

Measuring Arrays and Arrays of Struct

2023-11-14

Support Note SN-IMC-1-139

Author(s) Wittmer, Helena
Restrictions Public Document

Table of Contents

1	Overview.....	2
2	Working with CANape	2
	2.1 Creating a mdf via measurement.....	2
	2.1.1 Selection of the structure and it's elements.....	2
	2.1.2 Example: Full array and full AoS [old database description method].....	6
	2.1.3 Example: Full array and full AoS [new database description method - Typedef]	8
	2.1.4 Example: single arrayelements [old database description method].....	11
	2.1.5 Example: single arrayelements [new database description method - Typedef]	12
	2.2 Creating a measurement file via "save signals".....	14
	2.2.1 Example: Full array	14
	2.2.2 Example: Single arrayelements	15
	2.3 Creating a mdf via "save file as"	15
3	Summary.....	16
4	Working with MDF4lib.....	17
5	Contacts.....	18

1 Overview

When elements of an array or AoS (Array of Struct) with more than 10 elements get written into a measurement file in CANape there are several possible outcomes on how the name of these elements is saved into a mdf file.

For example:

Arrayelement [9]

Arrayelement [09]

This document's intent is to give an overview on which outcome is to be expected in different settings and how to handle these in combination with the MDF4 lib.

As there are several ways a measurement file is created in CANape the dependencies need to be understood beforehand so that the file can be created with the expected result.

2 Working with CANape

When a measurement file is written in CANape the form of the array element name can vary according to the entry of the arrayelement in the measurement configuration. In CANape a mdf file is not only written during a measurement, if an existing measurement gets converted or saved into another measurement file the result how the arrayelement name is saved can also vary.

2.1 Creating a mdf via measurement

Is the measurement file created by recording a measurement, the way the arrayelement is named in the measurement file is determined on one hand by the entry in the measurement configuration and on the other hand by the definition in the database.

The way the elements or structures are written into the measurement configuration also depends on the way the element is selected.

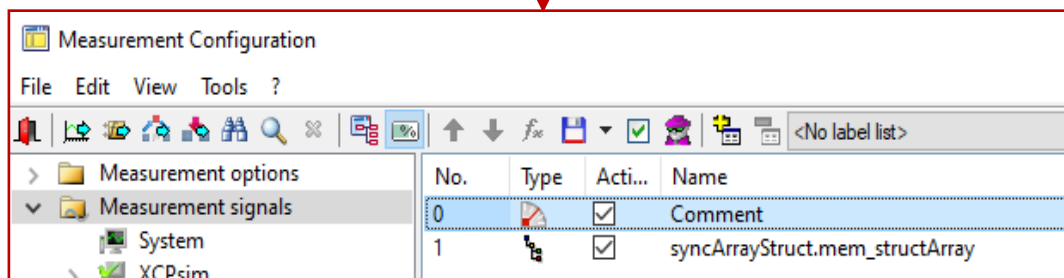
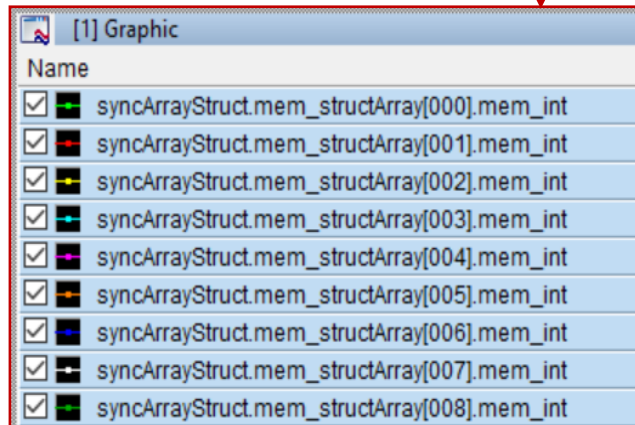
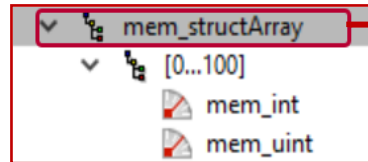
2.1.1 Selection of the structure and it's elements

How the AoS is selected or pulled into the graphic window for visualization can decide how the elements get written into the measurement file. This is however only the case if the AoS is not already included in the measurement configuration.

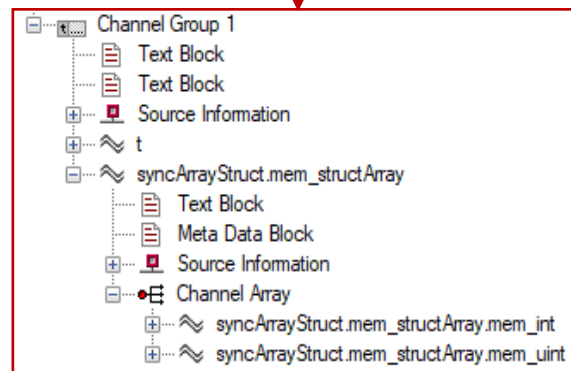
The following examples show different methods how the AoS or it's elements get selected for visualization from the symbol explorer in CANape, how it will show the elements in the graphic window and measurement configuration and what the result in the measurement file will look like (the AoS is defined in the database as typedef struct):

Selection of the structure:

Symbol



MD



The final result is that the AoS is written into the measurement file as whole and each element will be generated by the software reading this file.

