Exercises for imc FAMOS II – Digital Course

- Block 3 -

Doc. Rev.: 1.2- 28.08.2025

Exercise A

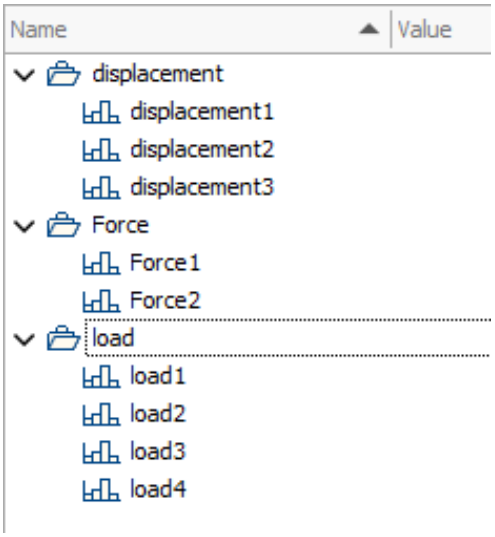
Exercise objective:

In this exercise you will learn how to use an alternative method to **FileLoad()** to load data in FAMOS format. While **FileLoad()** loads a file including its entire content immediately, the **FileOpenDSF()** function can be used to search the file for data objects first. Thus, the handling of existing measurement data can be influenced directly during loading. Selective loading from a file is also possible this way.

Task formulation:

The **LoadTestSample.dat** dataset contains seven datasets from 3 different categories (see also exercises from block 1). Create a separate group for each category and load the data sets into the associated groups.

Result:



After loading you will get three groups in the variable list, containing the related records, see the adjacent screenshot.

Exercise steps:

- Open the file **LoadTestSample.dat** with the function **FileOpenDSF()** for reading. In order not to forget to close the file at the end of the sequence, directly also create the corresponding command to close it.

```
file = FileOpenDSF(filename,0)
FileClose(file)
```

  The variable **filename** contains the complete path to the file **LoadTestSample.dat**.

  Attention: The argument for the opening type should be chosen carefully. 0 for **reading** is harmless, but a 1 for **writing** overwrites an existing file without asking!
- Query the number of data objects in the open file:

```
count = FileObjNum?(file)
```


- Next, create a **For** loop in which the data objects are processed one by one:

```
For i = 1 To count
   ; Loop code
End
```

- Inside the loop, use the **FileObjName?()** function to read the name of the i-th data object.

```
name = FileObjName?(file,i)
```

- Using an **If** query, the name is then to be assigned to its associated group by means of a partial text comparison. If the match you are looking for is found, add the desired group name followed by a colon to the name variable. As an example for **Load**:

```
If TLike(name, "Load*",0)
   name = "Load:" + name
End
```

  Create analogous **if**-queries for **Displacement** and **Force**.

- At the end of the loop, the measurement data can be read into a newly created variable using the **FileObjRead()** function. The name of the new variable is defined by the **name** textvariable.

```
<name> = FileObjRead(file,i)
```

Note: Usually there is only one data object in a *.raw file from an imc measuring device, with *.dat there can be several, as in the present example. A group in a file is counted as one object, independent of the number of variables in the group..

A possible complete solution to this exercise can be found on the next page. In the solution, the paths and file names were written into single variables and the complete name was assembled afterwards. In addition, all temporary variables were prefixed with an underscore and defined as temporary variables using the **local** command:


```
; Solution for the Exercise A Block 3
; by imc T&M Training Team
Local _*
_Basepath = "Please insert your Path"
_filename = _basepath + "LoadTestSample.dat"
_file = FileOpenDSF(_filename,0) ; important: only open for reading
not for writing
_count = FileObjNum?(_file)
For _i = 1 To _count
   _name = FileObjName?(_file,_i)
   ; the name may not follow the naming conventions of FAMOS!
   ; here we imply it does
   If TLike(_name, "Load*",0)
      _name = "Load:" + _name
   End
   If TLike(_name, "displ*",0)
      _name = "Displacement:" + _name
   End
   If TLike(_name, "forc*",0)
      _name = "Force:" + _name
   End
   <_name> = FileObjRead(_file,_i)
End
FileClose(_file)
```


Exercise B

Exercise objective:

This exercise shows the import of data beyond the FAMOS format. Specifically, this exercise deals with the import of text files, whereby two possible types of text files are presented and used. Firstly ASCII files, which contain long measurement data sets and are therefore read in via the **FileOpenFAS()** function and an appropriately created import filter, and secondly text files, which contain additional information such as part numbers, special measurement values of a special measuring device or similar..

Task formulation:

Load the measurement data from the file **Example_ASCII_Import.txt** using the function **FileOpenFAS()** into corresponding FAMOS variables. Then load the additional information like the names of the machine and the tester as well as the rel. humidity from the text file **Info_Measurement.txt** by reading the file line by line or as a whole and extracting the respective information using the text functions from the function library.

Result:

After the import, both the measurement data from the file **Example_ASCII_Import.txt** and the additional information from the file **Info_Measurement.txt** are available in the variable list.

