutSpeed, MaxSpeed), Maxspeed, "Max = "
Seq SetMarker Torque, Pos(CutTorque, MaxTorque), MaxTorque, "Max = "
Seq SetMarker Power_Engine, Pos(CutPower, MaxPower),
          MaxPower, "Max = "
```

<u>B) Subsequence for creating the parts and calculating their maximum values:</u>

- According to the exercise description, a subsequence call for the creation of the 600 s parts and the calculation of their maximum values should be implemented in the following general form:

```
Seq  CutMax  Channel,  X_Left,  X_Right,  Part,  MaxPart
```

Parameters 1 - 3 are to be passed to the subsequence, parameters 4 and 5 contain the return values.

- Create a new subsequence named **SetMarker** and define the **Cut()** and **Max()** functions in it using the corresponding parameters:

```
PA4 = Cut(PA1, PA2, PA3)
PA5 = Max(PA4)
```

- In the main sequence, replace the entire block of all **Cut()** and **Max()** functions with corresponding subsequence calls:

```
Seq CutMax Speed, Range0, Range1, CutSpeed, MaxSpeed
Seq CutMax Torque, Range0, Range1, CutTorque, MaxTorque
Seq CutMax Power_Engine, Range0, Range1, CutPower, MaxPower
```

Exercise B

Exercise objective:

In this exercise you will learn how to extend the FAMOS function library with your own self-created functions. The created functions behave like supplied functions. Among other things, they can be found via the search function in the function library and can be configured through the function wizard. Help texts for the operation of the function can also be included in the definition.

In addition to creating such sequence functions, this exercise covers techniques for querying the data type as well as how to handle error handling.

Task formulation:

Write a sequence function that extracts its time trace from a normal or from an XY data set without segments and without events. Catch errors when passing an incorrect data type. This function is implemented in FAMOS 2025 as CmpX(Data) -> ComponentX as a ready-made function. To illustrate the basic principles of sequence functions, a simplified example is provided here.

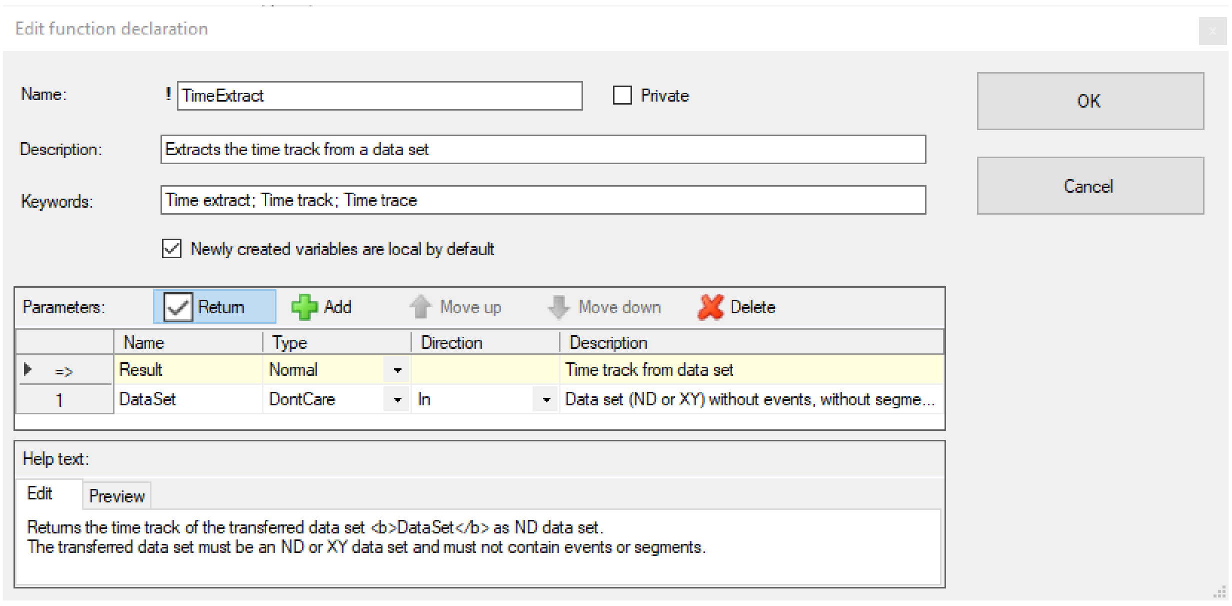
Result:

If a ND or XY variable is passed to the created function without events and without segments, a time track is created, otherwise an error message is displayed.

Exercise steps:

Step 1: Create declaration

- Open the input window and create a new sequence function by clicking on the corresponding menu icon . In the declaration dialog you then define the following:
  - Assign a name for the sequence (e.g. **TimeExtract**).
  - Enter a description for the sequence (e.g. **Extracts the time track from a data set**).
  - Use some keywords by which your sequence can be found later in the search function of the function library (e.g. **Time extract; Time track; Time Trace**).
  - Newly created variables should be local by default.
  - The function should not be declared as a private function.
  - Enter a help text that will be displayed in the help window (e.g. **Returns time track of an ND or XY dataset, no events or segments allowed**).
  - Add a transfer parameter by clicking on Add . This should correspond to the transferred data set (e.g. **DataSet**) and initially not belong to any special data type.
  - Create a return parameter as a normal dataset that returns the extracted time track (e.g. **Result**). To do this, activate the field ✓ Return .
  - The complete declaration dialog should look something like the screenshot on the next page.
  - Confirm the declaration dialog with OK and save the sequence in the FAMOS default folder for sequences. The editor window for the sequence function then opens.



Edit function declaration

Name:	! TimeExtract ☐ Private
Description:	Extracts the time track from a data set
Keywords:	Time extract; Time track; Time trace
	☑ Newly created variables are local by default

Parameters: ☑ Return | Add | Move up | Move down | Delete

	Name	Type	Direction	Description
=>	Result	Normal		Time track from data set
1	DataSet	DontCare	In	Data set (ND or XY) without events, without segme...

Help text:

Edit | Preview

Returns the time track of the transferred data set <b>DataSet</b> as ND data set.
The transferred data set must be an ND or XY data set and must not contain events or segments.

Step 2: Create source code in the editor

- Since the extraction of the time trace is different for the allowed data types, at the beginning of the sequence function the data type of the passed data set shall be classified using the function **VerifyVar()** as conditions for several **If** statements:

```
If VerifyVar(DataSet, "ND(->)")
    ; Sequence for ND datasets without events and segments
ElseIf VerifyVar(DataSet, "XY(->)")
    ; Sequence for XY datasets without events and segments
Else
    ; Sequence in case of error
End
```

  Within the commented code blocks the generation or extraction of the time trace and in case of an error the generation of an error message will be done later on.
- In the case of a normal data set (ND), create a ramp with the start time value of the passed data set, its sampling rate, and the number of samples of the passed data set, and pass the **Result**:

```
Result = Ramp(xOff?(DataSet), xDel?(DataSet), Leng?(Dataset))
```

- You get the time track from an XY dataset directly from the X track of the passed dataset:

```
Result = DataSet.X
```

- For the error case (no pure ND or XY data set) define an error message and output it using the **ThrowError()** function:

```
ThrowError("Command '!TimeExtract': Parameter must contain a ND or
           XY dataset without events or segments.")
```

  Note: By calling **ThrowError()** an error is generated. By default, FAMOS interrupts sequence processing and opens the sequence function editor in case of runtime errors, but also in case of generated errors. This default behavior can be changed with the function **OnError()**.
- At the very beginning of the sequence function, adjust the option for the behavior in case of error so that the sequence jumps to the end without interruption and the editor does not remain open:

```
OnError("ReturnFail")
```

- The complete sequence should look something like the following:

```
OnError("ReturnFail")
If VerifyVar(DataSet, "ND(->)")
   Result = Ramp(xOff?(DataSet), xDel?(DataSet), Leng?(Dataset))
ElseIf VerifyVar(DataSet, "XY(->)")
   Result = DataSet.X
Else
   ThrowError("Command '!TimeExtract': Parameter must contain a
            normal Dataset or XY dataset without events or segments")
End
```

- Save the sequence function.

Step 3: Test and debug new function

- Load the slope.dat dataset from the sample data into the variable list.
- Open the input window and search for the sequence function in the function library using the name or one of the previously specified keywords.
- Select the function and call the function wizard (**Shift + F1** key combination). Select the data set **slope** as parameter and enter a result name (e.g. **slope_timetrack**). Copy the command line into the input window and then close the wizard. The command line should read:

  ```
  slope_timetrack = !TimeExtract(slope)
  ```

  Highlight the command so the compiler arrow is in front of it and from the **Run** menu select the item Single step in sub-sequence (**Shift + F9 key combination**).
  The editor of the sequence function opens and you can execute the sequence as usual via single step () or the entire sequence execution (). If the execution is successful, you will receive the data set **slope_timetrack** in the variable list.
- Open the calculated time track and the original data set in the data editor and compare the time information.
- Load the **rpm_V.dat** file from the sample data and create a time track for the imported data set using the created sequence function. This time, execute the command directly without jumping to the subsequence.
- For testing the error case create a text variable and try to calculate a time trace from it.