Selection of the index:

Symbol Explorer:

The diagram illustrates the process of selecting an array index for measurement. It starts with the Symbol Explorer showing the `mem_structArray` structure. The index `[0...100]` is selected. This selection is then used in the Measurement Configuration window, where the measurement signals are configured. Finally, the resulting MDF file structure is shown, containing the selected array elements.

Symbol Explorer:

- `mem_structArray`
 - `[0...100]` (Selected)
 - `mem_int`
 - `mem_uint`

Measurement Configuration:

No.	Type	Acti...	Name
1		<input checked="" type="checkbox"/>	<code>syncArrayStruct.mem_structArray[000]</code>
2		<input checked="" type="checkbox"/>	<code>syncArrayStruct.mem_structArray[001]</code>
3		<input checked="" type="checkbox"/>	<code>syncArrayStruct.mem_structArray[002]</code>

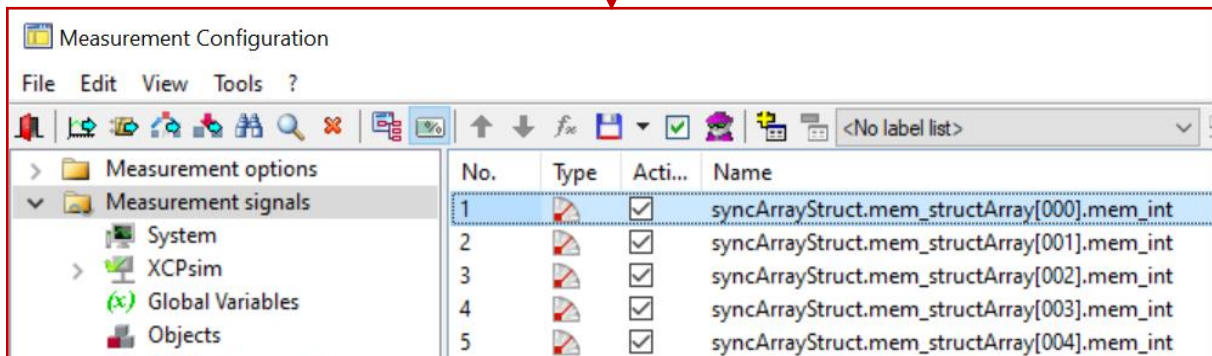
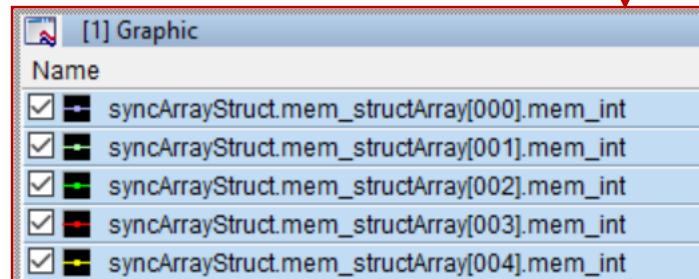
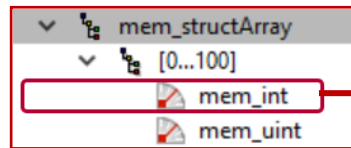
MDF:

- Channel Group 1
 - Text Block
 - Text Block
 - Source Information
 - t
 - `syncArrayStruct.mem_structArray[000]`
 - Text Block
 - Meta Data Block
 - Source Information
 - `syncArrayStruct.mem_structArray[000].mem_int`
 - `syncArrayStruct.mem_structArray[000].mem_uint`
 - `syncArrayStruct.mem_structArray[001]`
 - `syncArrayStruct.mem_structArray[002]`

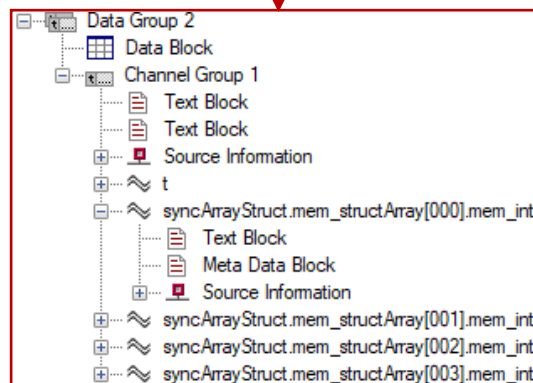
The AoS elements will be written separately into the measurement file, including leading zeros. If all elements of the structure are measured and the result from the previous method is desired the CANape feature "Optimize Measurement" can be used to combine all elements described in the database into the whole structure.