Exercise steps:

1st part: Loading the file **Example_ASCII_Import.txt**

- Copy the sequence from exercise A without the block for the **if**-queries and replace the function **FileOpenDSF()** by **FileOpenFAS()** with the ASC-II import filter:

```
file = FileOpenFAS(filename,
        "#ImportAscii1.dll|ASCII/Excel Import...", 0)
```

  The complete function then reads as follows:

```
; Solution for Exercise B Block 3
; by imc T&M training team
Local _*
_basepath = "Please insert your Path"
_filename = _basepath + "Beispiel_ASCII_Import.txt"
_file = FileOpenFAS(_filename,
        "#ImportAscii1.dll|ASCII/Excel Import...", 0)
_count = FileObjNum?(_file)
For _i = 1 To _count
   _name = FileObjName?(_file,_i)
   <_name> = FileObjRead(_file,_i)
End
FileClose(_file)
```


- Fill the **basepath** variable with the path of the file **Example_ASCII_Import.txt** from the example data and execute the sequence.
  Since in this case no filter was specified for the ASCII file, the wizard opens. If no channel names are assigned to the data series in the wizard, they are automatically generated as **Channel1** with consecutive numbering.

  Tip 1: A previously defined ASCII import filter can be assigned directly in the **FileOpenFAS()** function. The notation for this is:

  ```
  "#ImportAscii1.dll|ASCII/Excel Import...|Importfilter"
  ```

  Tip 2: The opening of the wizard can be prevented by specifying the parameter **Hide=1**:

  ```
  _file = FileOpenFAS(_filename,"#ImportAscii1.dll|ASCII/Excel
  Import...|Hide=1", 0)
  ```

  In this case, the wizard uses the last settings made, so the most recent utilization by the user can influence the automatism.

2nd part: Extraction of additional info from the text file **Info_Measurement.txt**.

Variant 1:

Use **FileOpenASCII()** to open a text file, then use **FileLineRead()** to load the contents line by line into a text array. This can then be processed element by element and decomposed to the desired information.

- Create a new sequence and create an empty text array:

  ```
  lines = TxArrayCreate(0)
  ```

- Open the file Info_Measurement.txt with the **FileOpenASCII()** function for reading. Again using this function the file must be closed after usage:

  ```
  file = FileOpenASCII(filename ,0,1)
  ; further code
  FileClose(file)
  ```

  The variable **filename** again contains the complete path of the file.
- Read the lines from the opened file into the previously created text array:

  ```
  status = FileLineRead(file,_lines,0)
  ```

  The information to be found is now contained in the individual elements of the text array.
- In order to read out the values, they still have to be separated from the description texts. To do this, use the colon as a separator in the **TxSplit()** function:

  ```
  lineparts = TxSplit(lines[1],":")
  ```

- From the parts you can now fill a corresponding variable. For removing unnecessary spaces use the function TConv():

  ```
  SerialNumber = TConv(lineparts[2],6)
  ```

- Proceed analogously to read out the **tester**:

  ```
  lineparts = TxSplit(lines [2],":")
  Tester = TConv(lineparts[2],4)
  ```


- Read the value for **Humidity**, this is to be created as a numeric value in FAMOS. To do this, convert the text using the **TtoSv()** function:

```
lineparts = TxSplit(lines[3],":")
Humidity = TtoSv(lineparts[2], "a")
```

- Set the unit of the **Humidity** variable by cutting it out of the text element using **TPart()**:

```
SetUnit(Humidity, TPart(lineparts[2], Tleng(lineparts[2]),1),1)
```

A possible complete solution to this exercise can be found below. In the solution, the paths and file names were written into single variables and the complete name was assembled afterwards. In addition, all temporary variables were prefixed with an underscore and defined as temporary variables by the **local** command:

```
; Solution for Exercise C Block 3
; by imc T&M Training Team
Local _*
_lines = TxArrayCreate(0)
_Basepath =
_filename = _basepath + "Info_Measurment.txt"
_file = FileOpenASCII(_filename ,0,1)
_status = FileLineRead(_file,_ lines,0)
FileClose(_file)
_lineparts = TxSplit(_lines [1],":")
SerialNumber = TConv(_lineparts [2],6)
_ lineparts = TxSplit(_lines [2],":")
Tester = TConv(_lineparts [2],4)
_ lineparts = TxSplit(_lines [3],":")
Humidity = TtoSV(_lineparts [2], "a")
SetUnit(Humidity,TPart(_lineparts [2],Tleng(_lineparts [2]),1),1)
```

Variant 2:

Instead of reading the file line by line, the function **FileOpenFAS()** with the argument **imc|text** loads the complete text file into a text variable. This is then split into individual lines using the **TxSplit()** function and the parts can then be further processed in the same way as in variant 1.

- Create a new sequence.
- Load the complete file into a text variable. To do this, open the file, read the content and close it again:

```
file = FileOpenFas(filename,"imc/Text/auto",0)
content = FileObjRead(file,1)
FileClose(file)
```

- The individual lines can be identified by the line break (ASCII characters 13 and 10). Use these in the **TxSplit()** function:

```
lines = TxSplit(content, "~013"+"~010")
```


Now the individual lines are again available in a text array and can be processed analogously to variant 1.
Note: Variant 1 is often more convenient for line-oriented texts, variant 2 is suitable for xml files or non-line-oriented text files.

