```
using System;
using System.Iec61131Lib;
using Eclr;
using Eclr.Pcos;
using System.Runtime.InteropServices;
using Iec61131.Engineering.Prototypes.Types;
using Iec61131.Engineering.Prototypes.Variables;
using Iec61131.Engineering.Prototypes.Methods;
using Iec61131.Engineering.Prototypes.Common;

namespace ExampleLib
{
    // The IecArray must be defined as a struct with a fixed size.
    // The array definition MUST have following Attributes:
    // 1. Array (Actually only one-dimensional arrays are supported by the PCWorx Engineer)
    // 2. ArrayDimension
    // 3. DataType to define the data type of the array elements
    [Array(1), ArrayDimension(0, ArrayProperties.lowerBound, ArrayProperties.upperBound), DataType("WORD")]
    [StructLayout(LayoutKind.Explicit, Size = ArrayProperties.size)]
    public struct WordArray
    {
        // Helper containing constants to have a
        // clear and maintainable definition for boundaries and size
        struct ArrayProperties
        {
            public const int lowerBound = 0;
            public const int upperBound = 7167;
            // the size must be changed to the correct size of your elements times the amount of elements
            public const int size = (upperBound - lowerBound + 1) * sizeof(UInt16);
        }

        // Fields
        // The field "Anchor" defines the beginning of the array.
        [FieldOffset(0)]
        // The Anchor's data type is the child data type of the array
        public ushort Anchor;
        // The constants LB and UB define the upper and lower bound. Boundaries will be checked by using them.
        public const int LB = ArrayProperties.lowerBound;
        public const int UB = ArrayProperties.upperBound;
        public int this[int index]
        {
            get
            {
                if (index >= LB && index <= UB)
                {
                    unsafe
                    {
                        fixed (UInt16* pValue = &Anchor)
                        {
                            int result = *(pValue + index);
                            return result;
```

```
52                          }
53                      }
54                  }
55                  else
56                  {
57                      throw new IndexOutOfRangeException();
58                  }
59              }
60              set
61              {
62                  if (index >= LB && index <= UB)
63                  {
64                      unsafe
65                      {
66                          fixed (UInt16* pValue = &Anchor)
67                          {
68                              *(pValue + index) = (UInt16) value;
69                          }
70                      }
71                  }
72                  else
73                  {
74                      throw new IndexOutOfRangeException();
75                  }
76              }
77          }
78      }
79
80      [FunctionBlock]
81      public class FB_with_user_array
82      {
83          //[Input]
84          public WordArray iIN;
85          [InOut, DataType("ANY")]
86          public WordArray ModBusData;
87          [Input]
88          public int iMIN_VALUE;
89          [Output]
90          public int iMAX_VALUE;
91          [Output]
92          public int iAVERAGE;
93          [Output]
94          public bool xIS_DATA_CHANGED;
95
96          private WordArray Backup;
97
98          [Initialization]
99          public void __Init()
100         {
101
102
103
104         }
105
106         [Execution]
107         public unsafe void __Process()
```

```
108         {
109             var test = ModBusData;
110             iMAX_VALUE = 0;
111             iMIN_VALUE = int.MaxValue;
112 
113             int total = 0;
114 
115 
116             // This kind of accessing the array works fine with all types of
                  arrays but it is really cumbersome
117             // Nevertheless it can also be used for arrays of e.g. IecStrings.
118             fixed (UInt16* data = &ModBusData.Anchor)
119             {
120                 for (int i = 0; i < (WordArray.UB - WordArray.LB + 1); i++)
121                 {
122                     data[i] = (UInt16)i;
123                     UInt16 currentValue = data[i];
124                     total += currentValue;
125                     iMIN_VALUE = Math.Min(iMIN_VALUE, currentValue);
126                     iMAX_VALUE = Math.Max(iMAX_VALUE, currentValue);
127                 }
128             }
129 
130             // For arrays of elementary types the index operator can be used
                  effectively.
131             xIS_DATA_CHANGED = false;
132             for (int i = WordArray.LB; i <= WordArray.UB; i++)
133             {
134                 if (Backup[i] != ModBusData[i])
135                 {
136                     xIS_DATA_CHANGED = true;
137                 }
138             }
139             Backup = ModBusData;
140 
141             iAVERAGE = total / (WordArray.UB - WordArray.LB + 1);
142         }
143     }
144 }
```