Selection of the arrayelement:

Symbol






MDF



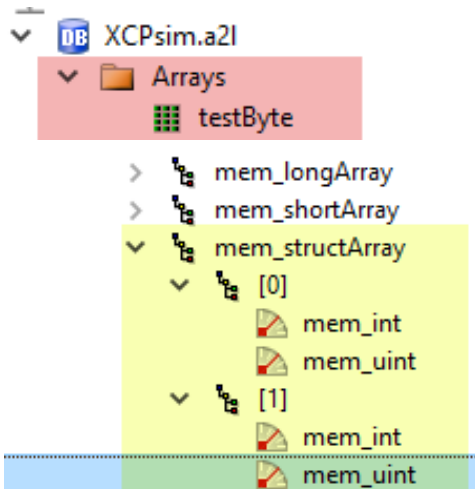
Similar to the previous method all measured elements are written separately into the measurement file with leading zeros.

2.1.2 Example: Full array and full AoS [old database description method]

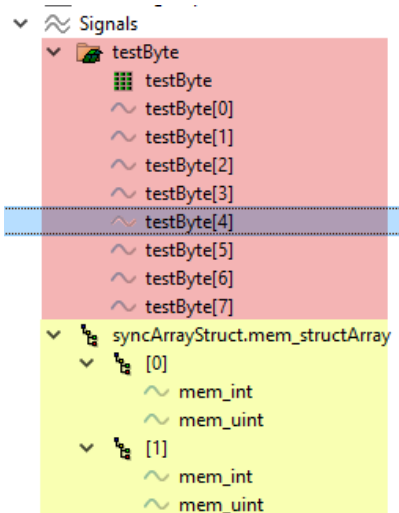
The complete array is measured, the entry in the measurement configuration describes it as full array. The same is done with an AoS:

No.	Type	Acti...	Name
0		<input checked="" type="checkbox"/>	Comment
1		<input checked="" type="checkbox"/>	syncArrayStruct.mem_structArray
2		<input checked="" type="checkbox"/>	testByte

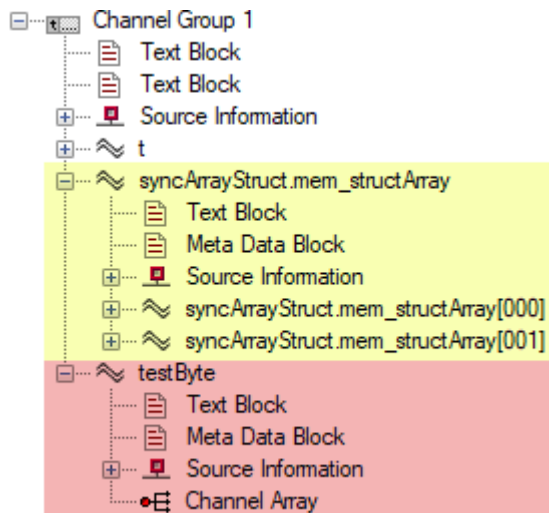
The visualization in the symbol explorer is also important to note because it shows the array as value block without the single elements and the AoS with the single elements:



When we load the mdf file into CANape the visualization in the symbol explorer shows us all elements of the array with their indexes and all elements of the AoS, which is important to note now that all the array elements are visualized under the array structure:



Looking at it's definition in the mdf file we get the following result:



So even if the AoS and the array are fully described in the measurement configuration we got two different results. The reason lies in the database description. As shown here in the database description of the AoS in the present case shows that every element is described separately on their own (one element for reference):

```
/begin MEASUREMENT syncArrayStruct.mem_structArray[000].mem_int ""
    SLONG_NO_COMPU_METHOD 0 0 -2147483648 2147483647
    ECU_ADDRESS 0x1BF4C4
    FORMAT "%.15"
    SYMBOL_LINK "syncArrayStruct.mem_structArray[000].mem_int" 0
    /begin IF_DATA CANAPE_EXT
        100
        LINK_MAP "syncArrayStruct.mem_structArray[000].mem_int" 0x1BF4C4 0 0 0 1 0xDF 0
        DISPLAY 0x0 -2147483648 2147483647
    /end IF_DATA
/end MEASUREMENT
```

The array description in the database shows that there is a description of the overlaying structure first and then all elements are each described separately but belong to the overlaying structure (structure and one element for reference):

```
/begin CHARACTERISTIC testByte ""
    VAL_BLK 0x1BE118 __UByte_Value 0 NO_COMPU_METHOD 0 255
    FORMAT "%.15"
    MATRIX_DIM 8
    SYMBOL_LINK "testByte000" 0
    /begin IF_DATA CANAPE_EXT
        100
        LINK_MAP "testByte000" 0x1BE118 0 0 0 1 0x87 0
        DISPLAY 0x0 0 255
    /end IF_DATA
/end CHARACTERISTIC
```

```




/begin CHARACTERISTIC testByte000 ""
    VALUE 0x1BE118 __UByte_Value_ColumnDir 0 Identity 0 255
    FORMAT "%.15"
    SYMBOL_LINK "testByte000" 0
/begin IF_DATA CANAPE_EXT
    100
    LINK_MAP "testByte000" 0x1BE118 0 0 0 1 0x87 0
    DISPLAY 0x0 0 255
/end IF_DATA
/end CHARACTERISTIC

/begin CHARACTERISTIC testByte000[000] ""
    VALUE 0x1BE118 __UByte_Value_ColumnDir 0 Identity 0 255
    FORMAT "%.15"
    SYMBOL_LINK "testByte000[000]" 0
/begin IF_DATA CANAPE_EXT
    100
    LINK_MAP "testByte000[000]" 0x1BE118 0 0 0 1 0x87 0
    DISPLAY 0x0 0 255
/end IF_DATA
/end CHARACTERISTIC
    
```