Exercise C

Exercise objective:

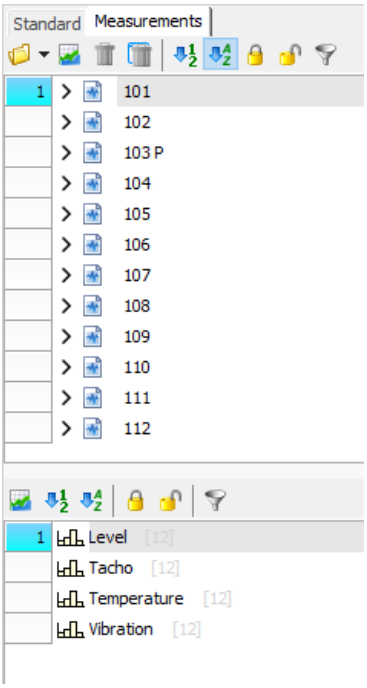
In this exercise, you will learn how to access the PC's file system in sequence using functions from the function library to search for and read in folders or files. In addition to reading in folder structures and file lists, techniques for splitting file paths and directly assigning a measurement affiliation are also used.

Task formulation:

Load all brake measurements of a train from the **Test** folder in a way that they have the same measurement name as the folder from which they were imported.

Result:

After loading the folder, all files including measurement affiliation are loaded. Switch to the **Measurements** tab in the Variables window to get the measurement view.



Exercise steps:

- Create a new sequence.
- Examine the **Test** folder from the sample files for its contents, including its subfolders. The variable **basepath** describes the complete path of the destination folder:

```
list_of_dirs = FsFileListNew(basepath, "*",1, 0, 1)
; further code
FsFileListClose(list_of_dirs)
```

  The result is a list of all complete paths to the found folders. Analogous to open files, lists must also be closed again after use, so add the line directly with it.
- Query the number of folders found:


```
number_of_dirs = FsFileListGetCount(list_of_dirs)
```

- With the found number of folders you create a loop over all found folders, so that they can be processed one after the other:

```
For i = 1 To number_of_dirs
    ; loop code
End
```

- Within the loop, first find out the current folder:

```
act_dir = FsFileListGetName(list_of_dirs,i)
```

- Find all files inside the current folder and save the full paths in a list. Even with this list, do not forget the command to close the list after its use:

```
list_files = FsFileListNew(act_dir, "*.raw",0, 0, 0)
; further code
FsFileListClose(list_files)
```

- Query the number of files found in the specified folder:

```
number_of_files = FsFileListGetCount(list_files)
```

- Using the number found, create another loop that processes all the files one by one. Make sure that you do not use the same index variable as in the already running loop:

```
For j = 1 To number_of_files
    ; loop code
End
```

- Write the path of the respective files from the list into a working variable:

```
filename = FsFileListGetName(list_files, j)
```

- Open the file, analogous to exercise A, assuming that it contains only a single data object and read the name of the data object. This assumption can be made because these are **.raw** files created by an imc measurement system. Such files usually contain only a single data object. Do not forget to close the file again at the end.

```
; raw files normally only contain one dataset
file = FileOpenDSF(filename, 0)
name = FileObjName?(file,1)
; further code
FileClose(file)
```

- A measurement name is to be added to the variable name, consisting of the file name. Cut the file name from the complete path and then load the data object including the measurement assignment:

```
measname = FsSplitPath(filename , 6)
{<name>}@{<measname>} = FileObjRead(file, 1)
```

Note: The curly brackets ensure that the names in FAMOS do not contain invalid characters and are cleaned up beforehand if necessary.

- Save the sequence and run it.

A possible complete solution to this exercise can be found below. In the solution, the paths and file names were written into individual variables and the complete name was then assembled. In addition, all temporary variables were prefixed with an underscore and defined as temporary variables by the **local** command:

```
; Solution for Exercise C Block 3
; by imc T&M Training Team
Local _*
_basepath = "Please adjust to your own path"
_basepath = "d:\xxx\Sample_Data\Test"
_list_of_dirs = FsFileListNew(_basepath, "*",1, 0, 1)
_number_of_dirs = FsFileListGetCount(_list_of_dirs )
For _i = 1 To _number_of_dirs
   _act_dir = FsFileListGetName(_list_of_dirs,_i)
   _list_files = FsFileListNew(_act_dir, "*.raw",0, 0, 0)
   _number_of_files = FsFileListGetCount(_list_files)
   For _j = 1 To _number_of_files
      _filename = FsFileListGetName(_list_files , _j)
      ; raw files normally only contain one Dataset
      _file = FileOpenDSF(_filename, 0)
      _name = FileObjName?(_file,1)
      _measname = FsSplitPath(_filename , 6)
      {<_name>}@{<_measname>} = FileObjRead(_file, 1)
      FileClose(_file)
   End
   FsFileListClose(_list_files)
End
FsFileListClose(_list_of_dirs)
```