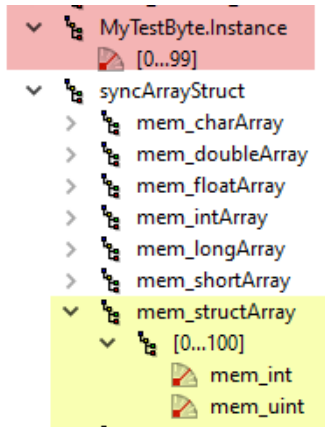
2.1.3 Example: Full array and full AoS [new database description method - Typedef]

With the new database description method we get descriptions for structures which don't need to describe every element on their own. All important data is summarized in the Typedef which also has the benefit that the database size is smaller than with the old method in which each element has a description on their own. For this example the description of the array was also changed from a value block to a measurement structure so it fits the Typedef description.

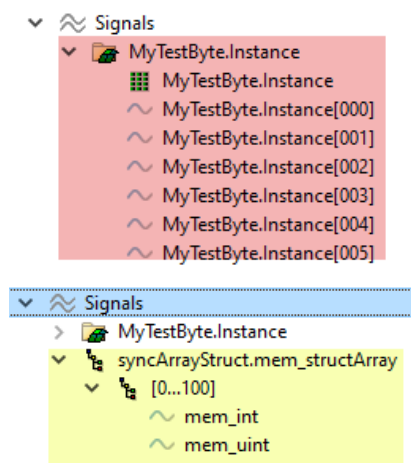
The measurement setup is practically the same as in the example before:

No.	Type	Acti...	Name
0		<input checked="" type="checkbox"/>	Comment
1		<input checked="" type="checkbox"/>	syncArrayStruct.mem_structArray
2		<input checked="" type="checkbox"/>	MyTestByte.Instance

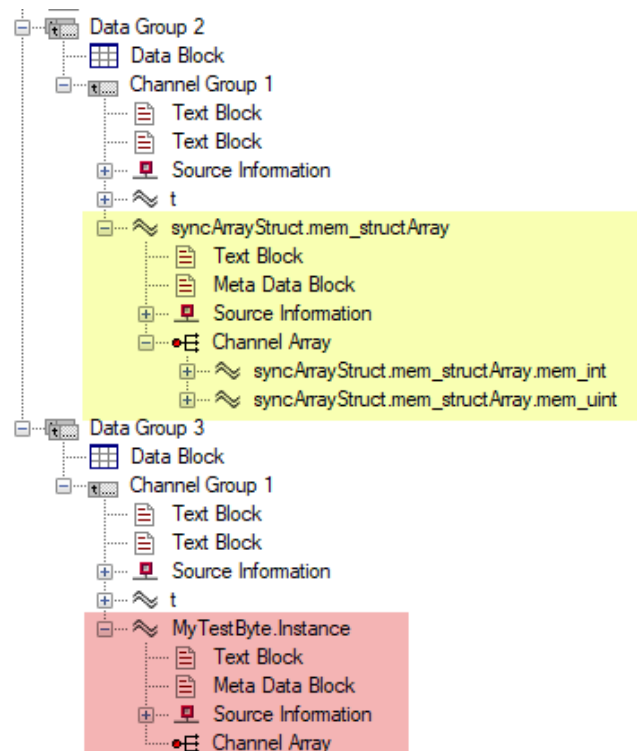
The array and the AoS from the database are visualized like this in the symbol explorer, with this method the visualization of the array looks like the one of the AoS:



And in the resulting measurement file the array is now listed with the elements, the AoS didn't change in it's visualization:



But the result in the measurement file differs, now we have a channel array in the AoS as well which means the index will not have an additional zero if there are more than 10 elements:



The database description of the AoS is now completely different from the previous method. Now the structure and the datatypes in it have a description section and the underlying structure has a short description, the elements in that structure are not described on their own but generated on the information from the structure and datatype description:

```

/begin TYPEDEF_STRUCTURE tSimpleStruct1_ ""
    0x8
    SYMBOL_TYPE_LINK "tSimpleStruct1_"
    /begin STRUCTURE_COMPONENT
        mem_int Measurement_SLong
        0
        SYMBOL_TYPE_LINK "mem_int"
    /end STRUCTURE_COMPONENT
    /begin STRUCTURE_COMPONENT
        mem_uint Measurement_ULong
        4
        SYMBOL_TYPE_LINK "mem_uint"
    /end STRUCTURE_COMPONENT
/end TYPEDEF_STRUCTURE

/begin TYPEDEF_STRUCTURE tSyncStruct1_ ""
    0x68
    SYMBOL_TYPE_LINK "tSyncStruct1_"
    ...
    /begin STRUCTURE_COMPONENT
        mem_structArray tSimpleStruct1_
        84
        SYMBOL_TYPE_LINK "mem_structArray"
        MATRIX_DIM 2
    /end STRUCTURE_COMPONENT
/end TYPEDEF_STRUCTURE

/begin INSTANCE syncArrayStruct ""
    tSyncStruct1_ 0x1BF470
    SYMBOL_LINK "syncArrayStruct" 0
/end INSTANCE
    
```

Which also goes for the array:




```

/begin INSTANCE testByte000 ""
    Measurement_UByte 0x1BE118
    MATRIX_DIM 101
    SYMBOL_LINK "testByte000" 0
/end INSTANCE
    
```

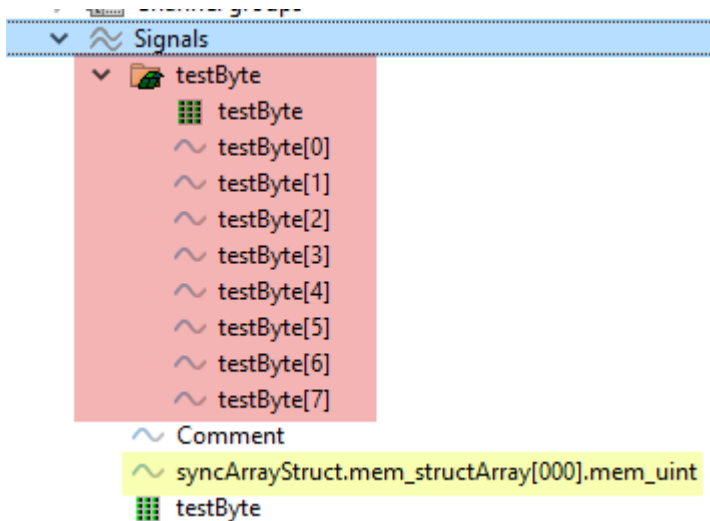
2.1.4 Example: single arrayelements [old database description method]

Now what happens if we don't measure the array or AoS as a whole. Looking at an array, which is defined as value block in the database, there is no way of measuring single elements on their own, the array is always measured whole. The array is added in the measurement for comparison.

Which leaves the AoS, which allows single elements to be measured. The measurement setup looks as followed:

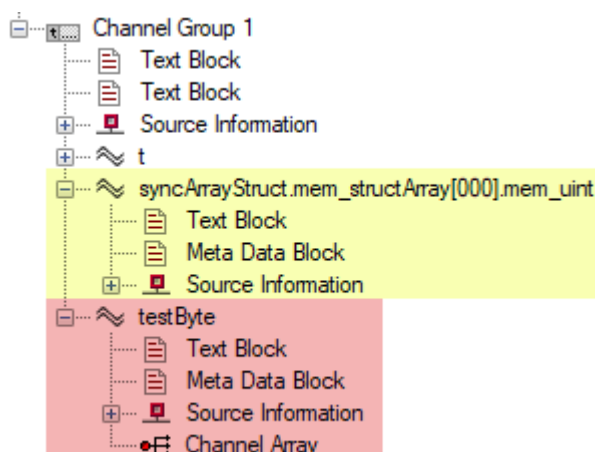
No.	Type	Acti...	Name
0		<input checked="" type="checkbox"/>	Comment
1		<input checked="" type="checkbox"/>	testByte
2		<input checked="" type="checkbox"/>	syncArrayStruct.mem_structArray[000].mem_uint

In the measurement file the elements are visualized as followed:



The array is still shown completely, the element from the AoS is shown on it's own as described in the database.

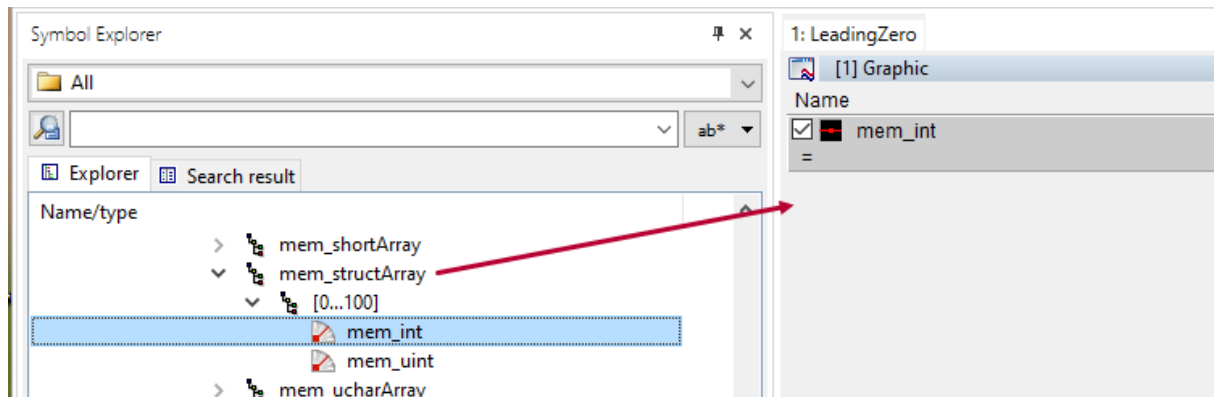
The result shows that the single measured AoS element has no channel array and is described outside it's structure:









2.1.5 Example: single arrayelements [new database description method - Typedef]

This example is similar to the one with the old database. However this will contain two different methods on how the elements were put into the graphic window.

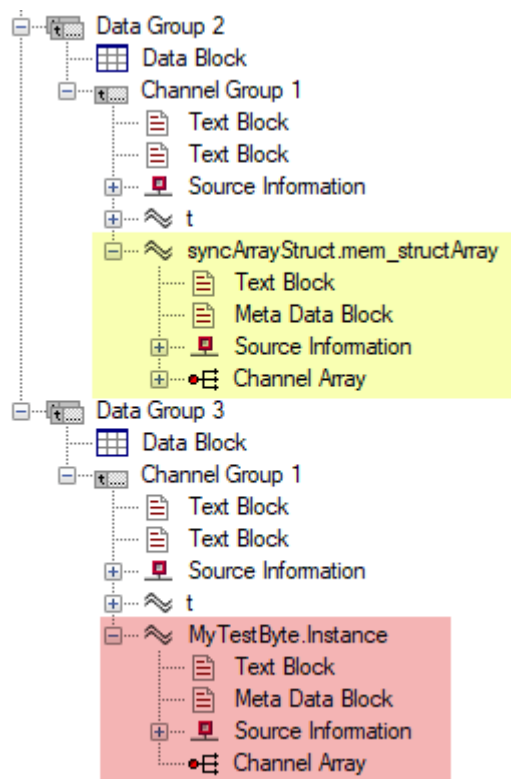
First, the variable is put into the graphic window by the structure:



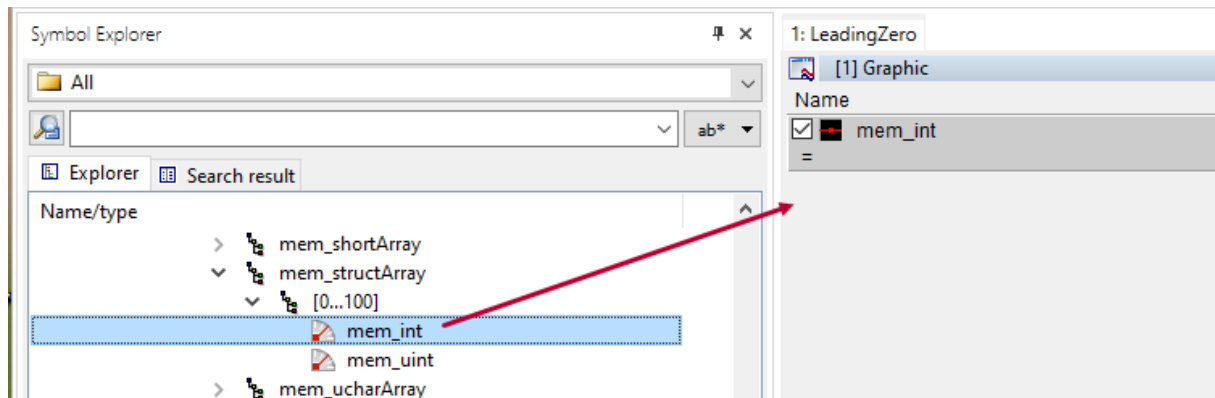
Then the measurement setup is as following, we can see that the structure and the array are set as whole structure:

No.	Type	Acti...	Name	Measurem...	Rate	Recorder
0		<input checked="" type="checkbox"/>	Comment	on input		<input checked="" type="checkbox"/> 
1		<input checked="" type="checkbox"/>	syncArrayStruct.mem_structArray	polling	100	<input checked="" type="checkbox"/> 
2		<input checked="" type="checkbox"/>	MyTestByte.Instance	100ms		<input checked="" type="checkbox"/> 

We get this result that both are measured completely even though we only selected one element for the graphic window visualization:



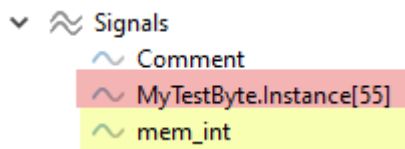
The second example is made with one difference, the arrayelements are pulled into the graphic window through the selection of the single element in the symbol explorer:



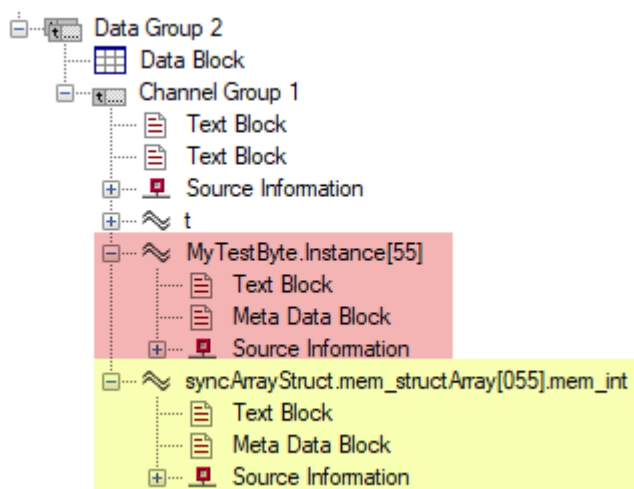
This one tiny change will result into a very different picture, first the entry in the measurement configuration:

No.	Type	Acti...	Name	Measurem...	Rate	Record
0		<input checked="" type="checkbox"/>	Comment	on input		<input checked="" type="checkbox"/>
1		<input checked="" type="checkbox"/>	mem_int	100ms		<input checked="" type="checkbox"/>
2		<input checked="" type="checkbox"/>	MyTestByte.Instance[55]	100ms		<input checked="" type="checkbox"/>

In the resulting measurement file the arrayelements are now listed on their own and the structure is completely lost:



Now the elements are really on their own in the measurement file which also leads to the result that the leading zero is present:



2.2 Creating a measurement file via “save signals”

When a measurement file is created through the “save signals” function the deciding factor, aside from the database, on how the array elements are saved into the file is the graphic window visualizing these variables.

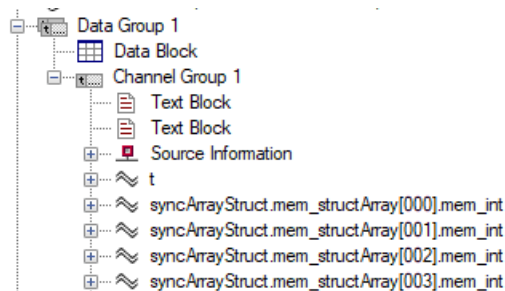
2.2.1 Example: Full array

When the full AoS [Typedef] is put into a graphic window all elements will be shown separately. Exactly the way they are visualized is the way they will be written into the measurement file:

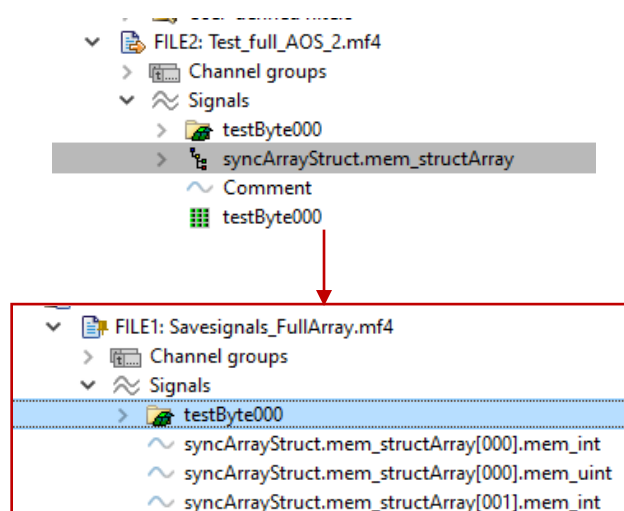
Symbol Explorer in CANape:

FILE1: Savesignals_FullArray.mf4	Savesignals_FullArray.mf4
Channel groups	
Signals	
testByte000	
syncArrayStruct.mem_structArray[000].mem_int	XCPsim
syncArrayStruct.mem_structArray[000].mem_uint	XCPsim
syncArrayStruct.mem_structArray[001].mem_int	XCPsim
syncArrayStruct.mem_structArray[001].mem_uint	XCPsim
syncArrayStruct.mem_structArray[002].mem_int	XCPsim
syncArrayStruct.mem_structArray[002].mem_uint	XCPsim
syncArrayStruct.mem_structArray[003].mem_int	XCPsim

MDF Validator:

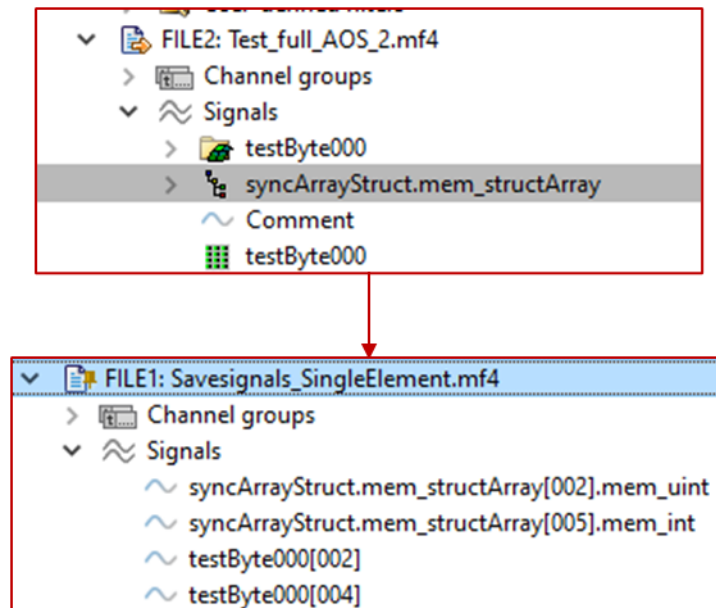


To summarize we went from a measurement file with a full AoS and an array to a measurement file with single AoS elements and an array:



2.2.2 Example: Single arrayelements

If single signals from the AoS and Array are selected and saved via “save signals” functionality only these single elements will be written into the resulting measurement file:



So the signals will now be saved in the measurement file completely without their structures.

2.3 Creating a MDF via “save file as”

This functionality will create an exact copy of the file this was used on.

3 Summary

Here is an overview of the previous topics:

	Method	Measurement file result	Leading Zero
Measurement (as in measurement configuration)	Full AoS in measurement configuration (not Typedef)	Overlaying structure with single element description	Yes
	Full Array (not Typedef)	Full Array with Channel Array	No
	Full AoS in measurement configuration (Typedef)	Full AoS with channel Array	No
	Full Array (Typedef)	Full Array with Channel Array	No
	Single AoS elements (no Typedef)	Single AoS element	Yes
	Arrayelement (no Typedef)	Full Array with Channel Array	No
	Single AoS elements (Typedef)	Single AoS element	Yes
	Arrayelement (Typedef)	Single Array element	Yes
Save Signals	Full AoS	Full AoS with channel Array	No
	Full Array	Full Array with Channel Array	No
	Single AoS element	Single AoS element	Yes
	Single Arrayelement	Single Array element	Yes
Save File As	Same as in previous measurement file.		

4 Working with MDF4lib

The MDF4 Lib creates the names of the array-elements (in a MDF4) synthetically because there is no template in the MDF-specification for that so it orientates itself on the CANape naming rules.

However with the MDF4 Lib V.1.9 there is a possibility to adapt the element names via config file. A further description can be found in the MDF4 Lib documentation "Advanced Topics\Configuration File"

In ASAM MDF4, there are no names defined for each array element, only the name for the array itself. Furthermore, there is no rule how an element's name shall be displayed with respect to its array indices.

MDF4 Lib always uses square brackets for each dimension to append the zero-based indices to the array name, i.e. A[0][3]. The index number by default uses the same naming rule as in Vector CANape, i.e. it can be preceded by leading zeros to allow easier alphabetic ordering of the channel elements by name. However, if required, it is possible to change this behavior with the following config file option in tag <names>:

Attribute name	Data type	Description	Possible values	Default value
arrayElementNamingRule	Integer	<p>Option (enum) how to name array elements, i.e. how to append the array index/indices to the base array name.</p> <p>Note: this only affects MDF4 files. For MDF3, the names of the array elements are explicitly stored in the MDF file.</p> <p>0 = omit leading zeros, i.e. for a 42x123 matrix called "A", the first member name now is "A[0][0]" and last one "A[42][123]"</p> <p>1 = add leading zeros (up to 4), i.e. for a 42x123 matrix called "A", the first member name now is "A[00][000]" and last one "A[42][123]", and for a 10x100 matrix called "M", the first member name now is "M[0][00]" and last one "M[9][99]"</p> <p>2 = add leading zeros like in Vector CANape (Default). This is similar to option 1, but for dimension size of 10, 100, 1000, 10000 one digit too much is used, i.e. for a 10x100 matrix called "M", the first member name now is "M[00][000]" and last one "M[09][099]"</p>	Integer number (0-2)	2 (i.e. use same rule as in Vector)

If the name of the array-element should remain the same but the intention is to access the element regardless, there is a way in the MDF Lib V1.9 as well: `IArrayInfo::GetElementChannel`.

```
virtual IChannel* IArrayInfo::GetElementChannel ( UINT64 *   pIndices,
                                                UINT16   dimCount,
                                                ErrorCode * pErrorCode = NULL
                                                )                [pure
                                                                virtual]
```

Get element channel of the array for a given position

Parameters:

[in] <i>pIndices</i>	array with zero-based indices (index for each dimension) to specify the desired element. Each index must be in range for respective dimension (see GetDimSize()).
[in] <i>dimCount</i>	size of elements in <i>pIndices</i> . Must be equal to number of dimensions in array (see GetDimCount()).
[out] <i>pErrorCode</i>	if not NULL the error code of the operation will be returned

Returns:

pointer to IChannel object

Please remember to call Release for the returned object or use a smart pointer (see Resource Management of Objects).

5 Contacts

For support related questions please address to the support contact for your country
<https://www.vector.com/int/en/company/contacts/support-contact/>.